Formal Verification of Safety-Critical Hybrid Systems

by

Carolos Livadas

S.M. in Aeronautics and Astronautics, MIT (1996)
S.B. in Computer Science and Engineering, MIT (1993)
S.B. in Aeronautics and Astronautics, MIT (1993)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

September 1997

Author ____
Department of Electrical Engineering and Computer Science
September 5, 1997

Certified by ____
Professor Nancy A. Lynch
Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ____
Professor Arthur C. Smith
Department of Electrical Engineering and Computer Science
Chairman, Department Committee on Graduate Theses

Formal Verification of Safety-Critical Hybrid Systems

by

Carolos Livadas

Submitted to the Department of Electrical Engineering and Computer Science on September 5, 1997, in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis investigates how the formal modeling and verification techniques of computer science can be used for the analysis of hybrid systems [7,14,22,37] — 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. As a case study, the thesis focuses on the development of techniques that use hybrid I/O automata [29, 30] to model and analyze automated vehicle transportation systems and, in particular, their various protection subsystems — control systems that are used to ensure that the physical plant at hand does not violate its various safety requirements.

The thesis is split into two major parts. In the first part, we develop an abstract model of a physical plant and its various protection subsystems — also referred to as *protectors*. The specialization of this abstract model results in the specification of a particular automated transportation system. Moreover, the proof of correctness of the abstract model leads to simple correctness proofs of the protector implementations for particular specializations of the abstract model. In this framework, the composition of independent protectors is straightforward — their composition guarantees the conjunction of the safety properties guaranteed by the individual protectors. In fact, it is shown that under certain conditions composition holds for dependent protectors also.

In the second part, we specialize the aforementioned abstract model to simplified versions of the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation. We examine overspeed and collision protection for a set of vehicles traveling on straight tracks, on binary merges, and on a directed graph of tracks involving binary merges and diverges. In each case, the protectors sample the state of the physical plant and take protective actions to guarantee that the physical plant does not reach hazardous states. The proofs of correctness of such protectors involve specializing the abstract protector to the physical plant at hand and proving that the suggested protector implementations are correct. This is done by defining simulations among the states of the protector implementations and their abstract counterparts.

Thesis Supervisor: Nancy A. Lynch, Ph.D.
Title: NEC Professor of Software Science and Engineering

Acknowledgments

The completion of the research leading to this thesis involved the continuous guidance and support of certain people. I would like to thank my advisor Prof. Nancy A. Lynch. Her knowledge and insight have been an invaluable asset to my research. Moreover, her thorough reviews and her wise suggestions have molded my research work into a clear and complete thesis. I would also like to thank Roberto Segala and Frits Vaandrager for bearing with my numerous questions on the HIOA model. Furthermore, I would like to thank John Lygeros for the insight and motivation he provided through numerous discussions on the modeling and verification of hybrid systems. I am also grateful to H. B. Weinberg for his initial work on modeling the PRT 2000™ and on his help on jump starting my involvement in the research leading to this thesis. Finally, I would like to thank my family for its continuous love and support and all my friends at MIT for making my student life interesting and enjoyable.

This research was performed at the Theory of Distributed Systems Group 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.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ATO	Automatic Train Operation
ATP	Automatic Train Protection
AVO	Automated Vehicle Operation
AVOS	Automated Vehicle Operation System
AVP	Automated Vehicle Protection
AVPS	Automated Vehicle Protection System
HIOA	Hybrid I/O Automaton, or Hybrid I/O Automata
PRT	Personal Rapid Transit

Hybrid I/O Automata Notation

(i, t)	A superdense time in an execution fragment $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$.
(i, t, s)	An occurrence of a state s in an execution fragment $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$.
α	A hybrid execution of a HIOA.
$\alpha.\mathit{fstate}$	The first state of a hybrid execution $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$.
$\alpha.\mathit{lstate}$	The last state of a finite hybrid execution $\alpha = w_0 a_1 w_1 \cdots a_n w_n$.
$\alpha.\mathit{ltime}$	The limit time of a hybrid execution $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$.
Σ	The set of all actions of a HIOA.
Σ^{int}	The set of internal actions of a HIOA.
Σ^{in}	The set of input actions of a HIOA.
Σ^{loc}	The set of locally controlled actions of a HIOA.
Σ^{out}	The set of output actions of a HIOA.
Θ	The set of initial states of a HIOA.
A	A hybrid I/O automaton.

E	The set of external variables of a HIOA.
L	The set of local variables of a HIOA.
U	The set of input variables of a HIOA.
V	The set of all variables of a HIOA.
X	The set of internal variables of a HIOA.
Y	The set of output variables of a HIOA.
$\mathcal{D}$	The set of discrete transitions of a HIOA.
$\mathcal{W}$	The set of trajectories of a HIOA.
$\textit{h-traces}(A)$	The set of hybrid traces of the HIOA A.
$\textit{h-trace}(\alpha)$	The hybrid trace of the hybrid execution α.
$\textit{states}(A)$	The set of all states of the HIOA A.

Latin Abbreviations

cf.	*confer*; Latin for "compare".
e.g.	*exempli gratia*; Latin for "for example".
et al.	*et alii*; Latin for "and others".
etc.	*et cetera*; Latin for "and so forth".
i.e.	*id est*; Latin for "that is".
ib. or *ibid.*	*ibidem*; Latin for "in the same work/place".
n.b.	*nota bene*; Latin for "take special note of".
op. cit.	*opere citato*; Latin for "in the work/text cited".
v.g.	*verbi gratia*; Latin for "for example".
v.i.	*vide infra*; Latin for "see below".
v.s.	*vide supra*; Latin for "see above".
viz.	*videlicet*; Latin for "that is to say" or "namely".
vs.	*versus*; Latin for "against".

Mathematical Notation

$\boldsymbol{Z}$	The set of valuations of the set of variables Z.
$\mathcal{V}$	The universal set of variables.
$\|X\|$	The cardinality of the set X.

$\overline{X}$	The complement of the set X.
$f \downarrow X$	The projection of the function f to the set X.
$f \downarrow y$	The projection of the function f to the element y.
$f \lceil X$	The restriction of the function f to the set X.
f, g, h	Functions.
T	The time axis, *i.e.*, a compact subgroup of $(\mathbb{R}, +)$.
T_I	An interval in the time axis, *i.e.*, a non-empty convex subset of T.
$w.fstate$	The first state of a trajectory w, *i.e.*, $w(0)$.
$w.lstate$	The last state of a closed trajectory w, *i.e.*, $w(w.ltime)$.
$w.ltime$	The limit time of a trajectory w, *i.e.*, $\sup(dom(w))$.
$X - Y$	The complement of the set Y in the set X, *i.e.*, $X \cap \overline{Y}$.
$X :\in S_X$	Assignment of an arbitrary element of the set of valuations S_X, where $S_X \subseteq \boldsymbol{X}$, to the set of variables X.
$x :\in X$	Assignment of an arbitrary element of the set of values X, where $X \subseteq type(x)$, to the variable x.
$X \cap Y$	The intersection of the sets X and Y.
$X \cup Y$	The union of the sets X and Y.
$X \setminus Y$	The complement of the set Y in the set X, *i.e.*, $X \cap \overline{Y}$.
$\mathcal{P}(X)$	The power set of the set X.
$dom(f)$	The domain of the function f.
$range(f)$	The range of the function f.
$trajs(Z)$	The collection of all trajectories over the set of variables Z.
$type(v)$	The domain over which the variable v ranges.
$\emptyset$	The empty (or null) set.
$\mathbb{N}^+$	The set of positive natural numbers, *i.e.*, the set $\{1, 2, 3, \ldots\}$.
$\mathbb{N}$	The set of natural numbers, *i.e.*, the set $\{0, 1, 2, 3, \ldots\}$.
$\mathbb{R}, \mathbb{R}^{\geq 0}, \mathbb{R}^+$	The set of all, non-negative, and positive real numbers.
$\mathbb{Z}, \mathbb{Z}^{\geq 0}, \mathbb{Z}^+$	The set of all, non-negative, and positive integers.

Physical Plant and Protector Notation

Δx_{max}	The maximum distance a vehicle can travel if braked after d_{max} time units.

$\dot{c}_{max}$	The maximum allowable velocity of any vehicle.
$\ddot{c}_{brake}$	The braking deceleration of a vehicle that has not collided.
$\ddot{c}_{max}$	The maximum acceleration of a vehicle that has not collided.
$\ddot{c}_{min}$	The minimum acceleration of a vehicle that has not collided.
$C_i(t)$	The section of track claimed by the vehicle i in time t.
c_{len}	The minimum allowable separation between vehicles.
d_{max}	The maximum protector sampling period.
E_i	The section of track occupied by the vehicle i, *i.e.*, the extent of the vehicle i.
O_i	The section of track owned by the vehicle i.

Chapter 1

Introduction

This thesis investigates how the formal modeling and verification techniques of computer science can be used for the analysis of hybrid systems [7,14,22,37] — 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. As a case study, the thesis focuses on the development of techniques that use hybrid I/O automata [29, 30] to model and analyze automated vehicle transportation systems and, in particular, their various protection subsystems — control systems that are used to ensure that the physical plant at hand does not violate its various safety requirements.

The thesis is split into two major parts. In the first part, we develop an abstract model of a physical plant and its various protection subsystems — also referred to as *protectors*. The specialization of this abstract model results in the specification of a particular automated transportation system. Moreover, the proof of correctness of the abstract model leads to simple correctness proofs of the protector implementations for particular specializations of the abstract model. In this framework, the composition of independent protectors is straightforward — their composition guarantees the conjunction of the safety properties guaranteed by the individual protectors. In fact, it is shown that under certain conditions composition holds for dependent protectors also.

In the second part, we specialize the aforementioned abstract model to simplified versions of the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation. We examine overspeed and collision protection for a set of vehicles traveling on straight tracks, on binary merges, and on a directed graph of tracks involving binary merges and diverges. In each case, the protectors sample the state of the physical plant and take protective actions to guarantee that the physical plant does not reach hazardous states. The proofs of correctness of such protectors involve specializing the abstract protector to the physical plant at hand and proving that the suggested protector implementations are correct. This is done by defining simulations among the states of the protector implementations and their abstract counterparts.

1.1 Hybrid Systems

The trend of system integration and automation has resulted in large and complex systems involving hierarchical and/or decentralized control structures. The higher levels of control are based on discrete algorithms and are often modeled using finite automata techniques from computer science. The lower levels of control address continuous behavior and are based on well established control theoretic techniques. The inherent complexity of the mix of continuous and discrete control and the need of a precise and efficient model for hybrid systems has encouraged research in this field.

The similarity of the hierarchical and/or decentralized control structure of hybrid systems with the distributed system setting in computer science has nurtured a distributed systems approach of analyzing hybrid systems. This approach is based on various formal modeling techniques developed for the verification and the proof of correctness of distributed systems in computer science. Such techniques use the principles of abstraction and modular decomposition to provide simple and concise models of complex systems. Once a particular system is decomposed into succinct parts, various composition theorems are used to prove that the system is functioning according to its specifications, *i.e.*, the system is *correct*.

1.1.1 Formal Framework

The formal modeling techniques that are used in this thesis are based on the *hybrid I/O automaton* model [29,30]. This model is an extension of the *timed I/O automaton* model [11, 34] and allows the explicit treatment of continuous behavior. The hybrid I/O automaton model is inspired by the phase transition models [2,4,35,36].

The hybrid I/O automaton model is a (possibly) infinite state model of a system involving both discrete and continuous behavior. The states of a hybrid I/O automaton (HIOA) are the valuations of a set of variables. The discrete behavior of a HIOA is modeled by discrete jumps in state which are described by labeled transitions. The labels of such transitions are the *actions* that carry out the transition from the initial to the final state of the jump. The continuous behavior of a HIOA is modeled by continuous changes in state which are described by sets of *trajectories*. The external interface of a HIOA is dictated by the partition of its variables and its actions into three categories: input, internal, and output. The behavior of the system being modeled over time is described by *hybrid executions* — finite or infinite alternating sequences of trajectories and actions. The externally visible part of a hybrid execution is denoted as the *hybrid trace* of the hybrid execution and involves the evolution of the input and output variables of the HIOA.

A HIOA A_1 implements another HIOA A_2 if every external behavior of A_1 is allowed by A_2. In this setting A_1 and A_2 are referred to as the *implementation* and the *specification*, respectively. The notion of an implementation relation is given by inclusion of the sets of hybrid traces; that is, the set of hybrid traces of A_1 is a subset of the set of hybrid traces of A_2. The composition of two HIOA is defined as their synchronization on shared input/output variables and input/output actions. Under straightforward and simple conditions, the composition of two HIOA results in a HIOA. Moreover, composition respects the implementation relation, *i.e.*, supposing B is a HIOA, if the HIOA A_1 implements the HIOA A_2, then the composition of A_1 with B implements the composition of A_2 with B.

Most of the proofs in the HIOA framework use *invariant assertions* and *simulations*. In the case of invariant assertions, the proofs are by induction on the length of a hybrid execution of the HIOA at hand. Such proofs show that a particular predicate on the state of the HIOA is satisfied in every state of the execution. A simulation is a mapping between the states of the two HIOA and is used to prove that one HIOA implements another. The fact that the mapping is indeed a simulation is again done by induction on the length of a hybrid execution of the implementation. This induction matches up individual steps in the implementation with either single steps, or sequences of steps, in the specification.

1.1.2 Related Work

The recent interest in the area of hybrid systems has resulted in a number of techniques to model and analyze their behavior. In particular, models that are analogous to the timed I/O automaton model [11,34] are the models of Alur and Dill [6], Lamport [20], and Henzinger, Manna, and Pnueli [18]. As is the case with the timed I/O automaton model, these models have also been extended to the hybrid setting; for instance, the timed transition model [18] has been extended to the phase transition model [35,36]. Phase transition systems are analogous to hybrid I/O automata — the transitions and the activities of phase transition systems correspond to the discrete transitions and the trajectories of hybrid I/O automata. The hybrid system model [2,4] is similar to the phase transition model with the distinction that, as in the hybrid I/O automaton model, discrete transitions are labeled, thus allowing the appropriate synchronization of composed automata. The distinction between the hybrid system model [2,4] and the hybrid I/O automaton model lies in the latter's classification of the discrete transitions and variables into input, internal, and output.

In the realm of applications, the formal modeling techniques presented above have been used for the analysis of various problems. The railroad crossing problem [15] and the steam boiler problem [1,21] comprise two commonly used benchmark problems. The former benchmark problem considers the control of a railroad gate that prevents cars and pedestrians from crossing the railroad tracks while the train is in the vicinity of the crossing. This gate must be lowered prior to the arrival of the train and lifted once the train has passed by. The latter benchmark problem involves the control of the level of water in a steam boiler.

The success in the modeling, analysis, and controller design for the above benchmark problems has encouraged the formal modeling of more complex hybrid systems; for example, automated transportation systems [41,42], industrial and chemical processes [9,40], rail-vehicle control [39], and complex automotive suspension systems [38]. The motivation behind such research lies mostly in the safety-critical nature of the systems at hand. In the case of automated transportation systems, the safety of the passengers has greatly encouraged the use of formal techniques.

The recent interest in addressing safety concerns related to automated highway systems and, in particular, the California PATH project [41], has resulted in a surge of hybrid system problems. The goal of PATH is to increase vehicle throughput by organizing traffic into platoons of closely spaced vehicles. Godbole, Lygeros, and Sastry [12, 13, 23, 25, 27] at U.C. Berkeley have studied various problems that arise in the control of the vehicle platoons. Such problems address the control of the leader of a platoon in view of following the preceding platoon at a safe distance, tracking an optimal cruising velocity, and performing various platoon maneuvers. The platoon maneuvers that have been addressed are the *platoon join*, in which two or more adjacent platoons join to form a single platoon, the *platoon split*, in which a platoon splits in two, and the *platoon lane change*. Lygeros [22] and Lygeros *et al.* [26, 27] used a game theoretic approach to prove that all platoon maneuvers are safe. Recently, Dolginova and Lynch [8] have used hybrid I/O automata to model and verify the safety of the platoon join maneuver.

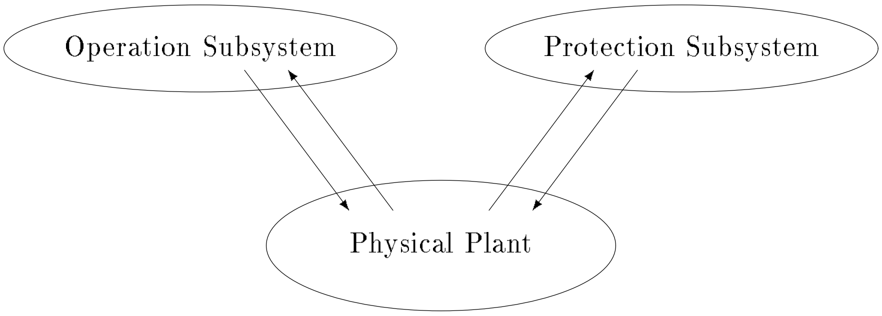
On a similar note, Weinberg [43] has analyzed a deceleration maneuver in which a discrete controller slows a train down to a target velocity range within a given distance. In further research, Weinberg *et al.* [42] have modeled the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation and verified the correct operation of the emergency control components used to guarantee that the vehicles neither exceed a prespecified speed limit, nor collide among themselves.

1.2 Automated Transportation Systems

Among the hybrid systems that are being analyzed using formal methods, systems in transportation are particularly common. This is due to the fact that such systems are safety-critical and, therefore, their correct analysis and verification is of uttermost importance.

An important feature of the design of the various autonomous transportation systems is

Figure 1.1 Separation of system functionality into operation and protection.



their absolute safety requirements. These requirements translate to stringent design criteria and have led to the complete separation of the system functionality into the parallel components of *operation* and *protection* as shown in Figure 1.1. The operation component is responsible for the "normal" control of the system and can be composed of complex software and hardware. The protection component is responsible for the "emergency" control of the system and is designed to be simple and reliable. In ordinary operation, the protection component is not supposed to take any action — it merely monitors the system. In a potentially hazardous situation, however, the protection component must react strongly enough to guarantee that, regardless of the behavior of the operation component, the safety requirements are met. In the interest of making the protection component reliable, designers keep it simple; instead of having complex control abilities, the protection component depends only on the correct execution of a few decisive emergency commands.

The separation of operation and protection functions is a generally recognized engineering paradigm for the design of safety-critical systems. In the realm of transportation systems, 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 (ATO) and Automatic Train Protection (ATP) 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 transportation 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.

1.2.1 The PRT 2000™

Raytheon engineers are currently working on the design and development of a new personal rapid transit (PRT) system called PRT 2000™. This system uses 4-passenger vehicles that travel on an elevated guideway with Y-shaped merges and diverges. Passengers on this system board at stations and travel directly to their desired destination stations without intermediate stops. Compared to conventional transportation systems, the PRT 2000™ 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.

Once again, the control of the PRT 2000™ is split into the Automated Vehicle Operation System (AVOS) and the Automated Vehicle Protection System (AVPS). The AVOS is in charge of the normal operation of the system and the AVPS is used to protect the system against hazards.

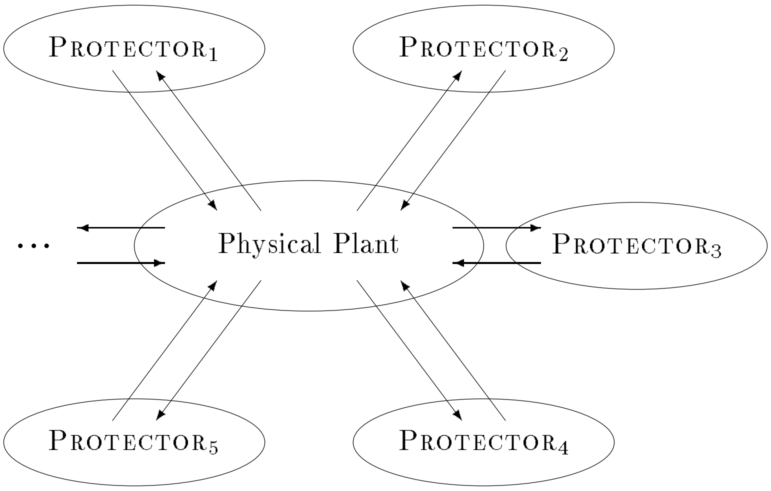
1.2.2 Formal Modeling of the PRT 2000™

The safety-critical nature of the PRT 2000™ has lead to an interest in modeling its protection system using formal modeling techniques from computer science. The advantage of using such modeling methods is twofold. First, they formalize the safety concerns addressed by the protection system and, second, they are used to prove the correctness of the protection system at hand. The safety properties that are addressed are those of overspeed and collision avoidance, *i.e.*, either the property that the vehicles comprising the system do not exceed the speed limit, or the property that they do not collide among themselves. These are by no means the only safety requirements enforced by the AVPS of the PRT 2000™, but they are among the most important and complex.

The approach to modeling this automated transportation system is based on abstraction and modular decomposition. Abstraction is used to mask all inessential implementation details from the model of the system. Modular decomposition is used either to model each of the safety properties in isolation, or to model a particular safety property as the conjunction of several less complex safety properties. As shown in Figure 1.2, the protection system is defined as the composition of a set of simpler modules referred to as *protectors*. The composition of all these protectors results in a protection system that enforces the conjunction of the safety properties enforced by the individual protectors being composed. For instance, in the case of a protection system that prevents the vehicles from exceeding the speed limit, each of the protectors would correspond to protection subsystems that prevent individual vehicles from exceeding the speed limit. However, their composition would constitute an overspeed protection system for all the vehicles.

This thesis extends the work by Weinberg, Lynch, and Delisle [42] on modeling the AVPS of the personal rapid transit system (PRT 2000™) under development at Raytheon Corpo-

Figure 1.2 Modular decomposition of the AVPS of the PRT 2000™.



ration. Weinberg *et al.* [42] model the PRT 2000™ as a transportation system where:

- vehicles are traveling on a single track,
- vehicle velocities are non-negative,
- vehicles can stop instantaneously, as if they could hit a brick wall,
- collisions among vehicles are pairwise,
- brakes are binary, *i.e.*, the braking of a particular vehicle results in a vehicle deceleration equal to a prespecified value,
- the acceleration is constrained to a particular range of values, and
- the vehicle brakes comprise monotonic system constraints, *i.e.*, the instruction of a vehicle to brake can never be revoked.

In addition to the above assumptions, the communication among the various subsystems of the PRT 2000™ is assumed to be reliable, periodic, and timely.

Weinberg *et al.* [42] verify the correctness of the overspeed and the collision protection subsystems. First, it is shown that the overspeed protector guarantees that none of the vehicles exceed the speed limit and that the collision protector prohibits vehicle collisions provided that none of the vehicles exceed the speed limit. Then using a one-way dependence protector composition theorem it is shown that the composition of the overspeed and collision protectors guarantees that the vehicles neither exceed the speed limit, nor collide among themselves. It should be noted that the model of the physical plant is simplified to the point that abrupt changes of the vehicle velocities, due to collisions for example, are not modeled. The advantage of this simplification is that the overspeed protector does not depend on the collision protector and, therefore, the one-way dependence protector composition theorem suffices. The disadvantage is that the simplified model might not be a truthful representation of the real physical plant.

In this thesis, we extend the protector composition results of Weinberg *et al.* [42] and relax their modeling assumptions about the PRT 2000™. Regarding the composition of protection systems, we present theorems that dictate the conditions under which the composition of independent, one-way dependent, and even two-way dependent protectors guarantees the conjunction of the safety properties guaranteed by the individual protectors being composed. Regarding the transportation system model, two of the aforementioned assumptions are relaxed. First, the constraint on the track topology is gradually relaxed from that of a single track to that of a general track topology involving a directed graph of tracks comprised of Y-shaped merges and diverges. Second, the monotonicity constraint on the instruction of the vehicles to brake is relaxed such that the instruction of a vehicle to brake may be revoked, provided the vehicle in question is out of risk. Moreover, in an effort to truthfully model the transportation system, we extend the model of the physical plant to allow vehicle collisions that can adversely affect the velocity and the acceleration of the vehicles involved in a collision. Thus, since collisions may cause instantaneous jumps in vehicle velocities, the overspeed protector must require that no collisions ever occur in the physical plant; that is, the overspeed and the collision protectors are two-way dependent. Subsequently, it is shown that the two-way dependence composition conditions are met by the proposed overspeed and collision protectors and that their composition results in a protection system that guarantees that the vehicles neither exceed the speed limit, nor collide among themselves.

1.3 Thesis Overview

In order for this thesis to be self contained, Chapter 2 gives a short and terse treatment of the hybrid I/O automaton model [30] and describes the conventions used in the specification of HIOA in this thesis. In order to facilitate the modeling of complex system properties, we introduce notation to allow the explicit restriction of the states of a hybrid I/O automaton to sets of states that are comprised of all states satisfying complex state properties of the HIOA. In Chapter 3, we present an abstract model of a physical plant interacting with various protection systems. Both the physical plant and the protection systems are modeled as hybrid I/O automata. Provided that protectors are independent, they can be composed and their composition guarantees the conjunction of the safety properties guaranteed by the individual protectors being composed. Under certain conditions, the same applies for the composition of protectors that rely on the correct operation of each other. The abstract protector is defined as the composition of a sensor automaton and a discrete controller automaton. The sensor samples the state of the physical plant at regular intervals of time and the discrete controller issues protective actions so as to guarantee that the physical plant exhibits a particular safety property.

In subsequent chapters, we present a simple model of the PRT 2000™ and introduce overspeed and collision protectors. This is done for increasingly complicated track topologies. First we consider a single track, then a Y-shaped merge, and, finally, a general track topology comprised of Y-shaped merges and diverges. Chapter 4 defines a system of n vehicles traveling on a single track and Chapters 5 and 6 define its overspeed and collision protectors. Chapter 7 extends the model of the physical plant to involve a Y-shaped merge and defines a collision protector for the new model. Chapter 8 augments the model of the physical plant to involve a general track topology comprised of Y-shaped merges and diverges and defines a collision protector for the new model. In Chapter 9, we prove that the overspeed and collision protectors of the various track topologies can be composed so as to guarantee that the vehicles neither exceed the speed limit, nor collide among themselves. Finally, in Chapter 10 we give a summary of the thesis, an evaluation of the research presented, and directions in which such research could be extended or continued.

Chapter 2

Hybrid I/O Automata

The hybrid I/O automaton (HIOA) model [29, 30] is based on the timed I/O automaton model [10, 11, 33, 34], but includes explicit treatment of continuous behavior. To make this thesis self contained, this chapter gives a complete but terse treatment of the HIOA model with an emphasis on those aspects used in subsequent chapters. The presentation follows precisely that of Lynch, Segala, Vaandrager, and Weinberg [30].

The chapter is organized as follows. We begin by defining auxiliary concepts and notation pertaining to *functions*, *time*, *variables*, *valuations*, and *trajectories*. We proceed to define *hybrid I/O automata*, *hybrid executions*, and *hybrid traces*. Next, we define a *simulation* relation between a pair of HIOA and the notion of HIOA *composition*. Finally, we describe the conventions used in the specification of HIOA in this thesis. In particular, we describe how states, discrete transitions, and trajectories of a HIOA are specified and how to explicitly restrict the states of a HIOA in view of enforcing complex state properties.

2.1 Preliminary Mathematical Notation

This section defines various auxiliary concepts and notation that are used in the definition of the hybrid I/O automaton model.

Functions

With $dom(f)$ and $range(f)$ we denote the domain and the range, respectively, of the function f. If f is a function and X a set, then we write $f \lceil X$ for the *restriction* of f to X, *i.e.*, the function g with $dom(g) = dom(f) \cap X$ satisfying $g(x) = f(x)$, for all $x \in dom(g)$. We say that two functions f and g are compatible if $f \lceil dom(g) = g \lceil dom(f)$. If f and g are compatible functions, then we write $f \cup g$ for the function h with $dom(h) = dom(f) \cup dom(g)$ such that $h(x) = f(x)$, if $x \in dom(f)$, and $h(x) = g(x)$, otherwise, for all $x \in dom(h)$. More generally, if F is a set of pairwise compatible functions then we write $\bigcup_{f \in F} f$ for the unique function g with $dom(g) = \bigcup_{f \in F} dom(f)$ such that $g(x) = f(x)$, for all $f \in F$ and $x \in dom(f)$. If f is a function whose range consists of a set of functions and X is a set, then the *projection* $f \downarrow X$ is the restriction of the functions in $range(f)$ to the set X, *i.e.*, the function g with $dom(g) = dom(f)$ defined by $g(x) \triangleq f(x)\lceil X$, for all $x \in dom(g)$. The projection operator $\downarrow$ extends to sets of functions by pointwise extension. Also, if f is a function whose range consists of a set of functions that all have an element y in their domain, then the *projection* $f \downarrow y$ is the function with domain $dom(f)$ defined by $f \downarrow y(x) \triangleq f(x)(y)$, for all $x \in dom(f)$.

Time

Throughout this thesis, we fix the *time axis* T to be a compact subgroup of $(\mathbb{R}, +)$, *i.e.*, the real numbers with addition. Henceforth, we exclusively use the set of real numbers $\mathbb{R}$ as the time axis. An *interval* T_I is a non-empty convex subset of T. As usual, intervals are denoted by $[t_1, t_2] = \{t \in T \mid t_1 \leq t \leq t_2\}$, *etc.* An interval T_I is right-open (left-open), if it does not have a maximum (minimum) element, and right-closed (left-closed), otherwise. We write $\max(T_I)$ and $\min(T_I)$ for the maximum and the minimum elements, respectively, of the interval T_I (if they exist), and $\sup(T_I)$ and $\inf(T_I)$ for the supremum and infimum, respectively, of the interval T_I in $T \cup \{-\infty, \infty\}$. For $T' \subseteq T$ and $t \in T$, we define $T' + t \triangleq \{t' + t \mid t' \in T'\}$. Thus, for a function f with domain T', we define $f + t$ to be the function with domain $T' + t$ satisfying $f + t(t') = f(t' - t)$, for all $t' \in T' + t$.

Variables and Valuations

We assume a universal set $\mathcal{V}$ of *variables*. Variables in $\mathcal{V}$ are typed, where the *type* of a variable, such as *reals*, *integers*, *etc.* is given by $type(v)$; that is, $type(v)$ is the domain over which the variable v ranges. Letting $V \subseteq \mathcal{V}$, a *valuation* of V is a function that associates to each variable v of V a value in $type(v)$. We adopt the convention that the set of all valuations of a set of variables V is denoted by $\boldsymbol{V}$. Often, valuations of a set of variables V are referred to as *states*.

Letting $v \in \mathcal{V}$ 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 $V \subseteq \mathcal{V}$ and $S_V \subseteq \boldsymbol{V}$, 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.

Let Z be a set of variables, z be a state of Z, and Z' be a subset of Z, *i.e.*, $Z \subseteq \mathcal{V}$, $z \in \boldsymbol{Z}$, and $Z' \subseteq Z$. The *restriction* of the state z to the set of variables Z', denoted by $z\lceil Z'$, is defined to be the valuation z' of the variables of Z' in z. Letting $\boldsymbol{X} \subseteq \boldsymbol{Z}$, we say that $\boldsymbol{X}$

is Z'-determinable if for all $x \in \boldsymbol{X}$ and $z \in \boldsymbol{Z}$, such that $x \lceil Z' = z \lceil Z'$, it is the case that $z \in \boldsymbol{X}$. Intuitively, if $\boldsymbol{X}$ is Z'-determinable then, for any state z in $\boldsymbol{Z}$, the information provided by the restriction of the state z to the set of variables Z' is sufficient to determine whether the state z is a member of the set $\boldsymbol{X}$. In other words, the information provided by the restriction of the state z to the set of variables $Z - Z'$ is irrelevant in the determination of whether the state z is a member of the set $\boldsymbol{X}$. Moreover, if $\boldsymbol{X}$ is Z'-determinable and $z' \in \boldsymbol{Z'}$, we use the notation $z' \in \boldsymbol{X}$ to denote that there exists a state $x \in \boldsymbol{X}$ such that $x \lceil Z' = z'$. In fact, since $\boldsymbol{X}$ is Z'-determinable, the existence of a state $x \in \boldsymbol{X}$ such that $x \lceil Z' = z'$ implies that for all states $z \in \boldsymbol{Z}$ such that $z \lceil Z' = z'$ it is the case that $z \in \boldsymbol{X}$.

Trajectories

A *trajectory* over a set of variables Z is a function $w : T_I \to \boldsymbol{Z}$, where T_I is a left-closed interval of T with left endpoint equal to 0. A trajectory represents the evolution of the valuations of the variables in Z within a T_I interval. With $dom(w)$ we denote the domain of w and with $trajs(Z)$ the collection of all trajectories over Z. A trajectory w with domain T_I is often referred to as a T_I-trajectory.

A trajectory w is *closed*, if its domain is a (finite) right-closed interval, and *full*, if its domain equals $T^{\geq 0}$. For W a set of trajectories, $Closed(W)$ and $Full(W)$ denote the subsets of closed and full trajectories in W, respectively. If w is a trajectory, then the *limit time* of w, denoted by $w.ltime$, is defined to be the supremum of $dom(w)$. A trajectory w is finite if $w.ltime \neq \infty$. 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 trajectory with domain $[0, 0]$ is called a *point* trajectory. If s is a state, then we define $\wp(s)$ to be the point trajectory that maps 0 to s.

For a trajectory w and $t \in T^{\geq 0}$, we define $w \trianglelefteq t \triangleq w \lceil [0, t]$ and $w \triangleleft t \triangleq w \lceil [0, t)$. It is important to note that $w \triangleleft 0$ is not a trajectory. By convention, $w \trianglelefteq \infty \triangleq w \triangleleft \infty \triangleq w$. Similarly, if w is a trajectory and T_I is a left-closed interval with $\min(T_I) \in dom(w)$, then we define the *curtailment* of w to T_I, denoted by $w \dagger T_I$, to be the trajectory $(w \lceil T_I) - \min(T_I)$, or equivalently the trajectory w' with domain $(T_I \cap dom(w)) - \min(T_I)$ defined by $w'(t') = w(t' + \min(T_I))$, for all $t' \in dom(w')$.

If w is a trajectory over Z and $Z' \subseteq Z$, then the *projection* $w \downarrow Z'$ is the trajectory over Z' with domain $dom(w)$ defined by $w \downarrow Z'\ (t)(z') \triangleq w(t)(z')$, for all $z' \in Z'$. The projection operation is extended to sets of trajectories by pointwise extension. Also, if w is a trajectory over Z and $z \in Z$, then the *projection* $w \downarrow z$ is the function from $dom(w)$ to the domain of z defined by $w \downarrow z\ (t) \triangleq w(t)(z)$.

If w is a finite trajectory with domain T_I, w' is a trajectory with domain T_I', and $w.lstate =$

$w'.fstate$ if w is closed, then we define the *concatenation* of w and w' to be the trajectory $w \frown w' \triangleq w \cup (w' + w.ltime)$. We extend the concatenation operator to an infinite sequence of finite trajectories $w_0 w_1 w_2 \cdots$. If $w_i.lstate = w_{i+1}.fstate$, for each trajectory pair w_i and w_{i+1}, for $i \in \mathbb{N}$, in which the trajectory w_i is closed, then we define the *infinite concatenation* of the infinite sequence of finite trajectories $w_0 w_1 w_2 \cdots$ to be the trajectory $w_0 \frown w_1 \frown w_2 \ldots \triangleq \bigcup_{i,j \in \mathbb{N}} (w_i + \sum_{j<i} w_j.ltime)$.

A trajectory w is a *prefix* of a trajectory w', denoted by $w \leq w'$, if $w = w' \lceil dom(w)$; that is, either $w = w'$, or $w' = w \frown w''$, for some trajectory w''. With $Pref(W)$ we denote the *prefix-closure* of W: $Pref(W) \triangleq \{w \mid \exists\, w' \in W : w \leq w'\}$. A set W is *prefix closed* if $W = Pref(W)$. A trajectory in W is *maximal* if it is not a prefix of any other trajectory in W. We write $Max(W)$ for the subset of maximal trajectories in W.

2.2 The Hybrid I/O Automaton Model

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

- Three disjoint sets U, X, and Y of variables, called *input*, *internal*, and *output* variables, respectively.

  Variables in $E \triangleq U \cup Y$ are called *external*, and variables in $L \triangleq X \cup Y$ are called *local*. We write $V \triangleq U \cup L$ and let s, u, and w range over $\boldsymbol{V}$, $\boldsymbol{U}$, and $trajs(V)$, respectively.

- Three disjoint sets Σ^{in}, Σ^{int}, and Σ^{out} of *actions*, called *input*, *internal*, and *output* actions, respectively.

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

- A non-empty set $\Theta \subseteq \boldsymbol{V}$ of *initial states* satisfying:

  Init (initial states closed under change of input variables)
  $$s \in \Theta \implies \exists\, s' \in \Theta \;:\; (s' \lceil U = u) \,\wedge\, (s' \lceil Y = s \lceil Y)$$

- A set $\mathcal{D} \subseteq \boldsymbol{V} \times \Sigma \times \boldsymbol{V}$ of *discrete transitions* satisfying:

  D1 (input action enabling)
  $$a \in \Sigma^{in} \implies \exists\, s' \in \boldsymbol{V} \;:\; s \xrightarrow{a}_A s'$$

D2 (environment actions that do not change inputs do not affect the state)
$(s \xrightarrow{e}_A s') \wedge (s\lceil U = s'\lceil U) \implies (s = s')$

D3 (discrete transitions do not depend on input variable changes)
$(s \xrightarrow{a}_A s') \implies \exists\, s'' \in \boldsymbol{V} \,:\, (s \xrightarrow{a}_A s'') \wedge (s''\lceil U = u) \wedge (s''\lceil Y = s'\lceil Y)$

For any discrete transition (s, a, s') of the automaton A, *i.e.*, $(s, a, s') \in \mathcal{D}$, the states s and s' are referred to as the *pre-state* and *post-state*, respectively, of the discrete transition (s, a, s'). Moreover, as in the above treatment, we often use the notation $s \xrightarrow{a}_A s'$ to denote that (s, a, s') is a discrete transition of the automaton A, *i.e.*, $(s, a, s') \in \mathcal{D}$.

- A set $\mathcal{W}$ of trajectories over V satisfying:

T1 (existence of point trajectories)
$\wp(s) \in \mathcal{W}$

T2 (closure under subintervals)
$w \in \mathcal{W} \wedge (T_I$ left-closed subinterval of $dom(w)) \implies w \dagger T_I \in \mathcal{W}$

T3 (completeness)
$(\forall\, t \in T^{\geq 0} : w \dagger [0, t] \in \mathcal{W}) \implies w \in \mathcal{W}$

The intuition captured by Axioms **Init** and **D1–D3** is that a HIOA is responsible for performing locally controlled actions and for modifying the values of its local variables, whereas the environment of a HIOA is responsible for performing input actions and modifying the values of the input variables.

Axiom **Init** says that a system may not constrain the initial values of its input variables. Thus, if we change the input variables of an initial state, then there is a way to change the internal variables as well (while leaving the output variables unchanged) so that the resulting state is an initial state also.

Axiom **D1**, which is simply the hybrid extension of the input enabling axiom from the (untimed) I/O automaton model [11, 32, 34], says that a HIOA should accept all input actions in all states. Axiom **D2** postulates that an environment action that does not affect the input variables can *not* be "detected" by the automaton and, therefore, leaves the state unchanged. Axiom **D3** states that there is no functional dependence between the input and the output variables of a HIOA during a transition; that is, a HIOA can *not* react instantaneously to an input variable change. If there is an a-step from a state s to a state s', then, for any valuation u of the input variables, there also exists an a-step from s to a state s'' with an input part u and an output part equal to that of s'. The internal variables of s' and s'' need not have the same values, since otherwise it would not be possible for a HIOA to record all the discrete changes in its input variables. The technical use of Axiom **D3** is to avoid cyclic constraints during the interaction of two systems. In this way, we can show that the composition of two HIOA is still input enabled and that the environment can never block the output actions of a system.

Axioms **D2** and **D3** imply that the environment action e can never affect the output variables of a HIOA. Consider any transition $(s, e, s') \in \mathcal{D}$ and suppose that $s' \lceil Y \neq s \lceil Y$. Letting $u = s \lceil U$, Axiom **D3** implies that there exists $s'' \in \boldsymbol{V}$ such that $(s, e, s'') \in \mathcal{D}$, $s'' \lceil U = s \lceil U$, and $s'' \lceil Y = s' \lceil Y$. Since $s'' \lceil U = s \lceil U$ and $s'' \lceil Y \neq s \lceil Y$, Axiom **D2** is violated. Therefore, it follows that there does not exist $(s, e, s') \in \mathcal{D}$ such that $s' \lceil Y \neq s \lceil Y$.

Axioms **T1**–**T3** state some natural conditions on the set of trajectories needed to set up our theory: existence of point trajectories, closure under subintervals, and the fact that a full trajectory is in $\mathcal{W}$ if and only if all its prefixes are in $\mathcal{W}$.

The Axiom **Init** and the Axioms **D1**–**D3** that are presented here are slightly different from the respective axioms introduced in the preliminary version of the HIOA model [29]. The new axioms allow a HIOA to change the values of its internal variables if the environment modifies the input variables of the HIOA.

Notation Let A be a HIOA as described above. If $s \in \boldsymbol{V}$ and $l \in \boldsymbol{L}$, then we write $s \xrightarrow{a}_A l$ if and only if there exists an $s' \in \boldsymbol{V}$ such that $s \xrightarrow{a}_A s'$ and $s' \lceil L = l$. Henceforth, the components of a HIOA A will be denoted by V_A, U_A, Σ_A, Θ_A, *etc.* Moreover, the components of a HIOA A_i will also be denoted by V_i, U_i, Σ_i, Θ_i, *etc.*

2.3 Hybrid Executions

A *hybrid execution fragment* α of a HIOA A is a finite or infinite alternating sequence $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$, where:

1. Each w_i is a trajectory in $\mathcal{W}_A$ and each a_i is an action in Σ_A.
2. If α is a finite sequence then it ends with a trajectory.
3. If w_i is not the last trajectory in α then its domain is a right-closed interval and it is the case that $w_i.\mathit{lstate} \xrightarrow{a_{i+1}}_A w_{i+1}.\mathit{fstate}$.

An execution fragment records all the discrete changes that occur in the evolution of a system, plus the "continuous" state changes that take place in between. The third item says that the discrete actions in α span between successive trajectories. We write *h-frags*(A) for the set of all hybrid execution fragments of A.

If $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment, then we define the *limit time* of α, denoted by $\alpha.\mathit{ltime}$, to be $\sum_{i \in \mathbb{N}} w_i.\mathit{ltime}$. Further, we define the *first state* of α, denoted by $\alpha.\mathit{fstate}$, to be $w_0.\mathit{fstate}$.

We distinguish several sorts of hybrid execution fragments. A hybrid execution fragment α is defined to be

- an *execution* if the first state of α is an initial state, *i.e.*, $\alpha.fstate \in \Theta_A$,
- *finite* if α is a finite sequence and the domain of its final trajectory is a right-closed interval,
- *admissible* if $\alpha.ltime = \infty$,
- *Zeno* if α is neither finite nor admissible, and
- a *sentence* if α is a finite execution that ends with a point trajectory.

If $\alpha = w_0 a_1 w_1 \cdots a_n w_n$ is a finite hybrid execution fragment then we define the *last state* of α, denoted by $\alpha.lstate$, to be $w_n.lstate$. A state of A is defined to be *reachable* if it is the last state of some finite hybrid execution of A.

A finite hybrid execution fragment $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots a_n w_n$ and a hybrid execution fragment $\alpha' = w'_0 a'_1 w'_1 a'_2 w'_2 \cdots$ of A can be *concatenated* if $w_n \frown w'_0$ is defined and is a trajectory of A. In this case, the *concatenation* $\alpha \frown \alpha'$ is the hybrid execution fragment defined by

$$\alpha \frown \alpha' \triangleq w_0 a_1 w_1 a_2 w_2 \cdots a_n (w_n \frown w'_0) a'_1 w'_1 a'_2 w'_2 \cdots$$

Let α and α' be hybrid execution fragments of a HIOA A. We say that α' is a *prefix* of α', denoted by $\alpha' \leq \alpha$, if either $\alpha' = \alpha$, or there exists some execution fragment α'' of A such that $\alpha' \frown \alpha'' = \alpha$.

A variable v of a HIOA A is called *continuous* if v is not modified by any discrete steps of A and for all trajectories w of A, $w \downarrow v$ is a continuous function. Let $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ be a hybrid execution fragment of A. Then we define $\alpha \downarrow v$ as follows:

$$\alpha \downarrow v = (w_0 \downarrow v) \frown (w_1 \downarrow v) \frown (w_2 \downarrow v) \ldots$$

Theorem 2.3.1 *If v is a continuous variable of a HIOA A and α is an execution fragment of A, then $\alpha \downarrow v$ is a continuous function.*

If $\alpha = w_0 a_1 w_1 a_2 w_2 \ldots$ is a hybrid execution fragment of a HIOA A and $Z \subseteq V$ then $\alpha \downarrow Z$ is defined to be the sequence $(w_0 \downarrow Z) a_1 (w_1 \downarrow Z) a_2 (w_2 \downarrow Z) \ldots$.

A *superdense time* in an execution fragment $\alpha = w_0 a_1 w_1 a_2 w_2 \ldots$ of a HIOA 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 $\alpha = w_0 a_1 w_1 a_2 w_2 \ldots$ of a HIOA A is a triple (i, t, s) such that (i, t) is a superdense time in α and $s = w_i(t)$. We order state occurrences in α according to the order of their superdense times.

If S is a set of states and α is an execution fragment, 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 $s' \in S$.

2.4 Hybrid Traces

Suppose $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment of A. In order to define the *hybrid trace* of α, let

$$\gamma = (w_0 \downarrow E_A) vis(a_1)(w_1 \downarrow E_A) vis(a_2)(w_2 \downarrow E_A) \cdots,$$

where, for any action a of A, $vis(a)$ is defined equal to τ if a is an internal action or an environment action e, and equal to a otherwise. Here τ is a special symbol which, as in the theory of process algebra, plays the role of the "generic" invisible action. An occurrence of τ in γ is called *inert* if the final state of the trajectory that precedes the τ equals the first state of the trajectory that follows it (after hiding of the internal variables). The *hybrid trace* of α, denoted by $h\text{-}trace(\alpha)$, is defined to be the sequence obtained from γ by removing all inert τ's and concatenating the surrounding trajectories.

The *hybrid traces* of A are the hybrid traces that arise from all the finite and admissible hybrid executions of A. We write $h\text{-}traces(A)$ for the set of hybrid traces of A.

The 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 mean 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.

2.5 Auxiliary HIOA Definitions

Given a HIOA A, we use the notation $states(A)$ to denote the state space of the automaton A, *i.e.*, $states(A) = \boldsymbol{V_A}$. If R is a subset of the set of states $states(A)$ of the automaton A and $s, s' \in R$, then we say that s' is *R-reachable from* s, denoted by $s \rightsquigarrow_R s'$, 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. We say that s' is *reachable from* s, denoted by $s \rightsquigarrow s'$, provided that s' is R-reachable from s, where R is the set of all states of the automaton A, *i.e.*, $R = states(A)$.

When analyzing a HIOA A, it is often useful to define *derived* variables for A. Such variables are functionally dependent on the variables of the automaton A and, although useful in the analysis of A, are not essential in its definition.

If s is a state of a HIOA A and z is a variable of A, *i.e.*, $s \in \mathit{states}(A)$ and $z \in V_A$, then $s.z$ denotes the value of the variable z in the state s. In terms of valuations, $s.z$ is the restriction of the valuation s to the element z, *i.e.*, $s.z = s \lceil z$.

If f is a function to states of a HIOA A and Z is a subset of the variables of A, *i.e.*, $\mathit{range}(f) = \mathit{states}(A)$ and $Z \subseteq V_A$, then $f \downarrow Z$ is the *projection* of f onto the variables in Z, *i.e.*, the function g with domain $\mathit{dom}(f)$ and range equal to the set of valuations of Z, defined by: $g(s)(z) = f(s)(z)$, for all $s \in \mathit{dom}(f)$ and $z \in Z$. In the special case where Z is a singleton set $\{z\}$, *i.e.*, $Z = \{z\}$, we write $f \downarrow z$ as shorthand for $f \downarrow Z$.

2.6 Simulation Relations

Let A and B be comparable HIOA. A *simulation* from A to B is a relation $R \subseteq \boldsymbol{V_A} \times \boldsymbol{V_B}$ satisfying the following conditions, for all states r and s of A and B, respectively:

1. If $r \in \Theta_A$, then there exists $s \in \Theta_B$ such that $r\ R\ s$.
2. If $r \xrightarrow{a}_A r'$ and $r\ R\ s$, then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{h\text{-}trace}(\wp(r)\ a\ \wp(r')) = \mathit{h\text{-}trace}(\alpha)$, and $r'\ R\ \alpha.\mathit{lstate}$.
3. If $r\ R\ s$ and w is a closed trajectory of A with $r = w.\mathit{fstate}$, then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{h\text{-}trace}(w) = \mathit{h\text{-}trace}(\alpha)$, and $w.\mathit{lstate}\ R\ \alpha.\mathit{lstate}$.

Theorem 2.6.1 *If A and B are comparable HIOA and there is a simulation from A to B, then $A \leq B$.*

2.7 Composition

We say that the HIOA A_1 and A_2 are *compatible* if, for $i, j \in \{1, 2\}, i \neq j$,

$$X_i \cap V_j = Y_i \cap Y_j = \Sigma_i^{int} \cap \Sigma_j = \Sigma_i^{out} \cap \Sigma_j^{out} = \emptyset.$$

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

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$
- $\Sigma^{in} = (\Sigma_1^{in} \cup \Sigma_2^{in}) - (\Sigma_1^{out} \cup \Sigma_2^{out})$, $\Sigma^{int} = \Sigma_1^{int} \cup \Sigma_2^{int}$, $\Sigma^{out} = \Sigma_1^{out} \cup \Sigma_2^{out}$

- $\Theta = \{s \in \boldsymbol{V} \mid s\lceil V_1 \in \Theta_1 \ \wedge \ s\lceil V_2 \in \Theta_2\}$

- Define, for $i \in \{1,2\}$, projection function $\pi_{A_i} : \Sigma \to \Sigma_i$ by $\pi_{A_i}(a) \triangleq a$, if $a \in \Sigma_i$, and $\pi_{A_i}(a) \triangleq e$, otherwise. Then $\mathcal{D}$ is the subset of $\boldsymbol{V} \times \Sigma \times \boldsymbol{V}$ given by

$$(s, a, s') \in \mathcal{D} \iff s\lceil V_1 \xrightarrow{\pi_{A_1}(a)}_{A_1} s'\lceil V_1 \ \wedge \ s\lceil V_2 \xrightarrow{\pi_{A_2}(a)}_{A_2} s'\lceil V_2$$

- $\mathcal{W}$ is the set of trajectories over V given by

$$w \in \mathcal{W} \iff w \downarrow V_1 \in W_1 \ \wedge \ w \downarrow V_2 \in W_2$$

Notation We extend the projection notation π_{A_i}, for $i \in \{1,2\}$, to states, trajectories, discrete actions, hybrid executions, and hybrid traces in the obvious way. If s, w, and a are a state, a trajectory, and a discrete action of the automaton $A = A_1 \times A_2$, then the respective projections π_{A_i}, for $i \in \{1,2\}$, are defined as $\pi_{A_i}(s) = s\lceil V_A$, $\pi_{A_i}(w) = w \downarrow V_A$, and $\pi_{A_i}(a) = a$ if $a \in \Sigma_A$ and $\pi_{A_i}(a) = e$ otherwise. Also, if $\alpha = w_0 a_1 w_1 a_2 \cdots$ is a hybrid execution of the automaton $A = A_1 \times A_2$, then the projection π_{A_i}, for $i \in \{1,2\}$, is defined as $\pi_{A_i}(\alpha) = \pi_{A_i}(w_0)\pi_{A_i}(a_1)\pi_{A_i}(w_1)\pi_{A_i}(a_2)\cdots$. Moreover, if γ is the hybrid trace of α, then $\pi_{A_i}(\gamma)$ is the sequence obtained from $(w_0 \downarrow E_{A_i}) vis_{A_i}(a_1)(w_1 \downarrow E_{A_i}) vis_{A_i}(a_2)(w_2 \downarrow E_{A_i}) \cdots$ by removing all inert τ's and concatenating the surrounding trajectories.

Proposition 2.7.1 *If A_1 and A_2 are compatible HIOA, then their composition $A_1 \times A_2$ is a HIOA.*

Lemma 2.7.2 *Let $A = A_1 \times A_2$, and let α be a hybrid execution of A. Then it is the case that $\pi_{A_i}(h\text{-}trace(\alpha)) = h\text{-}trace(\pi_{A_i}(\alpha))$, for $i \in \{1,2\}$.*

Lemma 2.7.3 *Let $A = A_1 \times A_2$. Then it is the case that $h\text{-}traces(A) = \{\gamma \mid \pi_{A_i}(\gamma) \in h\text{-}traces(A_i),\ for\ i \in \{1,2\}\}$.*

Theorem 2.7.4 *Suppose A_1, A_2, and B are HIOA with $A_1 \leq A_2$, and each of A_1 and A_2 is compatible with B. Then $A_1 \times B \leq A_2 \times B$.*

2.8 HIOA Specification Conventions

In this section we describe the conventions used in the specification of a HIOA A in this thesis. In particular, we describe how the states, the discrete transitions, and the trajectories of A are specified and introduce notational shorthand used to specify concisely complex state properties of A.

2.8.1 State Specification

Since the states of A are the set of valuations of its variable set V_A, the states of A are specified by simply defining the domain over which each variable in V_A ranges. Thus, the states of A are specified by a list of all input, internal, and output variables together with the domain over which each respective variable ranges. Similarly, the set of start states of A is specified by stating the set of values that each variable in V_A can initially assume. It is important to note that, by Axiom **Init** of the HIOA model, each input variable u of A may initially assume any value in $type(u)$; that is, the set of values that each input variable u of A can initially assume is the set $type(u)$.

2.8.2 Discrete Transition Specification

The set of discrete transitions of A is specified by collectively describing all discrete transitions involving each action a in Σ_A in *precondition-effect* format. This format is comprised of a *label*, a *precondition*, and an *effect clause*. The *label* corresponds to the label of the action a. The *precondition* is a predicate over the variables of A and specifies the conditions under which the action a is enabled; that is, the precondition defines the set of states in which the action a may be scheduled. It is important to note that an action a in Σ_A is not necessarily scheduled whenever it is enabled. The *effect clause* specifies the pseudo-code that must be applied to the pre-state of a discrete transition involving the action a so as to yield the post-state of the discrete transition. It follows that, in order for (s, a, s') to be a discrete transition of A, the precondition in the specification of the action a must be satisfied by the pre-state s. Moreover, the application of the pseudo-code in the effect clause of the specification of the action a to the pre-state s must yield the post-state s'.

The convention used in this thesis is that for any particular discrete transition (s, a, s') of A, the statements in the pseudo-code of the effect clause of the specification of a are applied sequentially to the state of A starting from the pre-state s. However, the effect clause in the specification of any action a of A is assumed to be executed indivisibly. Therefore, the execution of the action a in the state s represents a single transition from the pre-state s to the post-state s'. In order to be able to write effect clause pseudo-code involving the valuation of the variables of A in the pre-state, we adopt the convention that the value of a particular variable v of A in the pre-state s may be referred to as v_{pre}. Similarly, the value of the variable v in the post-state s' may be referred to as v_{post}.

Throughout this thesis, we adopt the convention that if the effect clause in the specification of an action a of A does not affect a local variable v, for any $v \in L_A$, the value of v in the post-state of any discrete transition involving the action a is equal to its value in the pre-state, *i.e.*, $v_{post} = v_{pre}$. Moreover, in order to conform to Axiom **D3** of the HIOA model, we adopt the convention that the effect clause in the specification of each action a of A must assign to each input variable u of A an arbitrary value in *type*(u); that is, the effect clause in the specification of the action a must include the assignment statement $u :\in$ *type*(u). In fact, we adopt the convention that such assignments precede any other statements in the effect clause of the specification of the action a. Obviously, if the automaton A has no input variables, *i.e.*, $U_A = \emptyset$, no such assignments are specified.

Axiom **D1** of the HIOA model defines HIOA to be *input-enabled*; that is, a HIOA is not capable of *blocking* the scheduling of its input actions. It follows that, each input action a of A is enabled in each state s of A. A consequence of this characteristic is that the precondition in the specification of each input action a of A is the trivial predicate `True`. Throughout this thesis, we adopt the convention that the precondition clause in the specification of any input action a of A is omitted; that is, the specification of each input action a of A is only comprised of the label and the effect clause of the action a.

The environment action e, which is considered an input action, allows the occurrence of a discrete transition in the external environment that is unobservable by A except (possibly) through its effect on the input variables of A. Environment actions are considered input actions because HIOA have no control over their external environment and, therefore, environment actions are enabled in all states. Thus, following the convention for input actions, the precondition clause in the specification of the environment action e is omitted. Moreover, according to Axioms **D2** and **D3** of the HIOA model, a discrete transition involving the environment action e can only affect the input and the internal variables of A. In fact, according to Axioms **D2** of the HIOA model, a discrete transition involving the environment action e can affect the internal variables of A only if the input variables are also affected. Therefore, the effect clause in the specification of the environment action e must be such that the internal variables are affected only if the valuation of the input variables in the post-state differs from their valuation in the pre-state; that is, for all $(s, e, s') \in \mathcal{D}$, it is the case that if $s\lceil X_A \neq s'\lceil X_A$ then $s\lceil U_A \neq s'\lceil U_A$. If the automaton A has no input variables, then the environment action e cannot affect its state; that is, if $U_A = \emptyset$ then for all $(s, e, s') \in \mathcal{D}$ it is the case that $s = s'$. In such cases, the environment action e is referred to as *stuttering* and the effect clause in its specification is comprised of the single statement "None". Often, when the environment action e does not affect the internal variables of a HIOA, or when the environment action e is stuttering, its specification is omitted. Thus, if the environment action e is omitted from the specification of a HIOA A, then it follows that the environment action e assigns arbitrary values to the input variables of A and does not affect the internal variables of A. Obviously, when the HIOA A has no input variables, the environment action e omitted in the specification of A is stuttering.

2.8.3 Trajectory Specification

The set of trajectories of A is specified by pseudo-code which describes the properties that any trajectory w involving the variables in the variable set of A must satisfy in order to be a trajectory of A. Thus, the trajectory pseudo-code consists of a collection of predicates all of which must be satisfied throughout any trajectory w of A. Since HIOA have no control over their input variables, the trajectory specification of A must not constrain its input variables. Thus, we adopt the convention that the trajectory specification of A includes a clause for each input variable u of A stating that the input variable u assumes arbitrary values in $type(u)$ throughout each trajectory w of A. Obviously, if the automaton A has no input variables, *i.e.*, $U_A = \emptyset$, no such clauses are specified. In contrast to the convention used in the specification of actions, if a particular local variable v of A is not constrained in the trajectory specification of A, then its value may assume arbitrary values in $type(v)$. Therefore, in order to specify that the value of the local variable v of A remains constant throughout each trajectory w of A, an explicit statement stating so must be used.

2.8.4 State Restriction

In the specification of a HIOA, it is often unwieldy to explicitly enforce complex state properties. In view of this specification inefficacy, we allow the enforcement of state properties through the restriction of the states of a HIOA to *property* sets. A property set P of A is a set of states of A that is comprised of all the states of A that satisfy a particular state property. The state property described by the property set P may be enforced through the use of "subject to P" clauses in the specification of either the initial states, the actions, or the trajectories of A. In the specification of the initial states of A, a "subject to P" clause signifies that all of the initial states of A are in the set P. In the specification of the actions of A, a "subject to P" clause in the effect clause of an action a signifies that the post-state of each discrete transition involving the action a is in the set P. Finally, in the specification of the trajectories of A, a "subject to P" clause signifies that all the states involved in each trajectory of A are in the set P. In the case of trajectories, such a clause may be interpreted as choosing the local variables of A that are unconstrained by the trajectory specification so that the states involved in the trajectory are in the set P.

Often, we collectively specify all complex state properties of a HIOA A using a single property set. This property set is distinct for each HIOA and is referred to as the set $VALID$ for the particular HIOA at hand.

Chapter 3

Abstract Physical Plant and Protector Models

This chapter is split into two parts. In the first part, we define an abstract model of a *physical system* that is comprised of a *physical plant* and a *protection system*. The protection system is modeled as a set of *protectors* that are communicating with the physical plant through distinct communication channels, or *ports*. These channels are used to sample and to control the state of the physical plant. Both the physical plant and the protectors are modeled as HIOA. It is shown that under certain conditions protectors can be composed such that their composition ensures the safety properties guaranteed by the individual protectors being composed. In the second part, we give an abstract model of a protector. The model is parameterized by the physical plant and various sets of states of the physical plant which describe the properties assumed and guaranteed by the abstract protector. The protector is defined as the composition of a sensor automaton and a discrete controller automaton. The sensor automaton samples the output state of the physical plant at a given sampling rate. The discrete controller automaton determines which protective action must be scheduled in order to ensure the safety of the physical plant up to the next sampling point. To conclude, the proposed abstract protector is shown to be correct.

3.1 Protected Plant Systems

In this section, we present an abstract model of a system consisting of a *physical plant* and a set of *protectors*. The model is abstract in that it does not specify any of the details of the physical plant — for instance, it does not specify that the plant includes vehicles and tracks. We also define what it means for a protector responsible for guaranteeing a particular property, *i.e.*, a protector used to avoid a particular mishap, to be correct.

3.1.1 Physical Plant Automata

Let J be a set of *ports*. A *physical plant automaton PP for J* is defined to be a hybrid I/O automaton (HIOA) in which:

1. The input action set Σ_{PP}^{in} is partitioned into subsets $\Sigma_{PP_j}^{in}$, one for each port j.
2. The output action set Σ_{PP}^{out} is partitioned into subsets $\Sigma_{PP_j}^{out}$, one for each port j.
3. The input variable set U_{PP} is partitioned into subsets U_{PP_j}, 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.

3.1.2 Protector Automata

Let PP be a physical plant automaton with port set J, and let $K \subseteq J$. A *protector automaton* A for the physical plant PP and the port set K is a HIOA that is compatible with PP, and that satisfies the following conditions:

1. Its output actions are exactly the input actions of PP on ports in K.
2. Its output variables are exactly the input variables of PP on ports in K.
3. All its input actions and input variables are outputs of PP.

Lemma 3.1.1 *Suppose that A_1 and A_2 are protectors for PP, with respective port sets K_1 and K_2, where $K_1 \cap K_2 = \emptyset$. If A_1 and A_2 are compatible then their composition $A_1 \times A_2$ is a protector for PP with port set $K_1 \cup K_2$.*

Proof: Since A_1 and A_2 are compatible, Proposition 2.7.1 implies that $A_1 \times A_2$ is a HIOA. Moreover, since A_1 and A_2 are compatible with PP it follows that $A_1 \times A_2$ is compatible with PP also. Therefore, it remains to be shown that the HIOA $A_1 \times A_2$ satisfies the three protector conditions presented above.

To begin, since the protectors A_1 and A_2 communicate with the plant PP through the port sets K_1 and K_2, respectively, their composition $A_1 \times A_2$ communicates with the plant PP through the port set $K_1 \cup K_2$. Therefore, there are three conditions to check:

1. The output actions of $A_1 \times A_2$ are exactly the input actions of PP on ports in $K_1 \cup K_2$.

   Since, the HIOA A_1 and A_2 are protectors, it is the case that their output actions are exactly the input actions of PP on the port sets K_1 and K_2, respectively. However, from the composition of the protectors A_1 and A_2, it is the case that $\Sigma_{A_1 \times A_2}^{out} = \Sigma_{A_1}^{out} \cup \Sigma_{A_2}^{out}$. Therefore, it trivially follows that the output actions of $A_1 \times A_2$ are exactly the input actions of PP on ports in $K_1 \cup K_2$.

2. The output variables of $A_1 \times A_2$ are exactly the input variables of *PP* on ports in $K_1 \cup K_2$.

   Since, the HIOA A_1 and A_2 are protectors, it is the case that their output variables are exactly the input variables of *PP* on the port sets K_1 and K_2, respectively. However, from the composition of the protectors A_1 and A_2, it is the case that $Y^{out}_{A_1 \times A_2} = Y^{out}_{A_1} \cup Y^{out}_{A_2}$. Therefore, it trivially follows that the output variables of $A_1 \times A_2$ are exactly the input variables of *PP* on ports in $K_1 \cup K_2$.

3. All the input actions and input variables of $A_1 \times A_2$ are outputs of *PP*.

   From the composition of the protectors A_1 and A_2, it is the case that $\Sigma^{in}_{A_1 \times A_2} = (\Sigma^{in}_{A_1} \cup \Sigma^{in}_{A_2}) - (\Sigma^{out}_{A_1} \cup \Sigma^{out}_{A_2})$ and $U_{A_1 \times A_2} = (U_{A_1} \cup U_{A_2}) - (Y_{A_1} \cup Y_{A_2})$. However, since the HIOA A_1 and A_2 are protectors, their output actions and output variables are inputs to the *PP* automaton. Therefore, it is the case that $\Sigma^{in}_{A_1 \times A_2} = \Sigma^{in}_{A_1} \cup \Sigma^{in}_{A_2}$ and $U_{A_1 \times A_2} = U_{A_1} \cup U_{A_2}$. It trivially follows that the input actions and input variables of $A_1 \times A_2$ are outputs of *PP*.

■

3.1.3 Protected Plant Systems

A *protected plant system* is the composition of a physical plant automaton *PP* and a set of protector automata. If s is a state of a protected plant system and P is a subset of the states of *PP*, we often write $s \in P$ as shorthand for $s \lceil PP \in P$. That is, we extend the definition of the set P to include states of the protected plant system that project to give *PP* states in P.

3.1.4 Substitutive and Compositional Correctness

Let S, R, and G be particular sets of states of *PP*. We say that 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. It is important to note that the first state of every such finite execution is in the set $\Theta_{PP \times A} \cap S$. In the special case where R is the set of all states of *PP*, we sometimes omit explicit mention of R. Moreover, we often omit mention of *PP* when the physical plant automaton is clear from context.

It is important to note that the definition of "guarantees" includes consideration of finite executions in which arbitrary inputs can arrive at *PP* on ports other than those in K. The protector definition infers that regardless of what inputs occur on those ports, the protector A still guarantees G in *PP* starting from S given R.

The following *substitutivity* theorem states that an implementation of a correct protector is itself a correct protector.

Theorem 3.1.2 *Let A_1 and A_2 be two protector automata for the same port set K, 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.*

Proof: Let $\alpha_{PP \times A_1}$ be any finite execution of the automaton $PP \times A_1$ that starts in a state in the set S and is restricted to states in the set R. We must show that $\pi_{PP}(\alpha_{PP \times A_1}.lstate) \in G$.

Let α_{PP} be the projection of $\alpha_{PP \times A_1}$ to the PP automaton and α_{A_2} be a finite execution of A_2 such that $h\text{-}trace(\alpha_{A_2}) = h\text{-}trace(\pi_{A_1}(\alpha_{PP \times A_1}))$. Adding environment actions appropriately to α_{PP} and α_{A_2}, we obtain two new finite executions $\alpha'_{PP} = w_0^{PP} a_1^{PP} w_1^{PP} a_2^{PP} w_2^{PP} \cdots$ and $\alpha'_{A_2} = w_0^{A_2} a_1^{A_2} w_1^{A_2} a_2^{A_2} w_2^{A_2} \cdots$ of PP and A_2, respectively, such that $w_i^{PP}.ltime = w_i^{A_2}.ltime$, for all $i \in \mathbb{N}$, and either $a_i^{PP} = a_i^{A_2}$, or $a_i^{PP} = e$ or $a_i^{A_2} = e$, for all $i \in \mathbb{N}^+$. The addition of environment actions to α_{PP} and α_{A_2} is intended to generate two new finite executions α'_{PP} and α'_{A_2} of PP and A_2, respectively, in which the limit times of the trajectories in α'_{PP} and α'_{A_2} are equal, the actions in α_{PP} and α_{A_2} shared by PP and A_2 appear in both hybrid executions α'_{PP} and α'_{A_2}, the internal actions of PP and the input actions of PP on ports other than port j appear as environment actions in α'_{A_2}, and the internal actions of A_2 appear as environment actions in α'_{PP}. Also, it is important to note that all the environment actions added to α_{PP} and α_{A_2} to obtain α'_{PP} and α'_{A_2}, respectively, correspond to inert τ's and do not appear in the hybrid traces $h\text{-}trace(\alpha'_{PP})$ and $h\text{-}trace(\alpha'_{A_2})$, i.e., $h\text{-}trace(\alpha'_{PP}) = h\text{-}trace(\pi_{PP}(\alpha_{PP \times A_1}))$ and $h\text{-}trace(\alpha'_{A_2}) = h\text{-}trace(\pi_{A_1}(\alpha_{PP \times A_1}))$.

Let $\alpha_i = w_0 a_1 w_1 a_2 w_2 \cdots a_i w_i$, for some $i \in \mathbb{N}$, be a finite hybrid execution comprised of a collection $w_0, w_1, w_2, \ldots, w_i$ of trajectories of $PP \times A_2$ and a collection $a_1, a_2, \ldots, a_i$ of actions of $PP \times A_2$, such that:

1. $\alpha_i = w_0 a_1 w_1 a_2 w_2 \cdots a_i w_i$ is a hybrid execution of $PP \times A_2$,
2. $\pi_{PP}(\alpha_i) = w_0^{PP} a_1^{PP} w_1^{PP} a_2^{PP} w_2^{PP} \cdots a_i^{PP} w_i^{PP}$, and
3. $\pi_{A_2}(\alpha_i) = w_0^{A_2} a_1^{A_2} w_1^{A_2} a_2^{A_2} w_2^{A_2} \cdots a_i^{A_2} w_i^{A_2}$.

By induction on the length i of the finite execution α_i, we show the existence of α_i, for all $i \in \mathbb{N}$, and, moreover, the existence of a finite execution $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ of $PP \times A_2$ comprised of a collection $w_0, w_1, w_2, \ldots$ of trajectories of $PP \times A_2$ and a collection $a_1, a_2, \ldots$ of actions of $PP \times A_2$, such that $\pi_{PP}(\alpha) = \alpha'_{PP}$ and $\pi_{A_2}(\alpha) = \alpha'_{A_2}$.

For the base case, consider the finite execution $\alpha_0 = w_0$ of length 0. Since $h\text{-}trace(\alpha'_{PP}) = h\text{-}trace(\pi_{PP}(\alpha_{PP \times A_1}))$, $h\text{-}trace(\alpha'_{A_2}) = h\text{-}trace(\pi_{A_1}(\alpha_{PP \times A_1}))$, and $w_0^{PP}.ltime = w_0^{A_2}.ltime$, it follows that $w_0^{PP}(t)(z) = w_0^{A_2}(t)(z)$, for all $z \in E_{PP} \cap E_{A_2}$ and $t \in [0, w_0^{PP}.ltime]$. Thus, the valuations of w_0^{PP} and $w_0^{A_2}$ are compatible, for all $t \in [0, w_0^{PP}.\mathit{ltime}]$, and the trajectory w_0 with domain $[0, w_0^{PP}.\mathit{ltime}]$ can be defined as $w_0 = w_0^{PP} \cup w_0^{A_2}$. By the definition of w_0, it follows that $\pi_{PP}(w_0) = w_0^{PP}$ and $\pi_{A_1}(w_0) = w_0^{A_2}$. Moreover, these two conditions imply that w_0 is a hybrid execution of $PP \times A_2$.

For the inductive step, assuming that the finite execution α_k satisfies the Properties 1, 2, and 3, for $i = k$, we must show that there exists a finite execution α_{k+1} that satisfies the Properties 1, 2, and 3, for $i = k+1$. Let $a_{k+1} = a_{k+1}^{PP}$, if $a_{k+1}^{PP} \neq e$, and $a_{k+1} = a_{k+1}^{A_2}$, otherwise. Since $\textit{h-trace}(\alpha'_{PP}) = \textit{h-trace}(\pi_{PP}(\alpha_{PP\times A_1}))$, $\textit{h-trace}(\alpha'_{A_2}) = \textit{h-trace}(\pi_{A_1}(\alpha_{PP\times A_1}))$, and $w_{i'}^{PP}.\mathit{ltime} = w_{i'}^{A_2}.\mathit{ltime}$, for all $i' \in \mathbb{N}$, it follows that $(w_0^{PP} a_1^{PP} w_1^{PP} \cdots a_k^{PP} w_k^{PP} a_{k+1}^{PP}$ $\wp(w_{k+1}^{PP}.\mathit{fstate})).\mathit{ltime} = (w_0^{A_2} a_1^{A_2} w_1^{A_2} \cdots a_k^{A_2} w_k^{A_2} a_{k+1}^{A_2} \wp(w_{k+1}^{A_2}.\mathit{fstate})).\mathit{ltime}$ and $w_{k+1}^{PP}(t)(z) = w_{k+1}^{A_2}(t)(z)$, for $z \in E_{PP} \cap E_{A_2}$ and $t \in [0, w_{k+1}^{PP}.\mathit{ltime}]$. Thus, the valuations of w_{k+1}^{PP} and $w_{k+1}^{A_2}$ are compatible, for all $t \in [0, w_{k+1}^{PP}.\mathit{ltime}]$, and the trajectory w_{k+1} with domain $[0, w_{k+1}^{PP}.\mathit{ltime}]$ can be defined as $w_{k+1} = w_{k+1}^{PP} \cup w_{k+1}^{A_2}$. By the definition of a_{k+1} and w_{k+1} it follows that $\pi_{PP}(\wp(w_k.\mathit{lstate}) a_{k+1} w_{k+1}) = \wp(w_k^{PP}.\mathit{lstate}) a_{k+1}^{PP} w_{k+1}^{PP}$ and $\pi_{A_2}(\wp(w_k.\mathit{lstate})$ $a_{k+1} w_{k+1}) = \wp(w_k^{A_2}.\mathit{lstate}) a_{k+1}^{A_2} w_{k+1}^{A_2}$. Thus, from the induction hypothesis it follows that the finite hybrid execution $\alpha_{k+1} = w_0 a_1 w_1 a_2 w_2 \cdots a_k (w_k \frown \wp(w_k.\mathit{lstate})) a_{k+1} w_{k+1} = w_0 a_1 w_1$ $a_2 w_2 \cdots a_{k+1} w_{k+1}$ satisfies the conditions $\pi_{PP}(\alpha_{k+1}) = w_0^{PP} a_1^{PP} w_1^{PP} \cdots a_{k+1}^{PP} w_{k+1}^{PP}$ and $\pi_{A_2}(\alpha_{k+1}) = w_0^{A_2} a_1^{A_2} w_1^{A_2} \cdots a_{k+1}^{A_2} w_{k+1}^{A_2}$. Moreover, these two conditions imply that the hybrid execution α_{k+1} is a hybrid execution of $PP \times A_2$, as needed.

From the above induction, it follows that there exists a hybrid execution α of $PP \times A_2$ such that $\pi_{PP}(\alpha) = \alpha'_{PP}$ and $\pi_{A_2}(\alpha) = \alpha'_{A_2}$. However, recall that the execution α'_{PP} of PP is derived from the execution $\pi_{PP}(\alpha_{PP\times A_1})$ by adding environment actions which correspond to inert τ's and do not appear in the hybrid trace of α'_{PP}. Therefore, the execution α'_{PP} of PP starts in a state in S and is restricted to states is R and, moreover, $\alpha'_{PP}.\mathit{lstate} = \pi_{PP}(\alpha_{PP\times A_1}.\mathit{lstate})$. Finally, since A_2 guarantees G in PP from S given R it follows that $\alpha'_{PP}.\mathit{lstate} \in G$. Moreover, since $\alpha'_{PP}.\mathit{lstate} = \pi_{PP}(\alpha_{PP\times A_1}.\mathit{lstate})$, it is the case that $\pi_{PP}(\alpha_{PP\times A_1}.\mathit{lstate}) \in G$, as needed. ■

We end this section with several *compositional* theorems for protectors. The first two theorems consider the composition of two or more independent protectors. The third theorem considers the composition of two protectors, one of which depends on the other; that is, a one-way protector dependency. The fourth and fifth theorems consider the composition of two or more protectors that depend on each other; that is, two-way and multiple-way protector dependencies.

Theorem 3.1.3 *Suppose that A_1 and A_2 are protector automata for PP, with respective port sets K_1 and K_2, where $K_1 \cap K_2 = \emptyset$. Suppose that A_1 guarantees G_1 from S_1 given R_1 and A_2 guarantees G_2 from S_2 given R_2. If the protectors A_1 and A_2 are compatible, then their composition $A_1 \times A_2$ is a protector that guarantees $G_1 \cap G_2$ from $S_1 \cap S_2$ given*

$R_1 \cap R_2$.

Proof: Let α be any finite execution of the HIOA $PP \times A_1 \times A_2$ that starts in a state in $S_1 \cap S_2$ and whose states are restricted to the set $R_1 \cap R_2$. Moreover, let α_{A_1} be the projection of α to the HIOA $PP \times A_1$, *i.e.*, $\alpha_{A_1} = \pi_{PP \times A_1}(\alpha)$. Since the execution α starts in a state in $S_1 \cap S_2$ and is restricted to the states in $R_1 \cap R_2$, the same applies to the projected execution α_{A_1}. However, since A_1 guarantees G_1 from S_1 given R_1, $S_1 \cap S_2 \subseteq S_1$, and $R_1 \cap R_2 \subseteq R_1$, it follows that all reachable states of PP in α_{A_1} are in G_1. Since α_{A_1} is the projection of α to the automaton $PP \times A_1$, it follows that all reachable states of PP in α are in G_1 also.

Taking a similar projection of the execution α to the automaton $PP \times A_2$, the desired result follows. ■

Theorem 3.1.4 *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for 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 that guarantees $\bigcap_{i \in \{1,\ldots,k\}} G_i$ from $\bigcap_{i \in \{1,\ldots,k\}} S_i$ given $\bigcap_{i \in \{1,\ldots,k\}} R_i$.*

Proof: Let α be any finite execution of the HIOA $PP \times \prod_{i \in \{1,\ldots,k\}} A_i$ that starts in a state in $\bigcap_{i \in \{1,\ldots,k\}} S_i$ and whose states are restricted to the set $\bigcap_{i \in \{1,\ldots,k\}} R_i$. Moreover, let $\alpha_{A_{i'}}$ be the projection of α to the HIOA $PP \times A_{i'}$, for some $i' \in \{1, \ldots, k\}$, *i.e.*, $\alpha_{A_{i'}} = \pi_{PP \times A_{i'}}(\alpha)$. Since the execution α starts in a state in $\bigcap_{i \in \{1,\ldots,k\}} S_i$ and is restricted to the states in $\bigcap_{i \in \{1,\ldots,k\}} R_i$, the same applies to the projected execution $\alpha_{A_{i'}}$. However, since $A_{i'}$ guarantees $G_{i'}$ from $S_{i'}$ given $R_{i'}$, $\bigcap_{i \in \{1,\ldots,k\}} S_i \subseteq S_{i'}$, and $\bigcap_{i \in \{1,\ldots,k\}} R_i \subseteq R_{i'}$, it follows that all reachable states of PP in $\alpha_{A_{i'}}$ are in $G_{i'}$. Since $\alpha_{A_{i'}}$ is the projection of α to the automaton $PP \times A_{i'}$, it follows that all reachable states of PP in α are in $G_{i'}$ also.

Taking similar projections of the execution α to each of the automata $PP \times A_{i''}$, for all $i'' \in \{1, \ldots, k\}$, the desired result follows. ■

Theorem 3.1.5 *Suppose that A_1 and A_2 are protector automata for PP, with respective port sets K_1 and K_2, where $K_1 \cap K_2 = \emptyset$. Suppose that A_1 guarantees G_1 from S_1 given R_1 and A_2 guarantees G_2 from S_2 given $R_2 \cap G_1$. If the protectors A_1 and A_2 are compatible, then their composition $A_1 \times A_2$ is a protector that guarantees $G_1 \cap G_2$ from $S_1 \cap S_2$ given $R_1 \cap R_2$.*

Proof: Let α be any finite execution of the HIOA $PP \times A_1 \times A_2$ that starts in a state in $S_1 \cap S_2$ and whose states are restricted to the set $R_1 \cap R_2$. Moreover, let α_{A_1} be the projection of α to the HIOA $PP \times A_1$, *i.e.*, $\alpha_{A_1} = \pi_{PP \times A_1}(\alpha)$. Since the execution α starts in a state in $S_1 \cap S_2$ and is restricted to the states in $R_1 \cap R_2$, the same applies to the projected execution α_{A_1}. However, since A_1 guarantees G_1 from S_1 given R_1, $S_1 \cap S_2 \subseteq S_1$, and $R_1 \cap R_2 \subseteq R_1$, it follows that all reachable states of PP in α_{A_1} are in G_1. Since α_{A_1} is the projection of α to the automaton $PP \times A_1$, it follows that all reachable states of PP in α are in G_1 also.

Now, let α_{A_2} be the projection of the execution α to the automaton $PP \times A_2$. Since the execution α starts in a state in $S_1 \cap S_2$ and is restricted to the states in $R_1 \cap R_2$, the same applies to the projected execution α_{A_2}. From above however, all reachable states in α are in G_1 and, therefore, it follows that the execution α_{A_2} is restricted to the states in $R_1 \cap R_2 \cap G_1$. However, since A_2 guarantees G_2 from S_2 given $R_2 \cap G_1$, $S_1 \cap S_2 \subseteq S_2$, and $R_1 \cap R_2 \cap G_1 \subseteq R_2 \cap G_1$, it follows that all reachable states of PP in α_{A_2} are in G_2. Finally, since α_{A_2} is the projection of α to the automaton $PP \times A_2$, it follows that all reachable states of PP in α are in G_2 also. ■

The fourth and fifth composition theorems require a preliminary lemma.

Lemma 3.1.6 *Suppose that A is a protector automaton for PP, with port set K. Suppose that A guarantees G from S given $R \cap G'$.*

Let α be any finite execution of $PP \times A$ starting in S and all of whose states are in R. Letting (i,t,s) be any state occurrence in α, if $s \notin G$ then $(i,t,s) \in past(\overline{G'}, \alpha)$.

Proof: Suppose for the sake of contradiction that $s \notin G$ and $(i,t,s) \notin past(\overline{G'}, \alpha)$. Let α_1 be the prefix of α ending with (i,t,s). Then, all states of α_1 are in G'. Since A guarantees G from S given $R \cap G'$, it follows that all states of α_1 are in G. But this contradicts the assumption that $s \notin G$. ■

Now we can prove the fourth composition theorem — the one involving a two-way protector dependency.

Theorem 3.1.7 *Suppose that A_1 and A_2 are protector automata for PP, with respective port sets K_1 and K_2, where $K_1 \cap K_2 = \emptyset$. Suppose that the protector A_1 guarantees G_1 from S_1 given $R_1 \cap G_2$ and the protector A_2 guarantees G_2 from S_2 given $R_2 \cap G_1$.*

Assume that α is any finite execution of the system $PP \times A_1 \times A_2$, starting from a state in $S_1 \cap S_2$ and all of whose states are in $R_1 \cap R_2$.

Then, one of the following holds:

1. *Every state in α is in $G_1 \cap G_2$.*
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* $G_1 \cap G_2$*,*

(b) *the last state occurrence in* α_1 *is in* G_1 *if and only if it is in* G_2*, and*

(c) *all state occurrences in* α_2 *except possibly the first are in* $past(\overline{G_1}, \alpha) \cap past(\overline{G_2}, \alpha)$*.*

Proof: Fix α as in the hypothesis. If every state in α is in $G_1 \cap G_2$ then we are done, so assume that some state in α is in $\overline{G_1} \cup \overline{G_2}$. Let B_1 and B_2 denote $\overline{G_1}$ and $\overline{G_2}$, respectively.

Let w_i be the first trajectory in α containing an occurrence of a state in $B_1 \cup B_2$, and suppose that w_i is a T_I-trajectory. Let T'_I be the subset of T_I consisting of all t such that $(i, t, w_i(t)) \in past(B_1 \cup B_2, \alpha)$. Then, T'_I is a non-empty subinterval of T_I that is "upward-closed", *i.e.*, if $t \in T'_I$, $t' \in T_I$, and $t < t'$ then $t' \in T'_I$. Since T'_I is an interval of reals, it has a left endpoint t, which might or might not itself be in T'_I. Let $s = w_i(t)$.

Then, we claim that splitting α exactly at (i, t, s) yields the needed decomposition into α_1 and α_2. There are three conditions to check:

1. All state occurrences in α_1 except possibly the last are in $G_1 \cap G_2$.

   This is true by the definitions of *past* and T'_I.

2. $s \in G_1$ if and only if $s \in G_2$.

   Suppose that $s \in B_1$. Then, Lemma 3.1.6 implies that $(i, t, s) \in past(B_2, \alpha)$. However, the definition of T'_I implies that no state occurrence preceding (i, t, s) is in B_2. Therefore, it follows that $s \in B_2$.

   Similarly, if $s \in B_2$ then $s \in B_1$.

3. All state occurrences in α_2 except possibly the first are in $past(B_1, \alpha) \cap past(B_2, \alpha)$.

   Consider any state occurrence (i', t', s') in α_2 other than the first. By definition of α_2 and *past*, it must be that $(i', t', s') \in past(B_1 \cup B_2, \alpha)$. Suppose, without loss of generality, that $(i', t', s') \in past(B_1, \alpha)$. This means that either $(i', t', s') \in B_1$, or there is a state occurrence (i'', t'', s'') preceding (i', t', s') in α such that $(i'', t'', s'') \in B_1$.

   In the former case, Lemma 3.1.6 implies that $(i', t', s') \in past(B_2, \alpha)$. In the latter case, Lemma 3.1.6 implies that $(i'', t'', s'') \in past(B_2, \alpha)$. This in turn implies that $(i', t', s') \in past(B_2, \alpha)$. This suffices.

■

In the following theorem, we extend the composition theorem of the two-way protector dependency case to the multiple-way protector dependency case; that is, the case in which the operation of each of the protectors within a prespecified set of protectors relies on the operation of all the other protectors in the set.

Theorem 3.1.8 *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for 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 $R_i \bigcap \left(\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'}\right)$.*

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 $\bigcap_{i \in \{1,\ldots,k\}} R_i$.

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 $\bigcap_{i \in \{1,\ldots,k\}} G_i$,*
   - *(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 $\bigcap_{i \in I} past(\overline{G_i}, \alpha)$, for some $I \subseteq \{1, \ldots, k\}$, where $|I| \geq 2$.*

Proof: Fix α as in the hypothesis. If every state in α is in $\bigcap_{i \in \{1,\ldots,k\}} G_i$ then we are done, so assume that some state in α is in $\bigcup_{i \in \{1,\ldots,k\}} \overline{G_i}$. For all $i \in \{1, \ldots, k\}$, let B_i denote $\overline{G_i}$.

Let w_j be the first trajectory in α containing an occurrence of a state in $\bigcup_{i \in \{1,\ldots,k\}} B_i$, and suppose that w_j is a T_I-trajectory. Let T'_I be the subset of T_I consisting of all t such that $(j, t, w_j(t)) \in past(\bigcup_{i \in \{1,\ldots,k\}} B_i, \alpha)$. Then, T'_I is a non-empty subinterval of T_I that is "upward-closed", *i.e.*, if $t \in T'_I$, $t' \in T_I$, and $t < t'$ then $t' \in T'_I$. Since T'_I is an interval of reals, it has a left endpoint t, which might or might not itself be in T'_I. Let $s = w_j(t)$.

Then, we claim that splitting α exactly at (j, t, s) yields the needed decomposition into α_1 and α_2. There are three conditions to check:

1. All state occurrences in α_1 except possibly the last are in $\bigcap_{i \in \{1,\ldots,k\}} G_i$.

   This is true by the definitions of *past* and T'_I.

2. 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 occurrence in α_1 is in $\overline{G_{i'}}$.

   Suppose that $s \in B_i$, for some $i \in \{1, \ldots, k\}$. Then, Lemma 3.1.6 implies that $(j, t, s) \in past(\overline{\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'}}, \alpha)$, *i.e.*, $(j, t, s) \in past(\bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}, \alpha)$. The definition of T'_I implies that no state occurrence preceding (j, t, s) is in the set $\bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}$. Therefore, it follows that $s \in \bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}$. This suffices.

3. All state occurrences in α_2 except possibly the first are in $\bigcap_{i \in I} past(\overline{G_i}, \alpha)$, for some $I \subseteq \{1, \ldots, k\}$, where $|I| \geq 2$.

   Consider any state occurrence (j', t', s') in α_2 other than the first. By definition of α_2 and *past*, it must be that $(j', t', s') \in past(\bigcup_{i \in \{1,\ldots,k\}} B_i, \alpha)$. Suppose, without loss of generality, that $(j', t', s') \in past(B_i, \alpha)$, for some $i \in \{1, \ldots, k\}$. This means that either $(j', t', s') \in B_i$, or there is a state occurrence (j'', t'', s'') preceding (j', t', s') in α such that $(j'', t'', s'') \in B_i$.

   In the former case, Lemma 3.1.6 implies that the state occurrence (j', t', s') satisfies the condition $(j', t', s') \in past(\overline{\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'}}, \alpha)$, which is equivalent to $(j', t', s') \in past(\bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}, \alpha)$. In the latter case, Lemma 3.1.6 implies that the state occurrence (j'', t'', s'') satisfies the condition $(j'', t'', s'') \in past(\overline{\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'}}, \alpha)$, which is equivalent to $(j'', t'', s'') \in past(\bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}, \alpha)$. This in turn implies that $(j', t', s') \in past(\bigcup_{i' \in \{1,\ldots,k\}, i' \neq i} B_{i'}, \alpha)$. This suffices.

■

3.2 An Abstract Protector

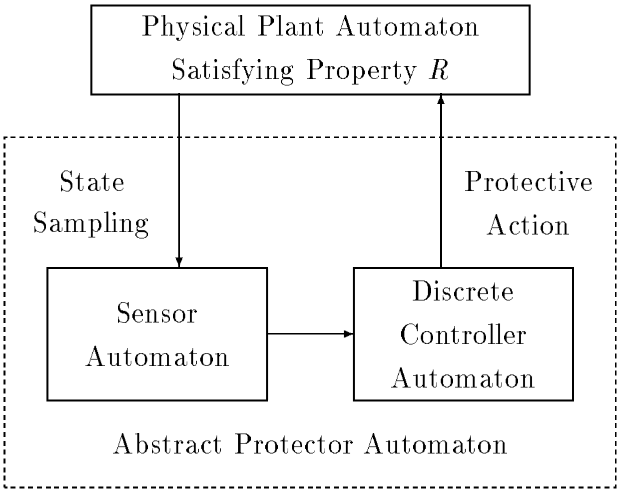
In this section, we define an abstract protector that is parameterized in terms of:

- *PP*, a particular physical plant automaton,
- R, G, and S, sets of states of *PP*,
- j, a particular port of *PP*, and
- d, a positive real-valued sampling period.

The *PP* automaton represents the physical plant being modeled. The set R 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 is the set of "good" states; that 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 protector communicates with the *PP* automaton through the port j and has a positive real-valued sampling period d.

The protector is composed of a *sensor automaton* and a *discrete controller automaton* as shown in Figure 3.1. Both the sensor and the discrete controller are described abstractly in terms of *PP*, *etc.* At intervals of d time units, the sensor automaton samples the output

Figure 3.1 Compositional structure of a physical plant and an abstract protector.



variables of the *PP* automaton. The discrete controller automaton is rather nondeterministic. Based on the output state information of the *PP* automaton sampled by the sensor, the discrete controller issues protective actions so as to guarantee that the *PP* automaton stays within the set G starting from S given R.

A particular instantiation of the abstract parameterized protector $Abs(PP, S, R, G, j, d)$ can be defined by simply specifying the parameters PP, S, R, G, j, and d. Often, after explicitly defining the parameters PP, S, R, G, j, and d, we refer to the particular abstract protector using only its port index, *i.e.*, Abs_j. The same applies for the parameterized sensor and discrete controller automata $Sensor(PP, S, R, G, j, d)$ and $DC(PP, S, R, G, j, d)$, respectively.

In several of the following chapters, we give explicit definitions of protectors for specific choices of *PP*, *etc.* The abstract protector of this section is used to aid in proving correctness of the later protectors.

3.2.1 Terminology and Assumptions

In this section, we define several functions and sets, which are useful in the definition and in the proof of correctness of the abstract protector, and present the assumptions made about the physical plant and the abstract protector automata. It is important to note that the assumptions presented in this section must be satisfied by any physical plant and abstract protector automata defined and analyzed using the framework developed in this thesis. Throughout this section, we also state several lemmas which are used in subsequent sections and chapters.

We begin by stating two simple assumptions about the physical plant automaton. First, we assume that the *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. Second, 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. These two assumptions are formally stated by the following two axioms.

Axiom 3.2.1 *The PP automaton has no input variables on any of its ports, i.e., $U_{PP_j} = \emptyset$, for all $j \in J$.*

Axiom 3.2.2 *The PP automaton has no output actions on any of its ports, i.e., $\Sigma^{out}_{PP_j} = \emptyset$, for all $j \in J$.*

Next, 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.

$future_{PP,R,j} : \mathcal{P}(R) \times \mathcal{P}(\mathbb{R}^{\geq 0}) \to \mathcal{P}(R)$, 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 $\alpha.ltime \in T$.

When either argument of the function $future_{PP,R,j}$ is a singleton set, we omit the set brackets, *e.g.*, for any $p \in R$ and $t \in \mathbb{R}^{\geq 0}$, we write $future_{PP,R,j}(p,t)$ as shorthand for $future_{PP,R,j}(\{p\},\{t\})$. Moreover, it is important to note that the function $future_{PP,R,j}$ depends on the automaton *PP*, the set R, and the port j. Henceforth however, when the automaton *PP*, the set R, and the port j are clear from context, they are omitted; that is, we use the notation *future* instead of $future_{PP,R,j}$.

Lemma 3.2.1 *For all $P, P' \subseteq R$, $T, T' \subseteq \mathbb{R}^{\geq 0}$, and $t, t' \in \mathbb{R}^{\geq 0}$, the following are true:*

1. *If $P \subseteq P'$ and $T \subseteq T'$ then $future_{PP,R,j}(P,T) \subseteq future_{PP,R,j}(P',T')$.*
2. *$future_{PP,R,j}(P, t+t') = future_{PP,R,j}(future_{PP,R,j}(P,t),t')$.*
3. *$P \subseteq future_{PP,R,j}(P,0)$.*
4. *$future_{PP,R,j}(future_{PP,R,j}(P,T),T') = future_{PP,R,j}(P,T'')$, where $T'' = \{\tau + \tau' \mid \tau \in T \text{ and } \tau' \in T'\}$.*

Proof: Follow directly from the definition of the function *future*. ∎

Lemma 3.2.2 *Suppose that π is any discrete action of PP other than an input action on port j and that $p, p' \in R$ such that $p \xrightarrow{\pi}_{PP} p'$. Then, for any $T \subseteq \mathbb{R}^{\geq 0}$, $future_{PP,R,j}(p', T) \subseteq future_{PP,R,j}(p, T)$.*

Proof: Lemma 3.2.1, part 1, and the fact that $p' \in \mathit{future}(p, 0)$ imply that $\mathit{future}(p', T) \subseteq \mathit{future}(\mathit{future}(p, 0), T)$. Moreover, Lemma 3.2.1, part 4, implies that $\mathit{future}(\mathit{future}(p, 0), T) = \mathit{future}(p, T)$. Therefore, it follows that $\mathit{future}(p', T) \subseteq \mathit{future}(p, T)$, as needed. ■

We define a function, $\textit{no-op}_{PP,R,j}$, 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.

$\textit{no-op}_{PP,R,j} : R \to \mathcal{P}(\Sigma^{in}_{PP_j})$, defined by:
$\pi \in \textit{no-op}_{PP,R,j}(p)$ if and only if π is an input action on port j of PP such that for all $p', p'' \in R$ satisfying $p' \in \mathit{future}_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' = p'$.

Henceforth, for any state p in R, the input actions in the set $\textit{no-op}_{PP,R,j}(p)$ are referred to as no-op input actions on port j of PP for the state p.

It is important to note that the above definition of the function $\textit{no-op}_{PP,R,j}$ conforms to Axiom **D3** of the HIOA model of Section 2.2 since, by Axiom 3.2.1, the PP automaton has no input variables on any of its ports. Moreover, the function $\textit{no-op}_{PP,R,j}$ depends on the automaton PP, the set R, and the port j. Henceforth however, when the automaton PP, the set R, and the port j are clear from context, they are omitted; that is, we use the notation *no-op* instead of $\textit{no-op}_{PP,R,j}$.

We proceed by stating another assumption about the physical plant automaton PP. We assume that there exist no-op input actions on port j for every state of the PP automaton in the set R. This assumption is formally stated by the following axiom.

Axiom 3.2.3 *For every $p \in R$, it is the case that $\textit{no-op}_{PP,R,j}(p) \neq \emptyset$.*

Axiom 3.2.3 states that no-op input actions on port j *exist* for every state p of PP in R. It is important to realize, however, that Axiom 3.2.3 does not claim that for $p \in R$ it is possible to determine from the valuation $y = p \lceil Y_{PP}$ of the output variables of the PP automaton which input actions are no-op input actions on port j for the state p. In fact, it is plausible that the information provided by the output variables Y_{PP} of the PP automaton is not sufficient to determine which of the input actions $\Sigma^{in}_{PP_j}$ are no-op input actions on port j for each state p of the PP automaton in the set R.

Since the PP automaton is assumed to have no input actions on any of its ports (Axiom 3.2.1), input actions of the physical plant are often "idempotent", in the sense that in any execution of the PP automaton if any particular input action π on port j is performed consecutively multiple times with no other intervening input actions on port j, then all such input actions π except the first, do not change the state of the PP automaton. For any physical plant automaton PP in which all input actions are idempotent and any state p of the PP automaton in the set R, the most recently performed input action on port j is a no-op input action on port j for the state p.

We define a set, $\mathit{very\text{-}safe}_{PP,R,G,j}$, 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 $\mathit{very\text{-}safe}_{PP,R,G,j}$ 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.

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

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

It is important to note that the set $\mathit{very\text{-}safe}_{PP,R,G,j}$ depends on the automaton PP, the sets R and G, and the port j. Henceforth however, when the automaton PP, the sets R and G, and the port j are clear from context, they are omitted; that is, we use the notation *very-safe* instead of $\mathit{very\text{-}safe}_{PP,R,G,j}$.

Lemma 3.2.3

1. *$\mathit{very\text{-}safe}_{PP,R,G,j} \subseteq G$.*
2. *If $p \in \mathit{very\text{-}safe}_{PP,R,G,j}$ then $\mathit{future}_{PP,R,j}(p, \mathbb{R}^{\geq 0}) \subseteq \mathit{very\text{-}safe}_{PP,R,G,j}$.*

Proof: Follow directly from the definition of *very-safe*. ■

We define a set, $\mathit{safe}_{PP,R,G,j}$, which is comprised of the states of PP that satisfy R and from which the protector on port j has a "winning protective strategy". Namely, there exists an input action on port j of the PP automaton whose immediate execution — 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 particular input action of PP on port j is in the set $\mathit{very\text{-}safe}_{PP,R,G,j}$.

$\mathit{safe}_{PP,R,G,j} \subseteq R$, defined by:

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

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

It is important to note that the set $safe_{PP,R,G,j}$ depends on the automaton PP, the sets R and G, and the port j. Henceforth however, when the automaton PP, the sets R and G, and the port j are clear from context, they are omitted; that is, we use the notation *safe* instead of $safe_{PP,R,G,j}$.

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 $very\text{-}safe_{PP,R,G,j}$.

$safe_{PP,R,G,j} : \Sigma^{in}_{PP_j} \to \mathcal{P}(R)$, 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 $p'' \in very\text{-}safe_{PP,R,G,j}$.

It is important to note that the function $safe_{PP,R,G,j}$ depends on the automaton PP, the sets R and G, and the port j. Henceforth however, when the automaton PP, the sets R and G, and the port j are clear from context, they are omitted; that is, we use the notation $safe(\pi)$ instead of $safe_{PP,R,G,j}(\pi)$, for any input action π of PP on port j.

Lemma 3.2.4

1. *$safe_{PP,R,G,j} \subseteq G$.*
2. *For any $p \in R$, $p \in safe_{PP,R,G,j}$ if and only if $future_{PP,R,j}(p, 0) \subseteq safe_{PP,R,G,j}$.*
3. *$very\text{-}safe_{PP,R,G,j} \subseteq safe_{PP,R,G,j}$.*

Proof:

1. Let p be any state in *safe*. From the definition of *safe* it follows that $future(p, 0) \subseteq G$. Therefore, Lemma 3.2.1, part 3, implies that $p \in G$. It follows that $safe \subseteq G$.
2. In the forward direction, let $p \in safe$ and $p' \in future(p, 0)$. We must show that $p' \in safe$; that is, we must show that (i) $future(p', 0) \subseteq G$, and (ii) there exists an input action π on port j such that for all $p'', p''' \in R$ satisfying $p'' \in future(p', 0)$ and $p'' \xrightarrow{\pi}_{PP} p'''$, it is the case that $p''' \in very\text{-}safe$. Lemma 3.2.2 implies that $future(p', 0) \subseteq future(p, 0)$ and, therefore, the conditions to be shown follow from the fact that $p \in safe$.

For the converse, let $p \in R$ and $\mathit{future}(p,0) \subseteq \mathit{safe}$. We must show that $p \in \mathit{safe}$. From Lemma 3.2.1, part 3, it is the case that $p \in \mathit{future}(p,0)$. Therefore, it follows that $p \in \mathit{safe}$.

3. Letting $p \in \mathit{very\text{-}safe}$, we must show that $p \in \mathit{safe}$; that is, we must show that (i) $\mathit{future}(p,0) \subseteq G$, and (ii) there exists an input action π on port j such that for all $p', p'' \in R$ satisfying $p' \in \mathit{future}(p,0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in \mathit{very\text{-}safe}$.

   For the first condition, Lemma 3.2.1, part 1, implies that $\mathit{future}(p,0) \subseteq \mathit{future}(p,\mathbb{R}^{\geq 0})$. However, since $p \in \mathit{very\text{-}safe}$ it is the case that $\mathit{future}(p,\mathbb{R}^{\geq 0}) \subseteq G$. Therefore, it follows that $\mathit{future}(p,0) \subseteq G$, as needed.

   For the second condition, since $\mathit{no\text{-}op}(p) \neq \emptyset$ by Axiom 3.2.3, let $\pi \in \mathit{no\text{-}op}(p)$. Moreover, let $p', p'' \in R$ such that $p' \in \mathit{future}(p,0)$ and $p' \xrightarrow{\pi}_{PP} p''$. Since $p \in \mathit{very\text{-}safe}$, Lemma 3.2.3, part 2, implies that $p' \in \mathit{very\text{-}safe}$. Moreover, since π is defined to be a no-op input action on port j for the state p, it follows that $p'' = p'$. Therefore, it is the case that $p'' \in \mathit{very\text{-}safe}$, as needed.

■

We proceed by stating two more assumptions about the *PP* automaton. We assume that membership of a state of the *PP* automaton in the set *safe* is determinable from the output variables of the *PP* automaton, *i.e.*, the set *safe* is Y_{PP}-determinable (as defined in Section 2.1). Moreover, we assume that for any state in the set *safe*, an appropriate action to guarantee safety can be determined from the output variables of the *PP* automaton, *i.e.*, the variables in Y_{PP}. These two assumptions are formally stated by the following two axioms.

Axiom 3.2.4 $\mathit{safe}_{PP,R,G,j}$ *is* Y_{PP}*-determinable.*

For any valuation y of the output variables Y_{PP} of the *PP* automaton, we use the notation $y \in \mathit{safe}$ to denote the existence of a state $p \in \mathit{safe}$ such that $p \lceil Y_{PP} = y$. In fact, by Axiom 3.2.4, for any valuation y of the output variables Y_{PP} of the *PP* automaton, the existence of a state $p \in \mathit{safe}$ such that $p \lceil Y_{PP} = y$ implies that all states $p' \in R$ such that $p' \lceil Y_{PP} = y$ are in the set *safe*.

Axiom 3.2.5 *There exists a function,* `decision`*, from valuations of* Y_{PP} *to* $\Sigma^{in}_{PP_j}$ *such that for any* $y \in \mathbf{Y_{PP}}$ *and* $p \in R$ *satisfying* $p \lceil Y_{PP} = y$*, it is the case that if* $y \in \mathit{safe}_{PP,R,G,j}$ *then* $p \in \mathit{safe}_{PP,R,G,j}(\texttt{decision}(y))$.

We define a function, $\mathit{delay\text{-}safe}_{PP,R,G,j}$, 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 $\mathit{safe}_{PP,R,G,j}$.

$delay\text{-}safe_{PP,R,G,j} : \mathbb{R}^{\geq 0} \rightarrow \mathcal{P}(R)$, 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. $future_{PP,R,j}(p,t) \subseteq safe_{PP,R,G,j}$.

It is important to note that the function $delay\text{-}safe_{PP,R,G,j}$ depends on the automaton *PP*, the sets R and G, and the port j. Henceforth however, when the automaton *PP*, the sets R and G, and the port j are clear from context, they are omitted; that is, we use the notation $delay\text{-}safe(t)$ instead of $delay\text{-}safe_{PP,R,G,j}(t)$, for any $t \in \mathbb{R}^{\geq 0}$.

Lemma 3.2.5 *For any $t, t' \in \mathbb{R}^{\geq 0}$, such that $t \leq t'$, the following hold:*

1. $very\text{-}safe_{PP,R,G,j} \subseteq delay\text{-}safe_{PP,R,G,j}(t)$.
2. $safe_{PP,R,G,j} = delay\text{-}safe_{PP,R,G,j}(0)$.
3. $delay\text{-}safe_{PP,R,G,j}(t') \subseteq delay\text{-}safe_{PP,R,G,j}(t)$.

Proof: Follow directly from the definitions of *very-safe*, *safe*, and $delay\text{-}safe(t)$, for any $t \in \mathbb{R}^{\geq 0}$, and the Lemmas 3.2.3 and 3.2.4. ■

We conclude by stating three assumptions made about the abstract protector automaton. In particular, 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 (as defined in Section 2.1). Moreover, we assume that the set of start states S is a subset of the set *safe*. These assumptions are formally stated by the following three axioms.

Axiom 3.2.6 *R is Y_{PP}-determinable.*

Axiom 3.2.7 *G is Y_{PP}-determinable.*

Axiom 3.2.8 $S \subseteq safe_{PP,R,G,j}$.

As noted above, all assumptions described by Axioms 3.2.1–3.2.8 must be satisfied by the physical plant and abstract protector automata defined and analyzed using the framework developed in this thesis.

3.2.2 Sensor Automata

The sensor automaton $Sensor_j$, defined in Figure 3.2, behaves as follows: at time 0 and every d time units thereafter, it outputs the valuation y of the output variables Y_{PP} of the *PP*

Figure 3.2 *$Sensor_j$* automaton definition.

Actions: Input: e, the environment action
Output: $\mathtt{snapshot}(y)_j$, for each valuation y of Y_{PP}, *i.e.*, for all $y \in \boldsymbol{Y_{PP}}$

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: $Y_{PP} :\in \boldsymbol{Y_{PP}}$

$\mathtt{snapshot}(y)_j$
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 $t \in T_I$
$w(t).now_j = w(0).now_j + t$
$w(t).now_j \leq w(t).next\text{-}snap_j$

automaton using a $\mathtt{snapshot}(y)_j$ output action. The *$Sensor_j$* automaton keeps track of the appropriate times for scheduling each $\mathtt{snapshot}(y)_j$ action, for $y \in \boldsymbol{Y_{PP}}$, using the internal variables now_j and $next\text{-}snap_j$. The variable now_j stores the time that has elapsed from the beginning of the particular execution of the *$Sensor_j$* automaton. The variable $next\text{-}snap_j$ stores the next point in time in which the output variables Y_{PP} of the *PP* automaton must be sampled.

The discrete actions of the *$Sensor_j$* automaton are the input action e and the output actions $\mathtt{snapshot}(y)_j$, for all $y \in \boldsymbol{Y_{PP}}$. The environment action e allows for arbitrary changes to the input variables Y_{PP} as a consequence of discrete transitions outside the *$Sensor_j$* automaton but does not affect the local variables of the *$Sensor_j$* automaton. Each $\mathtt{snapshot}(y)_j$ action, for $y \in \boldsymbol{Y_{PP}}$, outputs the valuation y of the output variables Y_{PP} of the *PP* automaton. In order to conform to Axiom **D3** of the HIOA model of Section 2.2, each input variable u of the *$Sensor_j$* automaton, for $u \in Y_{PP}$, is assigned an arbitrary value in the set $type(u)$. It can easily be seen that the *$Sensor_j$* automaton satisfies the Axioms **D1–D3** of the HIOA model of Section 2.2.

The trajectory specification for the $Sensor_j$ automaton gives restrictions on a trajectory w with domain T_I. Since the $Sensor_j$ automaton has no control over its input variables, the input variables of the $Sensor_j$ automaton are allowed to change arbitrarily throughout a trajectory w. It is important to note that the $Sensor_j$ automaton does not allow time-passage unless the condition $now_j \leq next\text{-}snap_j$ is satisfied. As a result, in order for time to proceed when $now_j = next\text{-}snap_j$, a $\texttt{snapshot}(y)_j$ output action, for some $y \in \boldsymbol{Y_{PP}}$, is eventually scheduled. It can easily be seen that the $Sensor_j$ automaton satisfies the Axioms **T1–T3** of the HIOA model of Section 2.2.

Finally, since each input variable u of the $Sensor_j$ automaton, for $u \in Y_{PP}$, can initially assume an arbitrary value in the set $type(u)$, the $Sensor_j$ automaton satisfies Axiom **Init** of the HIOA model of Section 2.2. Since the $Sensor_j$ automaton satisfies the Axioms **Init**, **D1–D3**, and **T1–T3** of the HIOA model of Section 2.2, it follows that it is a HIOA.

3.2.3 Discrete Controller Automata

The discrete controller automaton DC_j, defined in Figure 3.3, uses the valuation of the output variables of the PP automaton, which is sampled by the $Sensor_j$ automaton, to determine which protective action must be scheduled 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 $safe$.

The discrete actions of the DC_j automaton are the input action e, the input actions $\texttt{snapshot}(y)_j$, for all $y \in \boldsymbol{Y_{PP}}$, and the output actions π, for all $\pi \in \Sigma^{in}_{PP_j}$. The environment action e allows the scheduling of discrete transitions outside the DC_j automaton. Since the DC_j automaton has no input variables, the environment action e is stuttering; that is, the execution of the environment action e does not affect the state of the DC_j automaton. Each $\texttt{snapshot}(y)_j$ action, for $y \in \boldsymbol{Y_{PP}}$, determines which output action π in the set $\Sigma^{in}_{PP_j}$ should be scheduled and stores it in the internal variable $send_j$. In a subsequent step, prior to any time-passage but with the possibility of intervening discrete actions, the DC_j automaton schedules the output action π that is stored in the internal variable $send_j$. It is important to note that time-passage is not enabled while any of the actions π in $\Sigma^{in}_{PP_j}$ is enabled. As a result, in order for time to proceed, the action π that is stored in the internal variable $send_j$ is eventually scheduled. It can easily be seen that the $Sensor_j$ automaton satisfies the Axioms **D1–D3** of the HIOA model of Section 2.2.

The trajectory specification of the DC_j automaton is trivial. It simply states that the internal variable $send_j$, which comprises the state of the DC_j automaton, remains unchanged and equal to $null$ throughout any trajectory of the DC_j automaton. It can easily be seen that the DC_j automaton satisfies the Axioms **T1–T3** of the HIOA model of Section 2.2.

Finally, since the DC_j automaton has no input variables, Axiom **Init** of the HIOA model

Figure 3.3 DC_j automaton definition.

Actions: Input: e, the environment action (stuttering)
`snapshot`$(y)_j$, for each valuation y of Y_{PP}, *i.e.*, for all $y \in \boldsymbol{Y_{PP}}$
Output: π, for all $\pi \in \Sigma^{in}_{PP_j}$, *i.e.*, all the input actions on port j of PP

Variables: Internal: $send_j \in \Sigma^{in}_{PP_j} \cup \{null\}$, initially $null$

Discrete Transitions:

e

Eff: None

`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

$send_j :\in \Sigma^{in}_{PP_j}$

π

Pre: $send_j = \pi$

Eff: $send_j := null$

Trajectories:

$w.send_j \equiv null$

---

of Section 2.2 it trivially satisfied. Since the DC_j automaton satisfies the Axioms **Init**, **D1–D3**, and **T1–T3** of the HIOA model of Section 2.2, it follows that it is a HIOA.

The DC_j automaton's decision as to which output action to enable and subsequently schedule is made nondeterministically. Let y be any valuation of the output variables Y_{PP} of the PP automaton, *i.e.*, $y \in \boldsymbol{Y_{PP}}$.

On one hand, if $y \in safe$, then an output action ϕ in $\Sigma^{in}_{PP_j}$ is allowed only if for all $p, p', p'' \in R$ such that $p\lceil Y_{PP} = y$, $p' \in future(p, 0)$, and $p' \xrightarrow{\phi}_{PP} p''$, it is the case that $p'' \in delay\text{-}safe(d)$. Let Φ be the set of all output actions ϕ in $\Sigma^{in}_{PP_j}$ allowed by the DC_j automaton in this case. In order for an implementation of a particular instantiation of the DC_j automaton to exist, it is imperative that the set of output actions Φ be non-empty and that at least one of the actions in Φ can be determined from the valuation y of Y_{PP}. In fact, since $y \in safe$, an output action π in $\Sigma^{in}_{PP_j}$ that is allowed by the DC_j automaton is guaranteed to exist, *i.e.*, $\Phi \neq \emptyset$. Axiom 3.2.4 implies that for all $p \in R$ such that $p \lceil Y_{PP} = y$ it is the case that $p \in safe$, *i.e.*, for all $p \in R$ such that $p \lceil Y_{PP} = y$, there exists an action π in $\Sigma^{in}_{PP_j}$ such that for all $p', p'' \in R$ satisfying $p' \in future(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in very\text{-}safe$. Therefore, from Lemma 3.2.5, part 1, it follows that $p'' \in delay\text{-}safe(d)$, as needed. Moreover, by Axiom 3.2.5, an output action π in $\Sigma^{in}_{PP_j}$ that is allowed by the DC_j automaton can be determined from the valuation y of Y_{PP}; that is, there exists a function, `decision`, from valuations of Y_{PP} to $\Sigma^{in}_{PP_j}$, such that for any $y \in \boldsymbol{Y_{PP}}$ and $p \in R$ satisfying $p \lceil Y_{PP} = y$, it is the case that $p \in safe(\mathtt{decision}(y))$.

On the other hand, if $y \notin safe$, then any output action π of the DC_j automaton is allowed by default. However, as shown in the following section, this default case never occurs in states that are R-reachable by a finite execution of the composed system $PP \times Sensor_j \times DC_j$ starting in an initial state in the set S.

The nondeterminism in the description of the DC_j automaton allows the freedom to choose any response that satisfies the given conditions — however, in any 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.

Henceforth, let the "abstract protector" automaton Abs_j be the composition of the $Sensor_j$ and DC_j automata, *i.e.*, $Abs_j = Sensor_j \times DC_j$. Proposition 2.7.1, implies that the automaton Abs_j is a HIOA.

3.2.4 Correctness of the Abstract Protector

In this section, we prove that the abstract protector Abs_j guarantees G in the physical plant PP from S given R.

Lemma 3.2.6 *For any reachable state s of $Abs(PP, S, R, G, j, d)$, if $s.next\text{-}snap_j = s.now_j$, then $s.send_j = null$.*

Proof: Follows directly from the definition of the $Sensor_j$ and the DC_j automata. ■

The following lemma considers the composition $PP \times Abs_j$ of the physical plant automaton PP and the abstract protector automaton Abs_j. Let s be any state of the composed system and let $s.ppstate$ be the restriction of s onto the state space of the PP automaton, *i.e.*, $s.ppstate = s \lceil V_{PP}$.

Lemma 3.2.7 *The following are true in any state s of $PP \times Abs(PP, S, R, G, j, d)$, that is reachable from an initial state in $safe_{PP,R,G,j}$, via an execution that only involves states in R.*

1. *If* $s.send_j = null$*, then* $s.ppstate \in$ *delay-safe*$_{PP,R,G,j}(s.next\text{-}snap_j - s.now_j)$.
2. *If* $s.send_j = \phi$*, for some* $\phi \in \Sigma^{in}_{PP_j}$*, then*

   (a) *future*$_{PP,R,j}(s.ppstate, 0) \subseteq G$*, and*

   (b) *For every* $p', p'' \in R$ *such that* $p' \in$ *future*$_{PP,R,j}(s.ppstate, 0)$ *and* $p' \xrightarrow{\phi}_{PP} p''$*, it is the case that* $p'' \in$ *delay-safe*$_{PP,R,G,j}(d)$.

Proof: In an initial state of $PP \times Abs_j$ it is the case that $s.send_j = null$. Therefore, since the first clause of the invariant applies, we must show that $s.ppstate \in$ *delay-safe*$(s.next\text{-}snap_j - s.now_j)$. However, in an initial state $PP \times Abs_j$ it is the case that $s.next\text{-}snap_j = s.now_j = 0$. Therefore, we must show that $s.ppstate \in$ *delay-safe*(0), which by Lemma 3.2.5, part 2, is equivalent to $s.ppstate \in safe$. But this is true by our assumption about the start states of the executions considered in this lemma.

We now show that the invariant is preserved by every discrete transition $s \xrightarrow{\pi} s'$ of $PP \times Abs_j$, for $s, s' \in states(PP \times Abs_j)$ such that $s.ppstate, s'.ppstate \in R$ and $\pi \in \Sigma_{PP \times Abs_j}$. We consider cases:

1. $\pi = \texttt{snapshot}(y)_j$.

   From the effects of the $\texttt{snapshot}(y)_j$ action, it follows that $s'.send_j \in \Sigma^{in}_{PP_j}$. Therefore, we must show the second clause of the invariant for the state s'; that is, we must show that (a) *future*$(s'.ppstate, 0) \subseteq G$, and (b) for every $p', p'' \in R$ such that $p' \in$ *future*$(s'.ppstate, 0)$ and $p' \xrightarrow{s'.send_j}_{PP} p''$, it is the case that $p'' \in$ *delay-safe*(d).

   Lemma 3.2.6 and the precondition of the $\texttt{snapshot}(y)_j$ action imply that $s.send_j = null$. Therefore, the invariant for s implies that $s.ppstate \in$ *delay-safe*$(s.next\text{-}snap_j - s.now_j)$. Since the precondition of the $\texttt{snapshot}(y)_j$ action implies that $s.next\text{-}snap_j = s.now_j$, it follows that $s.ppstate \in$ *delay-safe*(0). Therefore, Lemma 3.2.5, part 2, implies that $s.ppstate \in safe$.

   For condition (a), since $s.ppstate \in safe$, it is the case that *future*$(s.ppstate, 0) \subseteq G$. Since the $\texttt{snapshot}(y)_j$ action affects only the $send_j$ of the DC_j automaton and the PP automaton has no input variables on any of its ports, it is the case that $s'.ppstate = s.ppstate$. Therefore, it follows that *future*$(s'.ppstate, 0) \subseteq G$, as needed.

   For condition (b), since $s.ppstate \in safe$, the "then clause" of the determination of $s'.send_j$ is used. Therefore, the discrete step $s \xrightarrow{\pi} s'$ sets the variable $s'.send_j$ to some ϕ in $\Sigma^{in}_{PP_j}$ with the property that for every $p', p'' \in R$ such that $p' \in$ *future*$(s'.ppstate, 0)$ and $p' \xrightarrow{\phi}_{PP} p''$, it is the case that $p'' \in$ *delay-safe*(d), as needed.
2. $\pi \in \Sigma^{in}_{PP_j}$.

   The precondition implies that $s.send_j = \pi \neq null$. Therefore, the invariant for the state s implies that *future*$(s.ppstate, 0) \subseteq G$ and that for every $p', p'' \in R$ such that

$p' \in \mathit{future}(s.ppstate, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in \mathit{delay\text{-}safe}(d)$. As a result of the step, it is the case that $s'.send_j = \mathit{null}$ and $s'.next\text{-}snap_j - s'.now_j = d$. Moreover, the invariant for the state s implies that $s'.ppstate \in \mathit{delay\text{-}safe}(d)$. Since $s'.next\text{-}snap_j - s'.now_j = d$ and $s'.ppstate \in \mathit{delay\text{-}safe}(d)$, it follows that $s'.ppstate \in \mathit{delay\text{-}safe}(s'.next\text{-}snap_j - s'.now_j)$, as needed.

3. $\pi \in \Sigma_{PP} - \Sigma^{in}_{PP_j}$ (π is a discrete action of *PP* other than an input action on port j).

   For any discrete action π of the *PP* automaton other than an input action on port j, it is the case that $s.send_j = s'.send_j$, $s.now_j = s'.now_j$, and $s.next\text{-}snap_j = s'.next\text{-}snap_j$.

   If $s.send_j = \mathit{null}$, then the invariant for s implies that $s.ppstate \in \mathit{delay\text{-}safe}(t)$, where $t = s.next\text{-}snap_j - s.now_j$; that is, $\mathit{future}(s.ppstate, [0, t]) \subseteq G$ and $\mathit{future}(s.ppstate, t) \subseteq \mathit{safe}$. However, Lemma 3.2.2 implies that $\mathit{future}(s'.ppstate, t) \subseteq \mathit{future}(s.ppstate, t)$, for all $t \in \mathbb{R}^{\geq 0}$. Since $s.next\text{-}snap_j - s.now_j = s'.next\text{-}snap_j - s'.now_j$, it follows that $\mathit{future}(s'.ppstate, [0, t]) \subseteq G$ and $\mathit{future}(s'.ppstate, t) \subseteq \mathit{safe}$, where $t = s'.next\text{-}snap_j - s'.now_j$. These two conditions imply that $s'.ppstate \in \mathit{delay\text{-}safe}(s'.next\text{-}snap_j - s'.now_j)$. This yields the invariant.

   A similar argument holds if $s.send_j = \phi$, for some $\phi \in \Sigma^{in}_{PP_j}$. In this case, the invariant for s implies that $\mathit{future}(s.ppstate, 0) \subseteq G$ and that for every $p', p'' \in R$ such that $p' \in \mathit{future}(s.ppstate, 0)$ and $p' \xrightarrow{\phi}_{PP} p''$, it is the case that $p'' \in \mathit{delay\text{-}safe}(d)$. However, Lemma 3.2.2 implies that $\mathit{future}(s'.ppstate, 0) \subseteq \mathit{future}(s.ppstate, 0)$. Therefore, it follows that $\mathit{future}(s'.ppstate, 0) \subseteq G$ and that for every $p', p'' \in R$ such that $p' \in \mathit{future}(s'.ppstate, 0)$ and $p' \xrightarrow{\phi}_{PP} p''$, it is the case that $p'' \in \mathit{delay\text{-}safe}(d)$. This yields the invariant.

4. $\pi = e$ (π is the environment action).

   Since the input variables of the $Sensor_j$ automaton are the output variables of the *PP* automaton, the DC_j automaton has no input variables, and the *PP* automaton has no input variables on any of its ports, it follows that the composition $PP \times Abs_j$ has no input variables. Therefore, the action π is the stuttering environment action, *i.e.*, $s' = s$, and the invariant for the state s implies the invariant for the state s'.

Finally, we show that the invariant is preserved by any non-trivial closed trajectory w in $\mathcal{W}_{PP \times Abs_j}$. Suppose that the states s and s', for some $s, s' \in \mathit{states}(PP \times Abs_j)$ such that $s.ppstate, s'.ppstate \in R$, are the first and last states of the trajectory w, respectively. Since time-passage is enabled, it is the case that $send_j = \mathit{null}$ throughout the trajectory w. Therefore, the invariant for the state s implies that $s.ppstate \in \mathit{delay\text{-}safe}(s.next\text{-}snap_j - s.now_j)$; that is, $\mathit{future}(s.ppstate, [0, s.next\text{-}snap_j - s.now_j]) \subseteq G$ and $\mathit{future}(s.ppstate, s.next\text{-}snap_j - s.now_j) \subseteq \mathit{safe}$. We must show that $s'.ppstate \in \mathit{delay\text{-}safe}(s'.next\text{-}snap_j - s'.now_j)$; that is, $\mathit{future}(s'.ppstate, [0, s'.next\text{-}snap_j - s'.now_j]) \subseteq G$ and $\mathit{future}(s'.ppstate, s'.next\text{-}snap_j - s'.now_j) \subseteq \mathit{safe}$. It suffices to show that $\mathit{future}(s'.ppstate, [0, s'.next\text{-}snap_j - s'.now_j]) \subseteq$

$\mathit{future}(s.\mathit{ppstate}, [0, s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j])$ and $\mathit{future}(s'.\mathit{ppstate}, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j) \subseteq \mathit{future}(s.\mathit{ppstate}, s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j)$.

From the fact that $s'.\mathit{ppstate} \in \mathit{future}(s.\mathit{ppstate}, w.\mathit{ltime})$ and Lemma 3.2.1, part 1, it follows that $\mathit{future}(s'.\mathit{ppstate}, [0, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j]) \subseteq \mathit{future}(\mathit{future}(s.\mathit{ppstate}, w.\mathit{ltime}), [0, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j])$. But Lemma 3.2.1, part 4, implies that $\mathit{future}(\mathit{future}(s.\mathit{ppstate}, w.\mathit{ltime}), [0, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j]) = \mathit{future}(s.\mathit{ppstate}, [w.\mathit{ltime}, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j + w.\mathit{ltime}])$. Moreover, from Lemma 3.2.1, part 1, it follows that $\mathit{future}(s.\mathit{ppstate}, [w.\mathit{ltime}, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j + w.\mathit{ltime}]) \subseteq \mathit{future}(s.\mathit{ppstate}, [0, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j + w.\mathit{ltime}])$. Finally, since $s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j + w.\mathit{ltime} = s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j$ it follows that $\mathit{future}(s'.\mathit{ppstate}, [0, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j]) \subseteq \mathit{future}(s.\mathit{ppstate}, [0, s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j])$, as needed.

Using similar arguments, it can be shown that $\mathit{future}(s'.\mathit{ppstate}, s'.\mathit{next\text{-}snap}_j - s'.\mathit{now}_j) \subseteq \mathit{future}(s.\mathit{ppstate}, s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j)$. ■

Lemma 3.2.8 *For any state s of $PP \times Abs(PP, S, R, G, j, d)$ that is reachable from an initial state in $\mathit{safe}_{PP,R,G,j}$ via an execution that only involves states in R, it is the case that $s.\mathit{ppstate} \in G$.*

Proof: If $s.\mathit{send}_j = \mathit{null}$ then Lemma 3.2.7 implies that the state $s.\mathit{ppstate}$ is in the set $\mathit{delay\text{-}safe}(s.\mathit{next\text{-}snap}_j - s.\mathit{now}_j)$, which implies that $\mathit{future}(s.\mathit{ppstate}, 0) \subseteq G$. On the other hand, if $s.\mathit{send}_j \neq \mathit{null}$, then Lemma 3.2.7 implies that $\mathit{future}(s.\mathit{ppstate}, 0) \subseteq G$. Thus, in either case it is the case that $\mathit{future}(s.\mathit{ppstate}, 0) \subseteq G$. Finally, Lemma 3.2.1, part 3, implies that $s.\mathit{ppstate} \in G$. ■

Theorem 3.2.9 *$Abs(PP, S, R, G, j, d)$ guarantees G in PP from $\mathit{safe}_{PP,R,G,j}$ given R.*

Proof: Let s be any state of the composed system $PP \times Abs_j$ that is reachable from an initial state in *safe* via an execution that only involves states in R. Then, Lemma 3.2.8 implies that $s.\mathit{ppstate} \in G$, as needed. ■

Corollary 3.2.10 *$Abs(PP, S, R, G, j, d)$ guarantees G in PP from S given R.*

Proof: Follows directly from Theorem 3.2.9 and Axiom 3.2.8. ■

Chapter 4

Modeling a System of n Vehicles

In this chapter, we present a model for a simplified version of the PRT 2000™ system under development at Raytheon Corporation. The physical plant model involves n vehicles traveling on a single track. Since this thesis is only concerned with safety, the details of the operation of the physical plant and the aspects of the system geared towards performance are omitted.

The model, called VEHICLES, is a HIOA and conforms to the restrictions on the *PP* automaton of Section 3.1 and the assumptions about the *PP* automaton of Section 3.2. We describe in detail the aspects of the physical plant model that were only abstract in Sections 3.1 and 3.2. These include: the state variables, the initial states, the discrete actions, and the trajectories of the *PP* automaton. Moreover, we define several auxiliary derived variables and sets that are used extensively by the protector automata presented in the following chapters.

The state variables of the VEHICLES automaton include the position, the velocity, and the acceleration of each vehicle and several other variables that record whether the vehicles of each of the vehicle pairs have collided into each other, whether each vehicle is braking, and whether each protector is requesting each vehicle to brake. The set of initial states is the set of states of the VEHICLES automaton that satisfy the physical properties of the system. The input actions are used by the protectors to instruct the vehicles to apply or release their "emergency" brakes, and the internal actions model the possibility that vehicles stop suddenly or collide among themselves. The trajectories model the motion of the vehicles with time, within their physical constraints.

4.1 Physical Plant: VEHICLES

In this section we describe the automaton VEHICLES, which models a set of n vehicles traveling on a single track. For simplicity, all the vehicles are assumed to have identical dimensions and acceleration/deceleration capabilities. The formal definition of the automaton VEHICLES and the formal definition of the derived variables and sets used in its definition are given in Figure 4.1 and Table 4.1, respectively. Their informal definitions follow.

The set I is the set of vehicles being modeled in the VEHICLES automaton. Each vehicle is identified by an element of this set. As described in Section 3.1, the set J is the set of ports that are used by the VEHICLES automaton to interact with the various protectors. In this setting, each of the protectors uses a single port to interact with the VEHICLES automaton. Therefore, the port index is often used to specify the protector itself.

The output variables of the VEHICLES automaton are the variables x_i, for $i \in I$, the variables $\dot{x}_i$, for $i \in I$, and the variables $collided(i, i')$, for $i, i' \in I, i' \neq i$. Each of the variables x_i, for $i \in I$, is the position of the vehicle i. The position of each vehicle i, for $i \in I$, is represented by a single point on the real line, *i.e.*, $x_i \in \mathbb{R}$, for $i \in I$, and specifies the position of the rear of the vehicle i on the track. The section of the track *occupied* by each vehicle i, for $i \in I$, often referred to as the *extent* of the vehicle i, is defined to be 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. The distance c_{len} is the minimum allowable separation between vehicles; that is, the length of the vehicle plus any desired extra margin specified by the system designer. The extent of each vehicle i, for $i \in I$, is given by the derived variable E_i; that is, $E_i = [x_i, x_i + c_{len}]$, for $i \in I$. Each of the variables $\dot{x}_i$, for $i \in I$, is the velocity of the vehicle i. The vehicles are only allowed to move forward on the track and, therefore, their velocities are restricted to be non-negative, *i.e.*, $\dot{x}_i \in \mathbb{R}^{\geq 0}$, for all $i \in I$. Once a vehicle in the VEHICLES automaton has collided, its velocity is assumed to be arbitrary.

Each output variable $collided(i, i')$, for $i' \in I, i' \neq i$, denotes whether the vehicle i has ever collided into the vehicle i'. For shorthand, each of the derived variables $collided(i, *)$, for $i \in I$, denotes whether the vehicle i has ever collided into any of the other vehicles, *i.e.*, $collided(i, *) = \bigvee_{i' \in I, i' \neq i} collided(i, i')$, and each of the derived variables $collided(*, i)$, for $i \in I$, denotes whether any of the other vehicles have ever collided into the vehicle i, *i.e.*, $collided(*, i) = \bigvee_{i' \in I, i' \neq i} collided(i', i)$. Moreover, each of the derived variables $collided(*, i, *)$, for $i \in I$, denotes whether the vehicle i has ever been involved in a collision; that is, either whether the vehicle i has ever collided into any other vehicle, or whether any other vehicle has ever collided into the vehicle i. In logical terms, $collided(*, i, *) = collided(*, i) \vee collided(i, *)$. Finally, the derived variable *collided* denotes whether any of the vehicles have ever collided among themselves, *i.e.*, *collided* =

Figure 4.1 The VEHICLES automaton.

Actions:

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

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

Variables

Internal:
$\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$
$brake(i) \in \texttt{Bool}$, for all $i \in I$, initially $\texttt{False}$
$brake\text{-}req(i,j) \in \texttt{Bool}$, for all $i \in I, j \in J$, initially $\texttt{False}$

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 $\dot{x}_i \in \mathbb{R}$
$collided(i,i') \in \texttt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\texttt{False}$

subject to $VALID$

Discrete Transitions:

$\texttt{brake}(i)_j$
Eff: $brake\text{-}req(i,j) := \texttt{True}$
if $\neg brake(i)$ then
$brake(i) := \texttt{True}$
if $\dot{x}_i = 0$ then $\ddot{x}_i := 0$
else $\ddot{x}_i := \ddot{c}_{brake}$

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

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

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

$\texttt{brick-wall}(i)$
Pre: $\texttt{True}$
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 $VALID$

$\bigvee_{i \in I} collided(i,*) = \bigvee_{i,i' \in I, i \neq i'} collided(i,i')$.

The internal variables of the VEHICLES automaton are the variables $\ddot{x}_i$, for $i \in I$, the

Table 4.1 Derived variables and sets used in the definition of the VEHICLES automaton.

$E_i \in \mathcal{P}(\mathbb{R})$, defined by

$$E_i = [x_i, x_i + c_{len}]$$

collided$(i, *) \in$ `Bool`, for $i \in I$, defined by

$$\mathit{collided}(i,*) = \bigvee_{i' \in I, i' \neq i} \mathit{collided}(i, i')$$

collided$(*, i) \in$ `Bool`, for $i \in I$, defined by

$$\mathit{collided}(*,i) = \bigvee_{i' \in I, i' \neq i} \mathit{collided}(i', i)$$

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 $i \in I$. $\}$

variables *brake*(i), for $i \in I$, and the variables *brake-req*(i, j), for $i \in I$ and $j \in J$. Each of the variables $\ddot{x}_i$, for $i \in I$, is the acceleration of the vehicle i. If no vehicle collisions involving a particular vehicle i have occurred, then (i) the acceleration of the vehicle i is bounded above and below as follows: $\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$, where $\ddot{c}_{min}, \ddot{c}_{max} \in \mathbb{R}$ and $\ddot{c}_{min} < 0 < \ddot{c}_{max}$, and (ii) if the vehicle i is braking, its acceleration is given by $\ddot{x}_i = \ddot{c}_{brake}$, where $\ddot{c}_{brake} \in \mathbb{R}$ and $\ddot{c}_{min} < \ddot{c}_{brake} < 0$. The difference between the minimum acceleration and the braking acceleration reflects a conservative estimate of the effect of a vehicle's braking system. Once a vehicle in the VEHICLES automaton has collided, its acceleration is assumed to be arbitrary and its braking system is assumed to be malfunctioning. Each of the boolean variables *brake*(i), for $i \in I$, denotes whether the vehicle i is braking. Each of the boolean variables *brake-req*(i, j), for $i \in I$ and $j \in J$, denotes whether the protector j is requesting the vehicle i to brake. It is assumed that each vehicle applies its "emergency" brake while any of the protectors is requesting it, *i.e.*, $\mathit{brake}(i) = \bigvee_{j \in J} \mathit{brake\text{-}req}(i, j)$, for all $i \in I$.

The input actions of the VEHICLES automaton are the environment action e and the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for $i \in I$ and $j \in J$. Since the VEHICLES automaton has no input variables, the environment action e is stuttering and its specification is omitted from the definition of the VEHICLES automaton. Each of the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, 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. It is important to note that the acceleration of the vehicle i is not set by the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ unless the variable $\mathit{brake}(i)$ gets toggled by the action being performed. Therefore, the $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ actions do not affect the acceleration of the vehicle i when $\mathit{brake}(i) = \texttt{True}$ and $\neg \mathit{brake}(i) \bigvee \left(\bigvee_{j' \in J, j' \neq j} \mathit{brake\text{-}req}(i, j')\right) = \texttt{True}$, respectively.

For simplicity, the set of input actions of the VEHICLES automaton includes the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for $i \in I$ and $j \in J$; that is, the VEHICLES automaton allows each protector j, for $j \in J$, to brake each vehicle i, for $i \in I$. However, it is often the case that a protector j, for some $j \in J$, need not schedule but a subset of the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for $i \in I$. In such cases, the protector j is specified as having only the output actions that it is capable of scheduling and the remaining input actions of the VEHICLES automaton on port j are ignored.

The discrete actions $\texttt{brick-wall}(i)$, for $i \in I$, $\texttt{colliding-pair}(i, i')$, for $i, i' \in I, i \neq i'$, and $\texttt{collision-effects}(i)$, for $i \in I$, are the internal actions of the VEHICLES automaton. Each $\texttt{brick-wall}(i)$ action, for $i \in I$, models the instantaneous stopping of the vehicle i — as if it hit a brick wall. Thereafter however, the vehicle i is allowed to reinitiate forward motion. The effects of the $\texttt{brick-wall}(i)$ action are to set the velocity of the vehicle i to zero and the acceleration of the vehicle i to an arbitrary non-negative value within the prespecified acceleration bounds. It is important to note that if the vehicle i was braking prior to the execution of the $\texttt{brick-wall}(i)$ action, the $\texttt{brick-wall}(i)$ action sets the acceleration of the vehicle i to zero. Each $\texttt{colliding-pair}(i, i')$ action, for $i, i' \in I, i \neq i'$, records the fact that the vehicle i has collided into the vehicle i'. The $\texttt{colliding-pair}(i, i')$ action sets the boolean variable $\mathit{collided}(i, i')$ to $\texttt{True}$. A collision between two vehicles is assumed to take place when the vehicles have overlapping extents. However, since the trailing vehicle is the only vehicle that can prevent the collision through braking, the collision is recorded only by the trailing vehicle as if the trailing vehicle were the only vehicle liable for the particular collision. Following a collision, the velocity and the acceleration of the vehicles involved in the collision are unconstrained and each vehicle's braking system is assumed to be malfunctioning. Each $\texttt{collision-effects}(i)$ 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. The $\texttt{collision-effects}(i)$ action sets the velocity and the acceleration of the vehicle i to arbitrary values. The system is modeled such that a collision allows but does not dictate immediate effects on the velocity and the acceleration of the vehicles involved in the collision; that is, $\texttt{collision-effects}(i)$ and `collision-effects`(i') actions do not necessarily follow a `colliding-pair`(i, i') action.

All discrete actions of the VEHICLES automaton, except the `collision-effects` actions, model the behavior of the vehicle as if no collisions had ever occurred. Once a vehicle has been involved in a collision, it is unknown whether the vehicle has incurred any damage and, therefore, its operation is uncertain. If the vehicle has not been damaged then its operation is modeled as if the vehicle had not collided. On the other hand, if the vehicle has been damaged, the malfunctioning vehicle apparatus is modeled by succeeding each of the discrete actions with a `collision-effects` action for the malfunctioning vehicle.

The definition of the VEHICLES automaton restricts the initial states and the trajectory states to the set *VALID*. The formal definition of the set *VALID* is given below and is included for reference in Table 4.1.

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

The restriction of the states of the VEHICLES automaton to the set *VALID* enforces some of the physical properties of the system. The first two conditions restrict the vehicle extents to be non-overlapping and the vehicle velocities to be non-negative. The vehicles are, however, allowed to "touch", *i.e.*, their extents are allowed to intersect at a single point. The final two properties only apply for vehicles that have not been involved in a collision. The third condition specifies the range of allowable vehicle acceleration and the fourth condition specifies the correct acceleration for a vehicle that is braking. Recall that once a vehicle has collided, its velocity and acceleration are assumed to be arbitrary and its braking system is assumed to be malfunctioning.

The trajectories of the VEHICLES automaton only affect the position, the velocity, and the acceleration of the vehicles of the VEHICLES automaton — the remaining variables of the VEHICLES automaton remain constant throughout the trajectories. The position and the velocity are assumed to be the integrals of the velocity and the acceleration, respectively. The acceleration is assumed to be changing arbitrarily throughout a trajectory with the restriction that all states of the trajectory remain within the set *VALID*. Finally, if a vehicle i collides into a vehicle i' for the first time, the trajectory is stopped so that the collision can be recorded by a `colliding-pair`(i, i') action.

The VEHICLES automaton complies with the assumptions made about the *PP* automaton in Section 3.2.1. The VEHICLES automaton has neither input variables, nor output actions, on any of its ports (Axioms 3.2.1 and 3.2.2, respectively). Moreover, the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for each vehicle $i \in I$ satisfying the conditions $\mathit{brake\text{-}req}(i,j) = \texttt{True}$ and $\mathit{brake\text{-}req}(i,j) = \texttt{False}$, respectively, are no-op input actions on port j for any $R \subseteq \mathit{VALID}$. Therefore, the set of no-op input actions on each port $j \in J$ and any $R \subseteq \mathit{VALID}$ is non-empty (Axiom 3.2.3).

4.2 Sets of Guarantee and Reliance for the VEHICLES Automaton

The protectors presented in the following chapters are designed to guarantee that the VEHICLES automaton remains within sets of states that are considered "good". In other words, the protectors are designed to keep the VEHICLES automaton from reaching states that are considered "bad" or hazardous. Bad or hazardous states involve vehicles that are either above the speed limit, or that have collided with each other. Sets of states that are considered "good" are informally referred to as sets of *guarantee*. Moreover, it is often the case that protectors rely on the restriction of the states of the VEHICLES automaton to sets comprised of states that exhibit particular properties of the VEHICLES automaton. Such sets of states are informally referred to as sets of *reliance*.

In the case of exceeding the speed limit, the set $P_{overspeed(i)}$ is the subset of VALID comprised of the states in which the vehicle i is above the speed limit. Let the maximum allowable velocity be given by $\dot{c}_{max}$.

$P_{overspeed(i)} \subseteq \mathit{VALID}$, for $i \in I$, defined by
$$P_{overspeed(i)} = \{p \in \mathit{VALID} \mid p.\dot{x}_i > \dot{c}_{max}\}.$$

Then the set $P_{overspeed} = \bigcup_{i \in I} P_{overspeed(i)}$ is the subset of VALID comprised of the states in which at least one of the vehicles is above the speed limit, and the set $P_{not\text{-}overspeed} = \mathit{VALID} - P_{overspeed}$ is the subset of VALID comprised of the states in which none of the vehicles are above the speed limit.

In the case of vehicle collisions, the set $P_{collided(i,i')}$ is the subset of VALID comprised of the states in which the vehicle i has collided into the vehicle i'.

$P_{collided(i,i')} \subseteq \mathit{VALID}$, for $i, i' \in I$, $i \neq i'$, defined by
$$P_{collided(i,i')} = \{p \in \mathit{VALID} \mid p.\mathit{collided}(i,i') = \texttt{True}\}.$$

Then the set $P_{collided(i)} = \bigcup_{i' \in I, i' \neq i} P_{collided(i,i')}$ is the subset of VALID comprised of the states in which the vehicle i has collided into at least one of the other vehicles. Moreover, the set $P_{collided} = \bigcup_{i \in I} P_{collided(i)} = \bigcup_{i,i' \in I, i \neq i'} P_{collided(i,i')}$ is the subset of VALID comprised of the states in which at least two distinct vehicles have collided into each other. Finally,

Table 4.2 Sets of guarantee and reliance for the VEHICLES automaton.

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

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

$P_{overspeed} \subseteq VALID$, defined by

$$P_{overspeed} = \bigcup_{i \in I} P_{overspeed(i)}$$

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

$$P_{not\text{-}overspeed} = VALID - P_{overspeed}$$

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

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

$P_{collided(i)} \subseteq VALID$, defined by

$$P_{collided(i)} = \bigcup_{i' \in I, i' \neq i} P_{collided(i,i')}$$

$P_{collided} \subseteq VALID$, defined by

$$P_{collided} = \bigcup_{i \in I} P_{collided(i)} = \bigcup_{i,i' \in I, i \neq i'} P_{collided(i,i')}$$

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

$$P_{not\text{-}collided} = VALID - P_{collided}$$

the set $P_{not\text{-}collided} = VALID - P_{collided}$ is the subset of *VALID* comprised of the states in which none of the vehicles have collided among themselves.

The sets of guarantee and reliance defined in this section comply with the assumptions made in Section 3.2.1; that is, the sets of guarantee and reliance defined in this section are Y_{VEHICLES}-determinable (Axioms 3.2.6 and 3.2.7).

For reference, the formal definitions of the sets of guarantee and reliance defined above appear in Table 4.2. These sets are extensively used in the definitions of the overspeed and collision protectors presented in the following chapters.

Table 4.3 Auxiliary derived variables for the VEHICLES automaton.

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

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

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

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

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

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

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

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

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

$$C_i(t) = \left[x_i, x_i + max\text{-}range_i(t) - max\text{-}vel_i(t)^2/(2\ddot{c}_{brake}) + c_{len}\right]$$

4.3 Auxiliary Derived Variables and Auxiliary Sets for the VEHICLES Automaton

This section presents several auxiliary derived variables and sets for the VEHICLES automaton. These variables and sets are used extensively in the following chapters.

For any state p in *VALID*, the auxiliary derived variables for any vehicle $i \in I$ and time $t \in \mathbb{R}^{\geq 0}$ are defined in Table 4.3. If the vehicle i is abiding by the global speed limit $\dot{c}_{max}$, then the derived variables of Table 4.3 can be interpreted as follows:

$stop\text{-}dist_i$, for $i \in I$, is the distance required to stop the vehicle i, assuming a braking deceleration equal to $\ddot{c}_{brake}$.

$max\text{-}range_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, is the maximum distance the vehicle i can travel in t time units, assuming a maximum acceleration equal to $\ddot{c}_{max}$.

$\mathit{max\text{-}vel}_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, is the maximum velocity achievable by the vehicle i in t time units, assuming a maximum acceleration equal to $\ddot{c}_{max}$.

O_i, for $i \in I$, is the section of the track that the vehicle i "owns"; that is, the range extending from the current position of the vehicle i to the point on the track that the vehicle can reach even if it is braked immediately.

$C_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, is 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 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.

We now define sets of states of the VEHICLES automaton that are used extensively in the following example protector chapters. While their formal definitions appear in Table 4.4, their informal interpretations are presented below. It is important to note that the interpretations of the sets $\mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$ and $\mathit{disjoint\text{-}claimed\text{-}tracks}(i, i', t)$, for $i, i' \in I, i \neq i'$, and $t \in \mathbb{R}^{\geq 0}$, are valid provided that all the vehicles of the VEHICLES automaton are abiding by the global speed limit $\dot{c}_{max}$.

$\mathit{disjoint\text{-}extents}(i, i')$, for $i, i' \in I, i \neq i'$, is the subset of VALID comprised of the states in which the extents of the vehicles i and i' are disjoint. We use P_E to denote the set of states in which the extents of all the vehicles are disjoint.

$\mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$, for $i, i' \in I, i \neq i'$, is the subset of VALID comprised of the states in which the sections of the track owned by the vehicles i and i' are disjoint. We use P_O to denote the set of states in which all vehicles own disjoint sections of the track. If a state of the VEHICLES automaton is not in P_O, then it cannot be guaranteed that the vehicles will not collide in the future; that is, irrespective of any protection action taken, it is possible for some vehicles to collide.

$\mathit{disjoint\text{-}claimed\text{-}tracks}(i, i', t)$, for $i, i' \in I, i \neq i'$, and $t \in \mathbb{R}^{\geq 0}$, is the subset of VALID comprised of the states in which the sections of the track claimed within t time units by the vehicles i and i' are disjoint. We use $P_{C(t)}$ to denote the set of states in which the sections of the track claimed within t time units by all the vehicles are disjoint. If a state of the VEHICLES automaton is not in $P_{C(t)}$ and no protective action is taken for t time units, then it cannot be guaranteed that the vehicles will subsequently not collide; that is, irrespective of any protection action taken after t time units, it is possible for some of the vehicles to collide.

Furthermore, let $P_{B_{ij}}$ be the subset of VALID comprised of the states in which the protector communicating with the VEHICLES automaton through the port j is requesting the vehicle i to brake, *i.e.*, $P_{B_{ij}} = \{p \in \mathit{VALID} \mid p.\mathit{brake\text{-}req}(i, j) = \texttt{True}\}$.

Table 4.4 Auxiliary sets for the VEHICLES automaton.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.4 Useful Lemmas for the VEHICLES Automaton

In this section we prove several useful lemmas that describe particular properties of the VEHICLES automaton and its derived variables.

Lemma 4.4.1 *For all $p \in \textit{VALID}$, $i \in I$, and $t \in \mathbb{R}^{\geq 0}$, the following hold:*

1. $p.\textit{stop-dist}_i \geq 0$.
2. $p.\textit{max-range}_i(t) \geq 0$.
3. $p.\textit{max-vel}_i(t) \geq 0$.

4. *If* $p.\dot{x}_i = 0$ *then* $p.\textit{stop-dist}_i = 0$.
5. $p.\textit{max-range}_i(0) = 0$.
6. $p.\textit{max-vel}_i(0) = p.\dot{x}_i$.

Proof: Follow directly from the definitions of the auxiliary derived variables $\textit{stop-dist}_i$, $\textit{max-range}_i(\tau)$, and $\textit{max-vel}_i(\tau)$, for $\tau \in \mathbb{R}^{\geq 0}$. ■

Lemma 4.4.2 *For all* $p \in \textit{VALID}$, $i \in I$, *and* $t, t' \in \mathbb{R}^{\geq 0}$, $t \leq t'$, *the following hold:*

1. $p.E_i \subseteq p.O_i \subseteq p.C_i(t)$.
2. $p.x_i = \min(p.E_i) = \min(p.O_i) = \min(p.C_i(t))$.
3. $p.O_i = p.C_i(0)$.
4. $p.C(t) \subseteq p.C(t')$.

Proof: Follow directly from the definitions of the derived variables E_i, O_i, and $C_i(\tau)$, for $\tau \in \mathbb{R}^{\geq 0}$. ■

Lemma 4.4.3 *If* $p, p' \in \textit{VALID}$, *where* p' *follows from* p *in a single discrete action, then the following hold:*

1. $p'.O_i \subseteq p.O_i$ *if and only if* $p'.\dot{x}_i \leq p.\dot{x}_i$.
2. $p'.C_i(t) \subseteq p.C_i(t)$, *for any* $t \in \mathbb{R}^{\geq 0}$, *if and only if* $p'.\dot{x}_i \leq p.\dot{x}_i$.

Proof: We prove each of the above statements separately.

1. Recall that $O_i = [x_i, x_i + \textit{stop-dist}_i + c_{len}]$. Since none of the actions of the VEHICLES automaton affect the position of a vehicle, it follows that $p'.x_i = p.x_i$. Therefore, the intervals $p.O_i$ and $p'.O_i$ have the same left endpoint, *i.e.*, $\min(p.O_i) = \min(p'.O_i)$. Moreover, since the variable $\textit{stop-dist}_i$ is positively correlated with the velocity of the vehicle i, it follows that $p'.\textit{stop-dist}_i \leq p.\textit{stop-dist}_i$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$; that is, $\max(p'.O_i) \leq \max(p.O_i)$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$.

   Since $\min(p.O_i) = \min(p'.O_i)$ and $\max(p'.O_i) \leq \max(p.O_i)$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$, it follows that $p'.O_i \subseteq p.O_i$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$.
2. Recall that $C_i(t) = \left[x_i, x_i + \textit{max-range}_i(t) - \textit{max-vel}_i(t)^2/(2\ddot{c}_{brake}) + c_{len}\right]$, for any $t \in \mathbb{R}^{\geq 0}$. As shown above, it is the case that $p'.x_i = p.x_i$ and, therefore, the intervals $p.C_i(t)$ and $p'.C_i(t)$ have the same left endpoint, *i.e.*, $\min(p.C_i(t)) = \min(p'.C_i(t))$. Now, consider the right endpoints of $p.C_i(t)$ and $p'.C_i(t)$. The variables $\textit{max-range}_i$

and *max-vel*$_i$ are positively correlated with the velocity of the vehicle i and, therefore, it follows that $\max(p'.C_i(t)) \leq \max(p.C_i(t))$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$.

Since $\min(p.C_i(t)) = \min(p'.C_i(t))$ and $\max(p'.C_i(t)) \leq \max(p.C_i(t))$ if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$, for any $t \in \mathbb{R}^{\geq 0}$, it follows that $p'.C_i(t) \subseteq p.C_i(t)$, for any $t \in \mathbb{R}^{\geq 0}$, if and only if $p'.\dot{x}_i \leq p.\dot{x}_i$.

■

Lemma 4.4.4 *If $p, p' \in$ VALID, where p' follows from p in a single trajectory, then the following hold:*

1. *If $p \in P_{B_{ij}}$ then $p'.O_i \subseteq p.O_i$.*
2. *If $t \in \mathbb{R}^{\geq 0}$ and $\Delta t \in [0, t]$ is the limit time of the trajectory leading from p to p', then $p'.C_i(t - \Delta t) \subseteq p.C_i(t)$.*

Proof: We prove each of the above statements separately.

1. Let $p \in P_{B_{ij}}$ and consider the left and right endpoints of the intervals $p.O_i$ and $p'.O_i$.

   The left endpoints of $p.O_i$ and $p'.O_i$ are $p.x_i$ and $p'.x_i$, respectively. Therefore, due to the non-negative constraint on the vehicle velocities, it is the case that $p.x_i \leq p'.x_i$; that is, $\min(p.O_i) \leq \min(p'.O_i)$.

   Since $p \in P_{B_{ij}}$ and because the *brake-req*(i, j) variable remains constant throughout any trajectory of the VEHICLES automaton, the vehicle i keeps braking throughout the trajectory from p to p'. From the definition of the variable *stop-dist*$_i$ it follows that $p.x_i + p.\textit{stop-dist}_i = p'.x_i + p'.\textit{stop-dist}_i$ and, therefore, the right endpoints of $p.O_i$ and $p'.O_i$ are equal; that is, $\max(p'.O_i) = \max(p.O_i)$.

   Since $\min(p.O_i) \leq \min(p'.O_i)$ and $\max(p'.O_i) = \max(p.O_i)$, we can easily conclude from the definition of O_i that $p'.O_i \subseteq p.O_i$.

2. Let $t \in \mathbb{R}^{\geq 0}$ and $\Delta t \in [0, t]$ be the limit time of the trajectory leading from p to p' and consider the left and right endpoints of the intervals $p.C_i(t)$ and $p'.C_i(t - \Delta t)$.

   The left endpoints of $p.C_i(t)$ and $p'.C_i(t - \Delta t)$ are $p.x_i$ and $p'.x_i$, respectively. Therefore, due to the non-negative constraint on the vehicle velocities, it is the case that $p.x_i \leq p'.x_i$; that is, $\min(p.C_i(t)) \leq \min(p'.C_i(t - \Delta t))$.

   Since the variables *max-range*$_i$ and *max-vel*$_i$ represent the worst case behavior of the system it is the case that $p'.x_i \leq p.x_i + p.\textit{max-range}_i(\Delta t)$ and $p'.\dot{x}_i \leq p.\textit{max-vel}_i(\Delta t)$. Since the variables *max-range*$_i$ and *max-vel*$_i$ are positively correlated with the velocity of the vehicle i and $p'.\dot{x}_i \leq p.\textit{max-vel}_i(\Delta t)$, it follows that $p'.x_i + p'.\textit{max-range}_i(t - \Delta t) \leq p.x_i + p.\textit{max-range}_i(t)$ and $p'.\textit{max-vel}_i(t - \Delta t) \leq p.\textit{max-vel}_i(t)$. Therefore, the right endpoint of $p.C_i(t)$ is at least as downstream as the right endpoint of $p'.C_i(t-\Delta t)$; that is, $\max(p'.C_i(t-\Delta t)) \leq \max(p.C_i(t))$.

Since $\min(p.C_i(t)) \leq \min(p'.C_i(t-\Delta t))$ and $\max(p'.C_i(t-\Delta t)) \leq \max(p.C_i(t))$, we can easily conclude from the definition of $C_i(\tau)$, for $\tau \in \mathbb{R}^{\geq 0}$, that $p'.C_i(t-\Delta t) \subseteq p.C_i(t)$.

■

Lemma 4.4.5 *For all $t, t' \in \mathbb{R}^{\geq 0}$, $t \leq t'$, the following hold:*

1. $P_{C(t)} \subseteq P_O \subseteq P_E$.
2. $P_{C(t')} \subseteq P_{C(t)}$.

Proof: Follow from Lemma 4.4.2 and the definitions of P_E, P_O, and $P_{C(\tau)}$, for $\tau \in \mathbb{R}^{\geq 0}$. ■

Chapter 5

Example 1: Overspeed Protection System

In this chapter, we present a protector that prevents the vehicles of the VEHICLES automaton from exceeding a prespecified speed limit. In an actual system, speed limits may vary from one region of the track to another; in this thesis, we assume a single global speed limit $\dot{c}_{max}$. We define a protector, called OS-PROT, that enforces the speed limit on all vehicles, provided that they do not collide among themselves. This protector is defined as 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 $i \in I$, is an implementation of a particular instantiation of the abstract protector automaton of Section 3.2 and guarantees that the vehicle i does not exceed the speed limit.

5.1 Protection System OS-PROT-SOLO$_i$

The OS-PROT-SOLO$_i$ automata, for $i \in I$, are vehicle-wise overspeed protectors, each of which individually guarantees that the vehicle i, for which it is responsible, does not exceed the speed limit $\dot{c}_{max}$, provided that no collisions among the vehicles occur. Each of the OS-PROT-SOLO$_i$ protectors, for $i \in I$, is an implementation of the abstract protector of Section 3.2 specialized to particular definitions of the parameters PP, S, R, G, j, and d.

The physical plant automaton, PP, is defined to be the VEHICLES automaton of Figure 4.1. The port j and the sampling period d are defined to be the port and sampling period with which the protector OS-PROT-SOLO$_i$ communicates with the VEHICLES automaton. They are assumed arbitrary and are fixed for the rest of the chapter. The set R is defined to be the set $P_{not\text{-}collided}$ defined in Section 4.2. This definition restricts the reachable states of the VEHICLES automaton to states in which no collisions among the vehicles have occurred. The set of "good" states G is defined to be the set of states in which the vehicle i is at or below the speed limit, *i.e.*, $G = VALID - P_{overspeed(i)}$. The set of states S is defined to be the set $safe_{PP,R,G,j}$ defined in Section 3.2.1; that is, the set of states of the *PP* automaton for which a single input action of *PP* on port j can guarantee that, provided no new input actions on port j are allowed, all subsequently R-reachable states will be in G. In Section 3.2.1, the definition of *safe* depended on the automaton *PP*, the sets R and G, and the port j which, at the time, were arbitrary. Here, they are defined to be the automaton VEHICLES, the sets $P_{not\text{-}collided}$ and $VALID - P_{overspeed(i)}$, and the port j, respectively; that is, we have specialized the definition of *safe* for these particular definitions of the automaton *PP*, the sets R and G, and the port j. In this chapter, we will use the notation R_i, G_i, and S_i to refer to the above definitions of the sets R, G, and S.

The OS-PROT-SOLO$_i$ protector automaton is an implementation of the abstract protector automaton $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. As is the case for the abstract protector automaton Abs_j, we define the OS-PROT-SOLO$_i$ automaton to be the composition of a sensor and a discrete controller automaton. These automata are implementations of their abstract equivalents of Figures 3.2 and 3.3, specialized however, to the above definitions of the parameters *PP*, S, R, G, j, and d. The sensor automaton is precisely the specialization of the sensor automaton of Figure 3.2 to the above definitions of the parameters *PP*, *etc.* The discrete controller automaton is defined in Figure 5.1.

It is important to note that the abstract protector automaton $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ complies with the assumptions made about the abstract protector in Section 3.2.1. In particular, since the vehicle velocity variables are output variables of the VEHICLES automaton, the set *safe* is Y_{VEHICLES}-determinable and actions that guarantee safety can be determined from the output variables Y_{VEHICLES} of the VEHICLES automaton (Axioms 3.2.4 and 3.2.5, respectively). Moreover, the sets R_i and G_i are Y_{VEHICLES}-determinable (Axioms 3.2.6 and 3.2.7, respectively) and the set of start states S_i is a subset of the set *safe* (Axiom 3.2.8), since S_i is defined to be the set *safe*.

In Section 3.1 it was shown that the abstract protector Abs_j guarantees that the physical plant *PP* remains within G starting from S given R. Similarly, the OS-PROT-SOLO$_i$ automaton guarantees that VEHICLES remains within G_i starting from S_i given R_i. This is shown in the following section.

5.2 Correctness of OS-PROT-SOLO$_i$

The main result to be shown is that OS-PROT-SOLO$_i$ $\leq Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. However, since both OS-PROT-SOLO$_i$ and $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ involve the composition of the same sensor automaton with distinct discrete controller automata, Theorem 2.7.4 applies. Therefore, it suffices to show that the discrete controller automaton of OS-PROT-SOLO$_i$ of Figure 5.1 implements the discrete controller automaton $DC(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ of

Figure 5.1 Discrete controller automaton for the protector OS-PROT-SOLO$_i$.

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

Discrete Transitions:

$\mathtt{snapshot}(y)_j$
Eff: if $(y.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max})$ then
$send_j := \mathtt{unbrake}$
else
$send_j := \mathtt{brake}$

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

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

Trajectories:
$w.send_j \equiv null$

Figure 3.3. According to Theorem 2.6.1, this follows by showing that there exists a simulation relation between the states of the discrete controller automaton of OS-PROT-SOLO$_i$ and DC(VEHICLES, S_i, R_i, G_i, j, d). We first give some useful set definitions, then prove some lemmas, and finally show the existence of such a simulation relation.

In this section, we use the notation *future*$_i$, *safe*$_i$, *very-safe*$_i$, and *delay-safe*$_i$ to denote the specialization of the function *future*, the sets *safe* and *very-safe*, and the function *delay-safe*, which are defined in Section 3.2.1, to the automaton VEHICLES, the sets R_i and G_i, and the port j of the OS-PROT-SOLO$_i$ protector. Moreover, since the environment action of the VEHICLES automaton is stuttering, its consideration is omitted in all inductive proofs involving the PP automaton.

We proceed by defining several sets that are used in the correctness proof of the protector OS-PROT-SOLO$_i$. For reference, their formal definitions appear in Table 5.1.

Let W_i be the set of states of the VEHICLES automaton in which none of the vehicles have collided and the vehicle i is at or below the speed limit; that is, $W_i = R_i \cap G_i$. Let V_i be the set of states of the VEHICLES automaton in which none of the vehicles have collided, the vehicle i is at or below the speed limit, and the protector j is requesting the

Table 5.1 Sets used in the correctness proof of OS-PROT-SOLO$_i$.

$W_i \subseteq VALID$, for $i \in I$, defined by

$$W_i = R_i \cap G_i$$

$V_i \subseteq VALID$, for $i \in I$, defined by

$$V_i = R_i \cap G_i \cap P_{B_{ij}}$$

$T_i \subseteq VALID$, for $i \in I$, defined by

$$T_i = \{p \in R_i \cap G_i \mid p.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}\}$$

vehicle i to brake; that is, $V_i = R_i \cap G_i \cap P_{B_{ij}}$. Furthermore, let T_i be the set of states of the VEHICLES automaton in which none of the vehicles have collided, the vehicle i is at or below the speed limit, and the condition $\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}$ is satisfied; that is, $T_i = \{p \in R_i \cap G_i \mid p.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}\}$.

In the following lemma, we show that if we restrict the states of the VEHICLES automaton to the set R_i and consider a state in which the vehicle i is at or below the speed limit and is being requested to brake by the protector j, then, provided that no new protective actions are issued by the protector j, the vehicle i remains at or below the speed limit thereafter.

Lemma 5.2.1 $\mathit{future}_i(V_i, \mathbb{R}^{\geq 0}) \subseteq G_i$.

Proof: Let α be an execution fragment of the VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in V_i, is only comprised of states in R_i, and involves no input actions on port j. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_i$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$. Since $p_{init} \in V_i$ and $V_i \subseteq G_i$, it follows that $p_{final} \in G_i$.

The inductive step involves showing that if α is an execution fragment of length $n = k + 1$, for some $k \in \mathbb{N}$, then $p_{final} \in G_i$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{final} \in G_i$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, we consider all possible discrete actions by cases:

1. the actions `brake`$(i)_j$ and `unbrake`$(i)_j$ are not enabled because α involves no input actions on port j.
2. the `brick-wall`(i) action sets the velocity of the vehicle i to zero. Therefore, it trivially follows that $p_{final} \in G_i$.
3. the actions `colliding-pair`(i', i''), for $i', i'' \in I, i' \neq i''$, and `collision-effects`$(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i; recall that $R_i = P_{not\text{-}collided}$.
4. the actions `brake`$(i')_{j'}$, `unbrake`$(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and `brick-wall`$(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i; that is, $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$. From the induction hypothesis we have that $p'_{final} \in G_i$ and, therefore, it follows that $p_{final} \in G_i$.

In the case of a trajectory, since the execution fragment α starts in a state in $V_i \subseteq P_{B_{ij}}$ and the only action that can set the *brake-req*(i, j) variable to `False` is not enabled throughout α, all states in α are in $P_{B_{ij}}$; that is, the vehicle i keeps braking throughout the execution fragment α. Therefore, since the vehicle i in state p'_{final} is in G_i, *i.e.*, at or below the speed limit, and the vehicle i is braking throughout the trajectory from p'_{final} to p_{final}, it trivially follows that the velocity of the vehicle i in p_{final} will be at or below the speed limit; that is, $p_{final} \in G_i$. ■

In the following two lemmas, we use Lemma 5.2.1 to show that $V_i \subseteq$ *very-safe*$_i$ and $V_i \subseteq$ *delay-safe*$_i(t)$, for any $t \in \mathbb{R}^{\geq 0}$, respectively.

Lemma 5.2.2 $V_i \subseteq$ *very-safe*$_i$.

Proof: From the definition of *very-safe* in Section 3.2.1, we must show that the condition *future*$_i(V_i, \mathbb{R}^{\geq 0}) \subseteq G_i$ is satisfied. This follows directly from Lemma 5.2.1. ■

Lemma 5.2.3 *For any* $t \in \mathbb{R}^{\geq 0}$, *it is the case that* $V_i \subseteq$ *delay-safe*$_i(t)$.

Proof: Follows directly from Lemma 5.2.2 and Lemma 3.2.5, part 1. ■

In the following two lemmas and the subsequent corollary, we show that the sets W_i and *safe*$_i$ are equal. First, we show that $W_i \subseteq$ *safe*$_i$ and *safe*$_i \subseteq W_i$. Then the fact that $W_i =$ *safe*$_i$ follows trivially.

Lemma 5.2.4 $W_i \subseteq$ *safe*$_i$.

Proof: From the definition of *safe* in Section 3.2.1, we must show that any state $p \in W_i$ satisfies: (i) $\mathit{future}_i(p, 0) \subseteq G_i$, and (ii) there exists some action π such that for every $p', p'' \in R_i$ satisfying $p' \in \mathit{future}_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in \mathit{very\text{-}safe}_i$.

For the first condition, let α be an execution fragment of the VEHICLES automaton of n steps, where $n \in \mathbb{N}$, that: starts in a state in W_i, is only comprised of states in R_i, involves no input actions on port j, and has a limit time equal to zero. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{\mathit{final}} \in G_i$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of no steps and, therefore, $p_{\mathit{final}} = p_{init}$. Since $p_{init} \in W_i$, it follows that $p_{\mathit{final}} \in G_i$.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{\mathit{final}} \in G_i$. Let α' be the part of the execution fragment α comprised of the first k steps. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{\mathit{final}} \in G_i$. Since the final state of α is reached from the final state of α' by a single step, the inductive step involves the consideration of all possible steps leading from p'_{final} to p_{final}.

To complete the induction, we consider all possible discrete actions by cases:

1. the actions $\mathtt{brake}(i)_j$ and $\mathtt{unbrake}(i)_j$ are not enabled because α involves no input actions on port j.
2. the $\mathtt{brick\text{-}wall}(i)$ action sets the velocity of the vehicle i to zero. Therefore, it trivially follows that $p_{\mathit{final}} \in G_i$.
3. the actions $\mathtt{colliding\text{-}pair}(i', i'')$, for $i', i'' \in I, i' \neq i''$, and $\mathtt{collision\text{-}effects}(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i; recall that $R_i = P_{not\text{-}collided}$.
4. the actions $\mathtt{brake}(i')_{j'}$, $\mathtt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and $\mathtt{brick\text{-}wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i; that is, $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. However, from the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_i$. Therefore, it trivially follows that $p_{\mathit{final}} \in G_i$.

For the second condition, consider the action $\pi = \mathtt{brake}(i)_j$. The effect of this action is to set the internal variable $\mathit{brake\text{-}req}(i, j)$ to $\mathtt{True}$. Therefore, it is the case that $p'' \in P_{B_{ij}}$. From the proof of the first condition, it is the case that $p' \in G_i$, and since the $\mathtt{brake}(i)_j$ action does not affect the velocity of the vehicle i, it is also the case that $p'' \in G_i$. From the above conditions and the fact that $p'' \in R_i$, it follows that $p'' \in V_i$. Finally, Lemma 5.2.2 implies that $p'' \in \mathit{very\text{-}safe}_i$, as needed. ■

Lemma 5.2.5 $\mathit{safe}_i \subseteq W_i$.

Proof: From Lemma 3.2.4, part 1, and the definition of *safe* in Section 3.2.1, it is the case that $safe_i \subseteq G_i$ and $safe_i \subseteq R_i$, respectively. It trivially follows that $safe_i \subseteq W_i$. ∎

Corollary 5.2.6 $W_i = safe_i$.

Proof: Follows directly from Lemmas 5.2.4 and 5.2.5. ∎

In the next three lemmas, we show that any state p in the set T_i is in the set $delay\text{-}safe_i(d)$; that is, any state R_i-reachable from p within an amount of time d through an execution fragment that involves no input actions on port j, is in the set G_i and any state R_i-reachable from the state p in exactly an amount of time d through an execution fragment that involves no input actions on port j, is in the set $safe_i$.

Lemma 5.2.7 $future_i(T_i, [0, d]) \subseteq G_i$.

Proof: Let α be an execution fragment of the VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in T_i, is only comprised of states in R_i, involves no input actions on port j, and has a limit time t that lies in the interval $[0, d]$. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_i$.

We use induction on the length n of the execution fragment α and the assertion $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$ to show that $p_{final} \in G_i$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$ and $p_{final}.\dot{x}_i = p_{init}.\dot{x}_i$. Moreover, since $t = 0$, it is the case that $t\ddot{c}_{max} = 0$. It trivially follows that $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, and $t' \in [0, t]$ is the limit time of α', then it is the case that $p'_{final}.\dot{x}_i \leq p'_{init}.\dot{x}_i + t'\ddot{c}_{max}$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, keeping in mind that the limit times of α' and α are equal, *i.e.*, $t' = t$, we consider all possible discrete actions by cases:

1. the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ are not enabled because α involves no input actions on port j.

2. the `brick-wall`(i) action sets the velocity of the vehicle i to zero and since all vehicle velocities are restricted to be non-negative, it follows that $p_{final}.\dot{x}_i \leq p'_{final}.\dot{x}_i$. Moreover, from the induction hypothesis, we have $p'_{final}.\dot{x}_i \leq p'_{init}.\dot{x}_i + t'\ddot{c}_{max}$. Since $p_{init} = p'_{init}$ and $t = t'$, it follows that $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$.
3. the actions `colliding-pair`(i', i''), for $i', i'' \in I, i' \neq i''$, and `collision-effects`$(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i; recall that $R_i = P_{not\text{-}collided}$.
4. the actions `brake`$(i')_{j'}$, `unbrake`$(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and `brick-wall`$(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i; that is, $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$. Moreover, from the induction hypothesis we have $p'_{final}.\dot{x}_i \leq p'_{init}.\dot{x}_i + t'\ddot{c}_{max}$. Since $p_{init} = p'_{init}$ and $t = t'$, it follows that $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$.

In the case of a trajectory, since the change in velocity is equal to the integral of the acceleration and the acceleration is bounded from above by the quantity $\ddot{c}_{max}$, it is the case that $p_{final}.\dot{x}_i \leq p'_{final}.\dot{x}_i + (t - t')\ddot{c}_{max}$. Moreover, from the induction hypothesis we have $p'_{final}.\dot{x}_i \leq p'_{init}.\dot{x}_i + t'\ddot{c}_{max}$. Since $p_{init} = p'_{init}$, it follows that $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$. This result completes the induction.

Since $p_{init} \in T_i$, it is the case that $p_{init}.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}$. Moreover, from the above induction we have $p_{final}.\dot{x}_i \leq p_{init}.\dot{x}_i + t\ddot{c}_{max}$. Therefore, $p_{final}.\dot{x}_i \leq \dot{c}_{max} - (d - t)\ddot{c}_{max}$, and since $d - t \geq 0$ and $\ddot{c}_{max} > 0$, it follows that $p_{final}.\dot{x}_i \leq \dot{c}_{max}$; that is, $p_{final} \in G_i$, as needed. ■

Lemma 5.2.8 *$future_i(T_i, 0) \subseteq T_i$.*

Proof: From Lemma 5.2.4 and the definition of T_i it is the case that $T_i \subseteq safe_i$. Therefore, from Lemma 3.2.4, part 2, it follows that $future_i(T_i, 0) \subseteq safe_i$. Moreover, Lemma 5.2.5 implies that $future_i(T_i, 0) \subseteq W_i$. It remains to be shown that for all $p, p' \in R_i$ such that $p \in T_i$ and $p' \in future_i(p, 0)$, it is the case that $p'.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}$.

Because of the non-negative constraint on the vehicle velocities, the only discrete action that could potentially increase the velocity of the vehicle i is the `collision-effects`(i) action. However, the `collision-effects`(i) action is not enabled because the function $future_i(p, 0)$ only considers R_i-reachable states. If follows that $p'.\dot{x}_i \leq p.\dot{x}_i$. Moreover, since $p \in T_i$, it is the case that $p.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}$. It trivially follows that $p'.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max}$, as needed. ■

Lemma 5.2.9 *$T_i \subseteq delay\text{-}safe_i(d)$.*

Proof: We must show that $future_i(T_i, [0, d]) \subseteq G_i$ and $future_i(T_i, d) \subseteq safe_i$. The first condition follows directly from Lemma 5.2.7. For the second condition, from Lemma 3.2.1, part 1, we have that $future_i(T_i, d) \subseteq future_i(T_i, [0, d])$. Therefore, from Lemma 5.2.7 it follows that $future_i(T_i, d) \subseteq G_i$. Moreover, since $future_i(T_i, d)$ restricts the reachable states to the set R_i, it is the case that $future_i(T_i, d) \subseteq R_i$. Therefore, it is the case that $future_i(T_i, d) \subseteq W_i$ and from Lemma 5.2.4 it follows that $future_i(T_i, d) \subseteq safe_i$, as needed. ∎

In the following lemma, we show that the OS-PROT-SOLO$_i$ protector implements the protector $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. Since the protector automata OS-PROT-SOLO$_i$ and Abs_j involve the composition of the same sensor automaton with distinct controller automata, it suffices to show that the discrete controller automaton of the protector OS-PROT-SOLO$_i$ implements the discrete controller automaton $DC(\text{VEHICLES}, S_i, R_i, G_i, j, d)$.

Lemma 5.2.10 OS-PROT-SOLO$_i$ $\leq Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$.

Proof: As noted above, both the OS-PROT-SOLO$_i$ and the Abs_j protectors involve the composition of the same sensor automaton with distinct controller automata. From Theorem 2.7.4, it suffices to show that the discrete controller automaton of OS-PROT-SOLO$_i$ implements DC_j. This is shown by a simulation from the discrete controller automaton of OS-PROT-SOLO$_i$ to DC_j.

The mapping between the states of the discrete controller automaton of OS-PROT-SOLO$_i$ and DC_j is almost the identity. In the discrete controller automaton of OS-PROT-SOLO$_i$, the variable $send_j$ is equal to either one of the labels `brake` and `unbrake`, or the value *null*. In the abstract discrete controller automaton, these valuations simply map to either the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, or the value *null*, respectively.

The start states for the discrete controller automaton of OS-PROT-SOLO$_i$ and DC_j are the states in which $send_j = null$. These are mapped to each other according to the mapping discussed above.

Furthermore, since the trajectories in both discrete controller automata are identical, we need only consider their discrete transitions. We analyze the actions of the implementation by cases, letting p denote any complete state of the VEHICLES automaton that corresponds to y, *i.e.*, $p \in VALID$ and $p \lceil Y_{\text{VEHICLES}} = y$.

1. The $\texttt{snapshot}(y)_j$ action of the implementation sets $send_j$ to `brake`, or `unbrake`. In order to show that the behavior of the implementation is allowed by the specification, we must show that the input action $\texttt{snapshot}(y)_j$ of the implementation sets the value of the $send_j$ variable in such a way that the subsequently enabled action π of the implementation (i) guarantees that for all $p', p'' \in R_i$ such that $p' \in future_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in delay\text{-}safe_i(d)$, if $p \in safe_i$, and (ii) is an arbitrary output action of the implementation, otherwise.

   First, consider the case in which $p \in safe_i$. Since Corollary 5.2.6 implies that $p \in W_i$, the discrete controller automaton of OS-PROT-SOLO$_i$ sets the variable $send_j$ according to whether the state p is in T_i, or not.

On one hand, if $p \notin T_i$ then the discrete controller automaton of OS-PROT-SOLO$_i$ sets the variable $send_j$ to `brake` and the $\texttt{brake}(i)_j$ action is enabled. However, since $p \in safe_i$, Lemma 3.2.4, part 2, implies that $p' \in safe_i$ and from Corollary 5.2.6 it follows that $p' \in W_i$. Moreover, since the $\texttt{brake}(i)_j$ action sets the $brake\text{-}req(i, j)$ variable to `True` and affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it is the case that $p'' \in R_i \cap G_i \cap P_{B_{ij}}$, *i.e.*, $p'' \in V_i$. Finally, from Lemma 5.2.3, it follows that $p'' \in delay\text{-}safe_i(d)$, as needed.

On the other hand, if $p \in T_i$ then the discrete controller automaton of OS-PROT-SOLO$_i$ sets the variable $send_j$ to `unbrake` and the $\texttt{unbrake}(i)_j$ action is enabled. From Lemma 5.2.8, it follows that $p' \in T_i$. Moreover, since the $\texttt{unbrake}(i)_j$ action sets the $brake\text{-}req(i, j)$ variable to `False` and affects neither the velocity of the vehicle i, nor any of the *collided* variables, it is the case that $p'' \in T_i$. Finally, from Lemma 5.2.9, it follows that $p'' \in delay\text{-}safe_i(d)$, as needed.

Next, consider the case in which $p \notin safe_i$. In this case, the $\texttt{snapshot}(y)_j$ action of the discrete controller automaton of OS-PROT-SOLO$_i$ sets the variable $send_j$ to either `brake` or `unbrake` and, subsequently, enables either the action $\texttt{brake}(i)_j$ or the action $\texttt{unbrake}(i)_j$. However, when $p \notin safe_i$, the DC_j automaton sets the variable $send_j$ arbitrarily and, subsequently, enables an arbitrary output action. Therefore, the behavior of the discrete controller automaton of OS-PROT-SOLO$_i$ is allowed by that of the DC_j automaton.

Therefore, the effects of the $\texttt{snapshot}(y)_j$ action of the implementation are allowed by its specification.

2. The $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ actions have identical effects in both discrete controller automata. When the $send_j$ variable matches the labels `brake` and `unbrake`, or the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, the respective action is performed and the $send_j$ variable is set to the value *null* in both discrete controller automata.

3. The environment action in both discrete controller automata is stuttering. It follows that the mapping between the states of the discrete controller automaton of OS-PROT-SOLO$_i$ and the DC_j automaton prior to and succeeding the execution of the environment action remains the same.

■

Corollary 5.2.11 *The protector* OS-PROT-SOLO$_i$ *guarantees* G_i *in the* VEHICLES *automaton starting from* S_i *given* R_i.

Proof: Follows directly from Lemma 5.2.10 and Theorem 3.2.9. ■

Table 5.2 Formal definitions of OS-PROT, $G_{\text{OS-PROT}}$, $S_{\text{OS-PROT}}$, and $R_{\text{OS-PROT}}$.

$$\text{OS-PROT} \equiv \prod_{i \in I} \text{OS-PROT-SOLO}_i$$

$$G_{\text{OS-PROT}} \equiv \bigcap_{i \in I} G_i$$

$$S_{\text{OS-PROT}} \equiv \bigcap_{i \in I} S_i$$

$$R_{\text{OS-PROT}} \equiv P_{not\text{-}collided}$$

5.3 Protection System OS-PROT

We now define the overspeed protector OS-PROT. As in the vehicle-wise case, we restrict the states of the VEHICLES automaton to the set $P_{not\text{-}collided}$ as defined in Section 4.2, *i.e.*, $R_{\text{OS-PROT}} = P_{not\text{-}collided}$. Let $G_{\text{OS-PROT}}$ and $S_{\text{OS-PROT}}$ be the intersection of the sets G_i and S_i, for all $i \in I$, respectively, and OS-PROT be the composition of the protectors OS-PROT-SOLO$_i$, for all $i \in I$. The protector OS-PROT guarantees that the VEHICLES automaton remains within $G_{\text{OS-PROT}}$ starting from $S_{\text{OS-PROT}}$ given $R_{\text{OS-PROT}}$. For reference, The formal definition of the OS-PROT automaton and of the sets $G_{\text{OS-PROT}}$, $S_{\text{OS-PROT}}$, and $R_{\text{OS-PROT}}$ are shown in Table 5.2.

Corollary 5.3.1 *The protector* OS-PROT *guarantees* $G_{\text{OS-PROT}}$ *in the* VEHICLES *automaton starting from* $S_{\text{OS-PROT}}$ *given* $R_{\text{OS-PROT}}$.

Proof: Follows directly from Corollary 5.2.11 and Theorem 3.1.4. ■

Chapter 6

Example 2: Collision Avoidance on a Single Track

This chapter is similar to Chapter 5; instead of an overspeed protector, here we present a collision protector for the VEHICLES automaton. We define the protector CL-PROT that guarantees that none of the vehicles collide, provided that they are all abiding by the speed limit. The CL-PROT protector is defined as 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 CL-PROT-SOLO$_i$ protectors, for $i \in I$, is an implementation of a particular instantiation of the abstract protector automaton of Section 3.2 and guarantees that the vehicle i does not collide into any of the vehicles it trails.

6.1 Protection System CL-PROT-SOLO$_i$

The CL-PROT-SOLO$_i$ automata are vehicle-wise collision protectors and individually guarantee that the vehicle i does not collide into any of the vehicles it trails, provided that all vehicles 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. Each of the CL-PROT-SOLO$_i$ protectors, for $i \in I$, is an implementation of the abstract protector of Section 3.2 specialized to particular definitions of the parameters PP, S, R, G, j, and d.

The physical plant automaton, PP, is defined to be the VEHICLES automaton of Figure 4.1. The port j and the sampling period d are defined to be the port and sampling period with which the protector CL-PROT-SOLO$_i$ communicates with the VEHICLES automaton and are assumed arbitrary. The set of "good" states G is defined to be the set of states in which the vehicle i has not collided into any of the other vehicles, *i.e.*, $G = VALID - P_{collided(i)}$. In this chapter, we use the notation G_i to refer to this definition of the set G. The set R is defined to be the set $R = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i' \in I, i' \neq i} G_{i'} \right)$. This definition restricts the states of the VEHICLES automaton to states in which all of the vehicles are abiding by the speed limit and in which each of the remaining vehicles has never collided into any other vehicle. The set of states S is defined to be the set *safe* defined in Section 3.2.1; that is, the set of states of the PP automaton for which a single input action of PP on port j can guarantee that, provided no new input actions on port j are allowed, all subsequently R-reachable states will be in G. Once again, the definition of the set *safe* is specialized to the above definitions of the automaton PP, the sets R and G, and the port j. In this chapter, we use the notation R_i and S_i to refer to the above definitions of the sets R and S.

The CL-PROT-SOLO$_i$ protector automaton is an implementation of the abstract protector automaton $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. More precisely, as is the case for the abstract protector Abs_j, we define the CL-PROT-SOLO$_i$ automaton to be the composition of a sensor and a discrete controller automaton. These automata are implementations of their abstract equivalents of Figures 3.2 and 3.3, specialized however, to the above definitions of the parameters PP, S, R, G, j, and d. The sensor automaton is precisely the specialization of the sensor automaton of Figure 3.2 to the above definitions of the parameters PP, *etc.* The discrete controller automaton is defined in Figure 6.1.

The braking strategy of the CL-PROT-SOLO$_i$ protector is as follows. The protector instructs the vehicle i to brake if it has a d time unit claim overlap with any of the vehicles it trails; that is, the protector instructs the vehicle i to brake if there exists a vehicle i', for $i' \in I, i' \neq i$, such that the sections of the track claimed by the vehicles i and i' in time d overlap and $x_i < x_{i'}$. 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.

It is important to note that the abstract protector automaton $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ complies with the assumptions made about the abstract protector in Section 3.2.1. In particular, since the vehicle position variables, the vehicle velocity variables, and the *collided* variables are output variables of the VEHICLES automaton, the set *safe* is Y_{VEHICLES}-determinable and actions that guarantee safety can be determined from the output variables Y_{VEHICLES} of the VEHICLES automaton (Axioms 3.2.4 and 3.2.5, respectively). Moreover, the sets R_i and G_i are Y_{VEHICLES}-determinable (Axioms 3.2.6 and 3.2.7, respectively) and the set of start states S_i is a subset of the set *safe* (Axiom 3.2.8), since S_i is defined to be the set *safe*.

In Section 3.1 it was shown that the abstract protector Abs_j guarantees that the physical plant PP remains within G starting from S given R. Similarly, the CL-PROT-SOLO$_i$ automaton guarantees that the VEHICLES automaton remains within G_i starting from S_i given R_i. This is shown in the following section.

Figure 6.1 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:

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

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

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

Trajectories:
$w.send_j \equiv null$

6.2 Correctness of CL-PROT-SOLO$_i$

The main result to be shown is that CL-PROT-SOLO$_i \leq Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. Since both CL-PROT-SOLO$_i$ and $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ involve the composition of the same sensor automaton with distinct discrete controller automata, Theorem 2.7.4 applies. Therefore, it suffices to show that the discrete controller automaton of CL-PROT-SOLO$_i$ of Figure 6.1 implements the discrete controller automaton $DC(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ of Figure 3.3. According to Theorem 2.6.1, this follows by showing that there exists a simulation relation between the states of the discrete controller automaton of CL-PROT-SOLO$_i$ and the discrete controller automaton $DC(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. We first give some useful set definitions, then prove some lemmas, and finally show the existence of such a simulation relation. The correctness proof follows the steps of the correctness proof of Section 5.2.

In this section, we use the notation $\textit{future}_i$, $\textit{safe}_i$, $\textit{very-safe}_i$, and $\textit{delay-safe}_i$ to denote the specialization of the function *future*, the sets *safe* and *very-safe*, and the function *delay-safe*, which are defined in Section 3.2.1, to the automaton VEHICLES, the sets R_i and G_i, and

Table 6.1 Sets used in the correctness proof of CL-PROT-SOLO$_i$.

$W_i \subseteq VALID$, for $i \in I$, defined by

$$W_i = \{p \in R_i \cap G_i \mid \nexists\, i' \in I, i' \neq i : p.O_i \cap p.O_{i'} \neq \emptyset \ \wedge\ p.x_i < p.x_{i'}\}$$

$V_i \subseteq VALID$, for $i \in I$, defined by

$$V_i = W_i \cap P_{B_{ij}}$$

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

$$T_i(t) = \{p \in R_i \cap G_i \mid \nexists\, i' \in I, i' \neq i : p.C_i(t) \cap p.C_{i'}(t) \neq \emptyset \ \wedge\ p.x_i < p.x_{i'}\}$$

the port j of the CL-PROT-SOLO$_i$ protector. Moreover, since the environment action of the VEHICLES automaton is stuttering, its consideration is omitted in all inductive proofs involving the PP automaton.

We proceed by defining several sets that are used in the correctness proof of the protector CL-PROT-SOLO$_i$. For reference, their formal definitions appear in Table 6.1.

Let W_i be the subset of $R_i \cap G_i$ comprised of the states in which the section of the track owned by the vehicle i does not overlap the section of track owned by any of the vehicles it trails; that is, for every state p in W_i, $p \in R_i \cap G_i$ and there does not exist $i' \in I, i' \neq i$ such that $p.O_i \cap p.O_{i'} \neq \emptyset$ and $p.x_i < p.x_{i'}$.

Let V_i be the subset of W_i comprised of the states in which the protector j is requesting the vehicle i to brake; that is, $V_i = W_i \cap P_{B_{ij}}$.

Let $T_i(t)$, where $t \in \mathbb{R}^{\geq 0}$, be the subset of $R_i \cap G_i$ comprised of the states in which the section of the track claimed in time t by the vehicle i does not overlap the section of the track claimed in time t by any of the vehicles it trails; that is, for every state p in $T_i(t)$, $p \in R_i \cap G_i$ and there does not exist $i' \in I, i' \neq i$ such that $p.C_i(t) \cap p.C_{i'}(t) \neq \emptyset$ and $p.x_i < p.x_{i'}$.

Lemma 6.2.1 *For all $t, t' \in \mathbb{R}^{\geq 0}$, $t \leq t'$, the following hold:*

1. $T_i(t) \subseteq W_i \subseteq G_i$.
2. $V_i \subseteq W_i \subseteq G_i$.
3. $T_i(t') \subseteq T_i(t)$.
4. $T_i(0) = W_i$.

Proof: Follow directly from the definitions of the sets G_i, W_i, and $T_i(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and Lemma 4.4.2. ■

In the following three lemmas, we show that any state R_i-reachable from a state in V_i through an execution fragment that involves no input actions on port j, is in V_i. In the first lemma, we show that if the final state of such an execution fragment is in G_i and the section of track owned by the vehicle i has not grown since the beginning of the execution fragment, then the final state of the execution fragment is in V_i. In the second lemma, we show that the final state of any such execution fragment is in G_i and the section of track owned by the vehicle i does not grow throughout the execution fragment. Finally, the third lemma combines these two results and states formally the desired property.

Lemma 6.2.2 *Let $p \in V_i$ and $p' \in future_i(p, \mathbb{R}^{\geq 0})$. If $p' \in G_i$ and $p'.O_i \subseteq p.O_i$ then $p' \in V_i$.*

Proof: We need to show that $p' \in R_i \cap G_i \cap P_{B_{ij}}$ and that there does not exist $i' \in I, i' \neq i$ such that $p'.O_i \cap p'.O_{i'} \neq \emptyset$ and $p'.x_i < p'.x_{i'}$. We consider these conditions by cases:

1. $p' \in R_i$.

   This is the case because the function $future_i(p, \mathbb{R}^{\geq 0})$ only considers R_i-reachable states.

2. $p' \in G_i$.

   This is true by assumption.

3. $p' \in P_{B_{ij}}$.

   Since $p \in P_{B_{ij}}$, it is the case that $p.brake\text{-}req(i, j) =$ `True`. Moreover, the $brake\text{-}req(i, j)$ variable can only be set to `False` by an `unbrake`$(i)_j$ action — an action not allowed by the function $future_i(p, \mathbb{R}^{\geq 0})$. Therefore, it follows that $p' \in P_{B_{ij}}$, as needed.

4. $\nexists\, i' \in I, i' \neq i$, such that $p'.O_i \cap p'.O_{i'} \neq \emptyset$ and $p'.x_i < p'.x_{i'}$.

   Because $p \in V_i$ we have that for all $i' \in I, i' \neq i$ such that $p.x_i < p.x_{i'}$ it is the case that $p.O_i \cap p.O_{i'} = \emptyset$; that is, for all $i' \in I, i' \neq i$ such that $p.x_i < p.x_{i'}$ it is the case that $\max(p.O_i) < \min(p.O_{i'})$. However, by assumption it is the case that $p'.O_i \subseteq p.O_i$. Therefore, since the vehicle velocities are restricted to be non-negative, it follows that for all $i' \in I, i' \neq i$ such that $p'.x_i < p'.x_{i'}$ it is the case that $\max(p'.O_i) < \min(p'.O_{i'})$. This is sufficient to guarantee that there does not exist $i' \in I, i' \neq i$ such that $p'.O_i \cap p'.O_{i'} \neq \emptyset$ and $p'.x_i < p'.x_{i'}$.

■

Lemma 6.2.3 *For all $p \in V_i$, if $p' \in future_i(p, \mathbb{R}^{\geq 0})$, then $p' \in G_i$ and $p'.O_i \subseteq p.O_i$.*

Proof: Let α be an execution fragment of the VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in V_i, is only comprised of states in R_i, and involves no input actions on port j. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{\mathit{final}} \in G_i$ and $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{\mathit{final}} = p_{init}$. From Lemma 6.2.1, part 2, and the fact that $p_{init} \in V_i$, it follows that $p_{\mathit{final}} \in G_i$. Moreover, the fact that $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$ is trivially true.

The inductive step involves showing that if α is an execution fragment of length $n = k + 1$, for some $k \in \mathbb{N}$, then $p_{\mathit{final}} \in G_i$ and $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{\mathit{final}} \in G_i$ and $p'_{\mathit{final}}.O_i \subseteq p'_{init}.O_i$. Moreover, from Lemma 6.2.2 it follows that $p'_{\mathit{final}} \in V_i$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, we consider all possible discrete actions by cases:

1. the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and does not affect the variables $\mathit{collided}(i, i')$, for $i' \in I, i' \neq i$.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_i$. Therefore, since the internal action $\texttt{brick-wall}(i)$ does not affect the variables $\mathit{collided}(i, i')$, for $i' \in I, i' \neq i$, it follows that $p_{\mathit{final}} \in G_i$.

   Moreover, since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$, as needed.
3. the actions $\texttt{brake}(i')_{j'}$, $\texttt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, affect neither the velocity of the vehicle i, nor the variables $\mathit{collided}(i, i''')$, for $i''' \in I, i''' \neq i$.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_i$. Therefore, since the actions $\texttt{brake}(i')_{j'}$, $\texttt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the variables $\mathit{collided}(i, i''')$, for $i''' \in I, i''' \neq i$, it follows that $p_{\mathit{final}} \in G_i$.

Moreover, since the input actions $\mathtt{brake}(i')_{j'}$, $\mathtt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and the internal actions $\mathtt{brick\text{-}wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{\mathit{init}}.O_i$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{\mathit{init}}.O_i$, as needed.

4. the actions $\mathtt{colliding\text{-}pair}(i', i'')$, for $i', i'' \in I, i' \neq i''$, and $\mathtt{collision\text{-}effects}(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i and $p'_{\mathit{final}} \in V_i$.

In the case of a trajectory, since $p' \in V_i$ and $V_i \subseteq P_{B_{ij}}$, Lemma 4.4.4, part 1, implies that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{\mathit{init}}.O_i$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{\mathit{init}}.O_i$. Moreover, since $p'_{\mathit{final}} \in G_i$ and the variables $\mathit{collided}(i, i')$, for all $i' \in I, i' \neq i$, remain constant throughout the trajectory, it follows that $p_{\mathit{final}} \in G_i$, as needed. ■

Lemma 6.2.4 $\mathit{future}_i(V_i, \mathbb{R}^{\geq 0}) \subseteq V_i$.

Proof: Follows directly from Lemmas 6.2.2 and 6.2.3. ■

In the following two lemmas, we use Lemma 6.2.4 to show that $V_i \subseteq \mathit{very\text{-}safe}_i$ and $V_i \subseteq \mathit{delay\text{-}safe}_i(t)$, for any $t \in \mathbb{R}^{\geq 0}$, respectively.

Lemma 6.2.5 $V_i \subseteq \mathit{very\text{-}safe}_i$.

Proof: From the definition of *very-safe* in Section 3.2.1, we must show that the condition $\mathit{future}_i(V_i, \mathbb{R}^{\geq 0}) \subseteq G_i$ is satisfied. This follows directly from Lemma 6.2.4 and Lemma 6.2.1, part 2. ■

Lemma 6.2.6 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $V_i \subseteq \mathit{delay\text{-}safe}_i(t)$.*

Proof: Follows directly from Lemma 6.2.5 and Lemma 3.2.5, part 1. ■

In the next three lemmas and the subsequent corollary, we show that the sets W_i and safe_i are equal. First, we show that any state that is R_i-reachable from a state p in W_i through an execution fragment that involves no input actions on port j and has a limit time equal to zero, is in the set W_i. Then, we show that $W_i \subseteq \mathit{safe}_i$ and $\mathit{safe}_i \subseteq W_i$. Finally, the subsequent corollary states that $W_i = \mathit{safe}_i$.

Lemma 6.2.7 $\mathit{future}_i(W_i, 0) \subseteq W_i$.

Proof: Let α be an execution fragment of the VEHICLES automaton of n steps, where $n \in \mathbb{N}$, that: starts in a state in W_i, is only comprised of states in R_i, involves no input actions on port j, and has a limit time equal to zero. Moreover, let p_{init} and p_{final} be the initial and final states of α, respectively. By induction on the length n of the execution fragment α, we show that $p_{final} \in W_i$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of no steps and, therefore, $p_{final} = p_{init}$. Since $p_{init} \in W_i$, it follows that $p_{final} \in W_i$.

The inductive step involves showing that if α is an execution fragment of length $n = k + 1$, for some $k \in \mathbb{N}$, then $p_{final} \in W_i$. Let α' be the part of the execution fragment α comprised of the first k steps. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{final} \in W_i$. Since the final state of α is reached from the final state of α' by a single step, the inductive step involves the consideration of all possible steps leading from p'_{final} to p_{final}.

To complete the induction, we consider all possible discrete actions by cases:

1. the actions `brake`$(i)_j$ and `unbrake`$(i)_j$ are not enabled because α involves no input actions on port j.

2. the actions `brake`$(i')_{j'}$, `unbrake`$(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, affect neither the velocity of any of the vehicles, nor the variables $collided(i, i'')$, for $i'' \in I, i'' \neq i$.

   From the induction hypothesis, it is the case that $p'_{final} \in W_i$. Since the actions `brake`$(i')_{j'}$, `unbrake`$(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, do not affect the variables $collided(i, i'')$, for $i'' \in I, i'' \neq i$, it follows that $p_{final} \in G_i$.

   Moreover, the actions `brake`$(i')_{j'}$ and `unbrake`$(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, do not affect the velocity of any of the vehicles, *i.e.*, $p_{final}.\dot{x}_{i''} = p'_{final}.\dot{x}_{i''}$, for all $i'' \in I$. From Lemma 4.4.3, part 1, it follows that the section of the track owned by each of the vehicles does not grow, *i.e.*, $p_{final}.O_{i''} \subseteq p'_{final}.O_{i''}$, for all $i'' \in I$. Since $p'_{final} \in W_i$, the sections of track owned in state p'_{final} by the vehicle i does not overlap the sections of track owned by any of the vehicles it trails. From above however, $p_{final}.O_{i''} \subseteq p'_{final}.O_{i''}$, for all $i'' \in I$, and, therefore, the same applies for the state p_{final}.

   Finally, since all states in α are, by definition, restricted to the set R_i, it follows that $p_{final} \in W_i$.

3. the `brick-wall`(i') actions, for $i' \in I$, set the velocity of the vehicle i' to zero and affect neither the variables $collided(i, i'')$, for $i'' \in I, i'' \neq i$, nor the velocity of any of the other vehicles, *i.e.*, $p_{final}.\dot{x}_{i'''} = p'_{final}.\dot{x}_{i'''}$, for all $i''' \in I, i''' \neq i'$.

   Without loss of generality, consider a particular `brick-wall`(i') action, for some $i' \in I$.

   From the induction hypothesis, it is the case that $p'_{final} \in W_i$. Since the `brick-wall`(i')

action does not affect the variables *collided*(i, i''), for $i'' \in I, i'' \neq i$, it follows that $p_{final} \in G_i$.

The brick-wall(i') action sets the velocity of the vehicle i' to zero. Therefore, since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_{i'} \leq p'_{final}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_{i'} \subseteq p'_{final}.O_{i'}$. Moreover, since the brick-wall(i') action does not affect the velocity of any of the other vehicles, it is the case that $p_{final}.\dot{x}_{i''} = p'_{final}.\dot{x}_{i''}$, for all $i'' \in I, i'' \neq i'$. Again, from Lemma 4.4.3, part 1, it follows that the section of the track owned by any of the vehicles other than i' does not grow, *i.e.*, $p_{final}.O_{i''} \subseteq p'_{final}.O_{i''}$, for all $i'' \in I, i'' \neq i'$.

Since $p'_{final} \in W_i$, the sections of track owned in state p'_{final} by the vehicle i does not overlap the sections of track owned by any of the vehicles it trails. From above however, $p_{final}.O_{i''} \subseteq p'_{final}.O_{i''}$, for all $i'' \in I$, and, therefore, the same applies for the state p_{final}.

Finally, since all states in α are, by definition, restricted to the set R_i, it follows that $p_{final} \in W_i$.

4. the actions colliding-pair(i', i''), for $i', i'' \in I, i' \neq i''$, and collision-effects$(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i and $p'_{final} \in W_i$.

■

Lemma 6.2.8 $W_i \subseteq$ *safe*$_i$.

Proof: From the definition of *safe* in Section 3.2.1, we must show that any state $p \in W_i$ satisfies: (i) *future*$_i(p, 0) \subseteq G_i$, and (ii) there exists some action π such that for every $p', p'' \in R_i$ satisfying $p' \in$ *future*$_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in$ *very-safe*$_i$.

(i) The first condition follows from Lemma 6.2.7, Lemma 6.2.1, part 1, and the fact that $p \in W_i$.

(ii) For the second condition, consider the state p'' that follows from p' after a brake$(i)_j$ action is executed, *i.e.*, let $\pi =$ brake$(i)_j$. Since the brake$(i)_j$ action does not affect the velocity of the vehicle i, it is the case that $p''.O_i = p'.O_i$. However, from Lemma 6.2.7 and the fact that $p \in W_i$ it follows that $p' \in W_i$. Since (i) $p' \in W_i$, (ii) the execution fragment α is restricted to the set R_i, and (iii) the brake$(i)_j$ action affects neither the variables *collided*(i, i'), for $i' \in I, i' \neq i$, nor the velocity of any of the vehicles (and, therefore, nor the section of the track owned by any of the vehicles), it follows that $p'' \in W_i$. Moreover, since p'' follows from p' after a brake$(i)_j$ action, it is the case that $p'' \in P_{B_{ij}}$. From the above conditions, it follows that $p'' \in V_i$. Finally, Lemma 6.2.5 implies that $p'' \in$ *very-safe*$_i$, as needed. ■

Lemma 6.2.9 *For any* $p \in R_i$*, if* $p \in$ *safe*$_i$ *then* $p \in W_i$.

Proof: We show the contrapositive; that is, for any $p \in R_i$, if $p \notin W_i$ then $p \notin \mathit{safe}_i$. Since $W_i = \{p \in R_i \cap G_i \mid \nexists\, i' \in I, i' \neq i : p.O_i \cap p.O_{i'} \neq \emptyset \ \wedge \ p.x_i < p.x_{i'}\}$ and $p \in R_i$, we consider the condition $p \notin G_i$ and the condition that there exists $i' \in I, i' \neq i$, such that $p.O_i \cap p.O_{i'} \neq \emptyset$ and $p.x_i < p.x_{i'}$.

1. $p \notin G_i$.

   From Lemma 3.2.4, part 1, it is the case that $\mathit{safe}_i \subseteq G_i$. Since $p \notin G_i$, it follows that $p \notin \mathit{safe}_i$.

2. $\exists\, i' \in I, i' \neq i$, such that $p.O_i \cap p.O_{i'} \neq \emptyset$ and $p.x_i < p.x_{i'}$.

   Without loss of generality, let $i' \in I, i' \neq i$, be the vehicle that satisfies the conditions $p.O_i \cap p.O_{i'} \neq \emptyset$ and $p.x_i < p.x_{i'}$. Since $p \in \mathit{VALID}$, it is the case that the vehicles in state p have no positive length extent overlap and, therefore, there is only one vehicle i', for $i' \in I, i' \neq i$, satisfying the conditions $p.O_i \cap p.O_{i'} \neq \emptyset$ and $p.x_i < p.x_{i'}$.

   We must show that $p \notin \mathit{safe}_i$. However, $p \in \mathit{safe}_i$ implies that there exists some input action π on port j such that for every $p', p'' \in R_i$ satisfying $p' \in \mathit{future}_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in \mathit{very\text{-}safe}_i$. Therefore, it suffices to show that for any input action π on port j, there exist $p', p'' \in R_i$ satisfying $p' \in \mathit{future}_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin \mathit{very\text{-}safe}_i$. We consider each input action π on port j separately.

   (a) $\pi = \mathtt{brake}(i)_j$.

   Consider the state $p' \in R_i$ that is reached from the state p through the execution of the action $\mathtt{brick\text{-}wall}(i')$ and satisfies the condition $p'.\ddot{x}_{i'} = 0$; that is, $p' \in R_i$ such that $p'.\dot{x}_{i'} = 0$ and $p'.\ddot{x}_{i'} = 0$.

   Since the actions $\mathtt{brick\text{-}wall}(i')$ and $\mathtt{brake}(i)_j$ affect neither the position, nor the velocity of the vehicle i, it is the case that $p''.x_i = p'.x_i = p.x_i$ and $p''.\dot{x}_i = p'.\dot{x}_i = p.\dot{x}_i$. Therefore, since the section of track owned by the vehicle i depends only on the position and the velocity of the vehicle i, it is the case that $p''.O_i = p'.O_i = p.O_i$. Similarly, since the $\mathtt{brick\text{-}wall}(i')$ action does not affect the position of the vehicle i' but sets its velocity to zero and the $\mathtt{brake}(i)_j$ action affects neither the position, nor the velocity of the vehicle i', it follows that $p''.x_{i'} = p'.x_{i'} = p.x_{i'}$ and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'} = 0$. Therefore, since $p.O_i \cap p.O_{i'} \neq \emptyset$, $p''.O_i = p.O_i$, and $p''.x_{i'} = p.x_{i'}$, it is the case that $p''.x_{i'} \in p''.O_i$.

   Now, consider the evolution of the VEHICLES automaton following the state p'' in which the vehicle i' remains stationary. Since $p''.x_{i'} \in p''.O_i$, it follows that at some state of such an evolution the action $\mathtt{colliding\text{-}pair}(i, i')$ is enabled and, subsequently, executed. The state of the VEHICLES automaton following the execution of the action $\mathtt{colliding\text{-}pair}(i, i')$ would, therefore, not be in G_i. It follows that $p'' \notin \mathit{very\text{-}safe}_i$ which implies that $p \notin \mathit{safe}_i$.

(b) $\pi = \texttt{unbrake}(i)_j$.

Consider the state $p' \in R_i$ that is reached from the state p through the execution of the actions $\texttt{brick-wall}(i')$ and $\texttt{unbrake}(i)_{j'}$, for all $j' \in J, j' \neq j$, and satisfies the condition $p'.\ddot{x}_{i'} = 0$; that is, $p' \in R_i$ such that $p'.\dot{x}_{i'} = 0$, $p'.\ddot{x}_{i'} = 0$, and $p'.\textit{brake-req}(i, j') = \texttt{False}$, for all $j' \in J, j' \neq j$.

Since the actions $\texttt{brick-wall}(i')$ and $\texttt{unbrake}(i)_{j'}$, for all $j' \in J$, affect neither the position, nor the velocity of the vehicle i, it follows that $p''.x_i = p'.x_i = p.x_i$ and $p''.\dot{x}_i = p'.\dot{x}_i = p.\dot{x}_i$. Therefore, since the section of track owned by the vehicle i depends only on the position and the velocity of the vehicle i, it is the case that $p''.O_i = p'.O_i = p.O_i$. Similarly, since the action $\texttt{brick-wall}(i')$ does not affect the position of the vehicle i' but sets its velocity to zero and the actions $\texttt{unbrake}(i)_{j'}$, for all $j' \in J$, affect neither the position, nor the velocity of the vehicle i', it follows that $p''.x_{i'} = p'.x_{i'} = p.x_{i'}$ and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'} = 0$. Therefore, since $p.O_i \cap p.O_{i'} \neq \emptyset$, $p''.O_i = p.O_i$, and $p''.x_{i'} = p.x_{i'}$, it is the case that $p''.x_{i'} \in p''.O_i$.

Now, consider the evolution of the VEHICLES automaton following the state p'' in which the vehicle i moves forward and the vehicle i' remains stationary. Since $p''.x_{i'} \in p''.O_i$, it follows that at some state of such an evolution the action $\texttt{colliding-pair}(i, i')$ is enabled and, subsequently, executed. The state of the VEHICLES automaton following the execution of the action $\texttt{colliding-pair}(i, i')$ would, therefore, not be in G_i. It follows that $p'' \notin \textit{very-safe}_i$ which implies that $p \notin \textit{safe}_i$.

Thus, for any input action π on port j, there exist $p', p'' \in R_i$ satisfying $p' \in \textit{future}_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin \textit{very-safe}_i$. It follows that $p \notin \textit{safe}_i$, as needed.

■

Corollary 6.2.10 $W_i = \textit{safe}_i$.

Proof: Follows directly from Lemmas 6.2.8 and 6.2.9. ■

In the next few lemmas, we show that any state p in the set $T_i(t)$, for any $t \in \mathbb{R}^{\geq 0}$, is in the set $\textit{delay-safe}_i(t)$; that is, any state R_i-reachable from p within an amount of time t through an execution fragment that involves no input actions on port j, is in the set G_i and any state R_i-reachable from the state p in exactly an amount of time t through an execution fragment that involves no input actions on port j, is in the set $\textit{safe}_i$.

Lemma 6.2.11 *Let $p \in T_i(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in \textit{future}_i(p, t)$, where $t \in [0, \tau]$. If $p' \in G_i$ and $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$, then $p' \in T_i(\tau - t)$.*

Proof: We need to show that $p' \in R_i \cap G_i$ and that there does not exist $i' \in I, i' \neq i$ such that $p'.C_i(\tau - t) \cap p'.C_{i'}(\tau - t) \neq \emptyset$ and $p'.x_i < p'.x_{i'}$. We consider the conditions by cases:

1. $p' \in R_i$.

   This is the case because the function $future_i(p, t)$ only considers R_i-reachable states.

2. $p' \in G_i$.

   This is true by assumption.

3. $\nexists\, i' \in I, i' \neq i$, such that $p'.C_i(\tau - t) \cap p'.C_{i'}(\tau - t) \neq \emptyset$ and $p'.x_i < p'.x_{i'}$.

   Because $p \in T_i(\tau)$ we have that for all $i' \in I, i' \neq i$, such that $p.x_i < p.x_{i'}$, it is the case that $p.C_i(\tau) \cap p.C_{i'}(\tau) = \emptyset$; that is, for all $i' \in I, i' \neq i$, such that $p.x_i < p.x_{i'}$, it is the case that $\max(p.C_i(\tau)) < \min(p.C_{i'}(\tau))$. However, by assumption it is the case that $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$. Therefore, since the vehicle velocities are restricted to be non-negative, it follows that for all $i' \in I, i' \neq i$, such that $p'.x_i < p'.x_{i'}$, it is the case that $\max(p'.C_i(\tau - t)) < \min(p'.C_{i'}(\tau - t))$. This is sufficient to guarantee that there does not exist $i' \in I, i' \neq i$, such that $p'.C_i(\tau - t) \cap p'.C_{i'}(\tau - t) \neq \emptyset$ and $p'.x_i < p'.x_{i'}$.

■

Lemma 6.2.12 *For all $p \in T_i(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in future_i(p, t)$, where $t \in [0, \tau]$, it is the case that $p' \in G_i$ and $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$.*

Proof: Let $\tau \in \mathbb{R}^{\geq 0}$ and α be an execution fragment of the VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $T_i(\tau)$, is only comprised of states in R_i, involves no input actions on port j, and has a limit time t that lies in the interval $[0, \tau]$. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_i$ and $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$ and $\alpha.ltime = 0$, *i.e.*, $t = 0$. From Lemma 6.2.1, part 1, and the fact that $p_{init} \in T_i(\tau)$, it follows that $p_{final} \subseteq G_i$. Moreover, since $t = 0$, the fact that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ is trivially true.

The inductive step involves showing that if α is an execution fragment of length $n = k + 1$, for some $k \in \mathbb{N}$, with $\alpha.ltime = t$, where $t \in [0, \tau]$, then $p_{final} \in G_i$ and $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories and let $\alpha'.ltime = t'$, where $t' \in [0, t]$. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{final} \in G_i$ and $p'_{final}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$. Moreover, from Lemma 6.2.11

it follows that $p'_{\mathit{final}} \in T_i(\tau - t')$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, keeping in mind that the limit times of α' and α are equal, *i.e.*, $t' = t$, we consider all possible actions by cases:

1. the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and does not affect the variables $\mathit{collided}(i, i')$, for $i' \in I, i' \neq i$.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_i$. Therefore, since the internal action $\texttt{brick-wall}(i)$ does not affect the variables $\mathit{collided}(i, i')$, for $i' \in I, i' \neq i$, it follows that $p_{\mathit{final}} \in G_i$.

   Moreover, since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 2, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{\mathit{init}}.C_i(\tau)$, as needed.
3. the actions $\texttt{brake}(i')_{j'}$, $\texttt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, affect neither the velocity of the vehicle i, nor the variables $\mathit{collided}(i, i''')$, for $i''' \in I, i''' \neq i$.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_i$. Therefore, since the actions $\texttt{brake}(i')_{j'}$, $\texttt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the variables $\mathit{collided}(i, i''')$, for $i''' \in I, i''' \neq i$, it follows that $p_{\mathit{final}} \in G_i$.

   Moreover, since the input actions $\texttt{brake}(i')_{j'}$, $\texttt{unbrake}(i')_{j'}$, for $i' \in I, j' \in J, j' \neq j$, and the internal actions $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 2, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{\mathit{init}}.C_i(\tau)$, as needed.
4. the actions $\texttt{colliding-pair}(i', i'')$, for $i', i'' \in I, i' \neq i''$, and $\texttt{collision-effects}(i''')$, for $i''' \in I$, are not enabled because α is only comprised of states in R_i and $p'_{\mathit{final}} \in T_i(\tau - t')$.

In the case of a trajectory, Lemma 4.4.4, part 2, implies that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$.

Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$. Moreover, since $p'_{final} \in G_i$ and the variables *collided*(i, i'), for all $i' \in I, i' \neq i$, remain constant throughout the trajectory, it follows that $p_{final} \in G_i$, as needed. ■

Lemma 6.2.13 *For $\tau \in \mathbb{R}^{\geq 0}$ and $t \in [0, \tau]$, it is the case that $future_i(T_i(\tau), t) \subseteq T_i(\tau - t)$.*

Proof: Follows directly from Lemmas 6.2.11 and 6.2.12. ■

Lemma 6.2.14 *For all $t \in \mathbb{R}^{\geq 0}$, it is the case that $T_i(t) \subseteq$ delay-safe$_i(t)$.*

Proof: We need to show that $future_i(T_i(t), [0, t]) \subseteq G_i$ and $future_i(T_i(t), t) \subseteq safe_i$. The first condition follows directly from Lemma 6.2.13 and Lemma 6.2.1, part 1. For the second condition, from Lemma 6.2.13 and Lemma 6.2.1, part 3, it is the case that $future_i(T_i(t), t) \subseteq W_i$. Therefore, Lemma 6.2.8, implies that $future_i(T_i(t), t) \subseteq safe_i$, as needed. ■

In the following lemma, we show that the CL-PROT-SOLO$_i$ protector implements the protector $Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$. Since the protector automata CL-PROT-SOLO$_i$ and Abs_j involve the composition of the same sensor automaton with distinct discrete controller automata, it suffices to show that the discrete controller automaton of the protector CL-PROT-SOLO$_i$ implements the $DC(\text{VEHICLES}, S_i, R_i, G_i, j, d)$ automaton.

Lemma 6.2.15 CL-PROT-SOLO$_i \leq Abs(\text{VEHICLES}, S_i, R_i, G_i, j, d)$.

Proof: Both the CL-PROT-SOLO$_i$ and the Abs_j protectors involve the composition of the same sensor automaton with distinct discrete controller automata. From Theorem 2.7.4, it suffices to show that the discrete controller automaton of CL-PROT-SOLO$_i$ implements DC_j. This is shown by a simulation from the discrete controller automaton of CL-PROT-SOLO$_i$ to DC_j.

As in the overspeed case, the mapping between the states of the discrete controller automaton of CL-PROT-SOLO$_i$ and DC_j is almost the identity. In the discrete controller automaton of CL-PROT-SOLO$_i$, the variable $send_j$ is equal to either one of the labels `brake` and `unbrake`, or the value *null*. In the abstract discrete controller automaton, these valuations simply map to either the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, or the value *null*, respectively.

The start states for the discrete controller automaton of CL-PROT-SOLO$_i$ and DC_j are the states in which $send_j = null$. These are mapped to each other according to the mapping discussed above.

Furthermore, since the trajectories in both discrete controller automata are identical, we need only consider their discrete transitions. We analyze the actions of the implementation by cases, letting p denote any complete state of the VEHICLES automaton that corresponds to y, *i.e.*, $p \in VALID$ and $p \lceil Y_{\text{VEHICLES}} = y$.

1. The $\texttt{snapshot}(y)_j$ action of the implementation sets $send_j$ to $\texttt{brake}$, or $\texttt{unbrake}$. In order to show that the behavior of the implementation is allowed by the specification, we must show that the input action $\texttt{snapshot}(y)_j$ of the implementation sets the value of the $send_j$ variable in such a way that the subsequently enabled action π of the implementation (i) guarantees that for all $p', p'' \in R_i$ such that $p' \in \mathit{future}_i(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in \mathit{delay\text{-}safe}_i(d)$, if $p \in \mathit{safe}_i$, and (ii) is an arbitrary output action of the implementation, otherwise.

   First, consider the case in which $p \in \mathit{safe}_i$. Since Corollary 6.2.10 implies that $p \in W_i$, the discrete controller automaton of CL-PROT-SOLO$_i$ sets the variable $send_j$ according to whether the state p is in $T_i(d)$, or not.

   On one hand, if $p \notin T_i(d)$ then the discrete controller automaton of CL-PROT-SOLO$_i$ sets the variable $send_j$ to $\texttt{brake}$ and the $\texttt{brake}(i)_j$ action is enabled. However, since $p \in W_i$, Lemma 6.2.7 implies that $p' \in W_i$. Moreover, since the $\texttt{brake}(i)_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_i$, $p'' \in G_i$, and $p''.\dot{x}_i = p'.\dot{x}_i$. Therefore, Lemma 4.4.3, part 1, implies that $p''.O_i \subseteq p'.O_i$. From the above conditions and the non-negative constraint on the vehicle velocities, it follows that $p'' \in W_i$. Moreover, since the $\texttt{brake}(i)_j$ action sets the $\mathit{brake\text{-}req}(i, j)$ variable to $\texttt{True}$, it follows that $p'' \in V_i$. Finally, from Lemma 6.2.6 it follows that $p'' \in \mathit{delay\text{-}safe}_i(d)$, as needed.

   On the other hand, if $p \in T_i(d)$ then the discrete controller automaton of the protector CL-PROT-SOLO$_i$ sets the variable $send_j$ to $\texttt{unbrake}$ and the $\texttt{unbrake}(i)_j$ action is enabled. However, since $p \in T_i(d)$, Lemma 6.2.13 implies that $p' \in T_i(d)$. Since the $\texttt{unbrake}(i)_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_i$, $p'' \in G_i$, and $p''.\dot{x}_i = p'.\dot{x}_i$. Therefore, Lemma 4.4.3, part 2, implies that $p''.C_i(d) \subseteq p'.C_i(d)$. From the above conditions and the non-negative constraint on the vehicle velocities, it follows that $p'' \in T_i(d)$. Finally, from Lemma 6.2.14 it follows that $p'' \in \mathit{delay\text{-}safe}_i(d)$, as needed.

   Next, consider the case in which $p \notin \mathit{safe}_i$. In this case, the $\texttt{snapshot}(y)_j$ action of the discrete controller automaton of CL-PROT-SOLO$_i$ sets the variable $send_j$ to either $\texttt{brake}$ or $\texttt{unbrake}$ and, subsequently, enables either the action $\texttt{brake}(i)_j$, or the action $\texttt{unbrake}(i)_j$. However, when $p \notin \mathit{safe}_i$, the DC_j automaton sets the variable $send_j$ arbitrarily and, subsequently, enables an arbitrary output action. Therefore, the behavior of the discrete controller automaton of CL-PROT-SOLO$_i$ is allowed by that of the DC_j automaton.

   Therefore, the effects of the $\texttt{snapshot}(y)_j$ action of the implementation are allowed by its specification.

2. The $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ actions have identical effects in both discrete controller automata. When the $send_j$ variable matches the label $\texttt{brake}$ or $\texttt{unbrake}$ or the action

$\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, respectively, the respective action is performed and the $send_j$ variable is set to *null* in both discrete controller automata.

3. The environment action in both discrete controller automata is stuttering. It follows that the mapping between the states of the discrete controller automaton of CL-PROT-SOLO$_i$ and the DC_j automaton prior to and succeeding the execution of the environment action remains the same.

■

Corollary 6.2.16 *The protector* CL-PROT-SOLO$_i$ *guarantees* G_i *in the* VEHICLES *automaton starting from* S_i *given* R_i*.*

Proof: Follows directly from Lemma 6.2.15 and Theorem 3.2.9. ■

6.3 Protection System CL-PROT

We now define the collision protector CL-PROT. While considering the CL-PROT automaton, we restrict the states of the VEHICLES automaton to the set $P_{not\text{-}overspeed}$ as defined in Section 4.2, *i.e.*, $R_{\text{CL-PROT}} = P_{not\text{-}overspeed}$. Let $G_{\text{CL-PROT}}$ and $S_{\text{CL-PROT}}$ be the intersection of G_i and S_i, for all $i \in I$, respectively, and CL-PROT be the composition of the protectors CL-PROT-SOLO$_i$, for all $i \in I$. The protector CL-PROT guarantees that the VEHICLES automaton remains within $G_{\text{CL-PROT}}$ starting from $S_{\text{CL-PROT}}$ given $R_{\text{CL-PROT}}$. For reference, the formal definitions of the CL-PROT automaton and the sets $G_{\text{CL-PROT}}$, $S_{\text{CL-PROT}}$, and $R_{\text{CL-PROT}}$ are shown in Table 6.2.

Lemma 6.3.1 *The protector* CL-PROT *guarantees* $G_{\text{CL-PROT}}$ *in the* VEHICLES *automaton starting from* $S_{\text{CL-PROT}}$ *given* $R_{\text{CL-PROT}}$*.*

In the following proof, we show that all the states of an execution of $PP \times$ CL-PROT starting from $S_{\text{CL-PROT}}$ given $R_{\text{CL-PROT}}$ are in $G_{\text{CL-PROT}}$. This is done by applying Theorem 3.1.8 and showing that the second condition of the theorem does not hold.

Proof: Let α be any execution of the system $PP \times$ CL-PROT starting from a state in $S_{\text{CL-PROT}}$ and in which all states are in $R_{\text{CL-PROT}}$.

From Theorem 3.1.8, one of the following holds:

1. Every state in α is in $G_{\text{CL-PROT}} = \bigcap_{i \in I} G_i$.
2. α can be written as $\alpha_1 \frown \alpha_2$, where

Table 6.2 Formal definitions of CL-PROT, $G_{\text{CL-PROT}}$, $S_{\text{CL-PROT}}$, and $R_{\text{CL-PROT}}$.

$$\text{CL-PROT} \equiv \prod_{i \in I} \text{CL-PROT-SOLO}_i$$

$$G_{\text{CL-PROT}} \equiv \bigcap_{i \in I} G_i$$

$$S_{\text{CL-PROT}} \equiv \bigcap_{i \in I} S_i$$

$$R_{\text{CL-PROT}} \equiv P_{not\text{-}overspeed}$$

(a) All state occurrences in α_1 except possibly the last state occurrence are in the set $G_{\text{CL-PROT}} = \bigcap_{i \in I} G_i$.

(b) If the last state occurrence in α_1 is in $\overline{G_i}$, for some $i \in I$, then there exists $i' \in I, i' \neq i$, such that the last state occurrence in α_1 is in $\overline{G_{i'}}$.

(c) All state occurrences in α_2 except possibly the first state occurrence are in the set $\bigcap_{i \in N} \mathit{past}(\overline{G_i}, \alpha)$, for some $N \subseteq I$, where $|N| \geq 2$.

We proceed by showing that it is not possible to decompose α as $\alpha_1 \frown \alpha_2$ while satisfying the three aforementioned conditions.

The violation of $\bigcap_{i \in I} G_i$ can only occur through the violation of at least one of the conditions G_i, where $i \in I$. Moreover, each of these conditions are violated only through the execution of a `colliding-pair` action. Without loss of generality, suppose that the first condition that is violated in α is the condition G_i, for some $i \in I$, and that such a violation has resulted through a `colliding-pair`(i, i') action, for some $i' \in I, i' \neq i$. Let p and p' be the states of the VEHICLES automaton prior to and succeeding this `colliding-pair`(i, i') action, *i.e.*, $p, p' \in R_{\text{CL-PROT}}$ such that $p \xrightarrow{\pi} p'$, where $\pi =$ `colliding-pair`(i, i'). Since the `colliding-pair`(i, i') action only sets the *collided*(i, i') variable to `True`, it follows that $p' \in \overline{G_i} \bigcap \left(\bigcap_{i'' \in I, i'' \neq i} G_{i''} \right)$. Now, we attempt to decompose α as $\alpha_1 \frown \alpha_2$:

1. Suppose we split α at any state preceding the state p. Then the state p is in α_2. Since p' is the first state in which one of the conditions $G_{i''}$, for $i'' \in I$, is violated, it is the case that $p \in \bigcap_{i'' \in I} G_{i''}$ and there does not exist $N \subseteq I$ such that $|N| \geq 2$ and $p \in \bigcap_{i \in N} \mathit{past}(\overline{G_i}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

2. Suppose we split α at the state p. Then the state p' is in α_2. Since p' is the first state in which one of the conditions $G_{i''}$, for $i'' \in I$, is violated and since the state p' is in $\overline{G_i} \bigcap \left(\bigcap_{i'' \in I, i'' \neq i} G_{i''} \right)$, it follows that there does not exist $N \subseteq I$ such that $|N| \geq 2$ and $p' \in \bigcap_{i \in N} past(\overline{G_i}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

3. Suppose we split α at the state p'. Then p' is the last state of α_1 and the first state of α_2. However, $p' \in \overline{G_i} \bigcap \left(\bigcap_{i'' \in I, i'' \neq i} G_{i''} \right)$. Therefore, the second condition is violated and this decomposition of α is not valid.

4. Suppose we split α at any state succeeding p'. Then the state p' is in α_1. Since $p' \in \overline{G_i} \bigcap \left(\bigcap_{i'' \in I, i'' \neq i} G_{i''} \right)$, it is the case that $p' \notin \bigcap_{i'' \in I} G_{i''}$. Therefore, the first condition is violated and this decomposition of α is not valid.

Therefore, the execution α cannot be decomposed into any such α_1 and α_2. It follows that the first clause of Theorem 3.1.8 must hold; that is, every state in α is in $G_{\text{CL-PROT}}$. This implies that the protector CL-PROT guarantees $G_{\text{CL-PROT}}$ in the VEHICLES automaton starting from $S_{\text{CL-PROT}}$ given $R_{\text{CL-PROT}}$. ■

Chapter 7

Example 3: Collision Avoidance on Merging Tracks

This chapter treats collision avoidance among vehicles that are traveling on a track involving a binary merge. We first augment the model of the PRT 2000™ to involve a track topology consisting of two merging tracks — the new model is referred to as the MERGE-VEHICLES automaton. Then we define the protector MERGE-PROT that guarantees that none of the vehicles of the MERGE-VEHICLES automaton collide, assuming that they are all abiding by the speed limit. 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 $\{i, i'\}$ of vehicles of the MERGE-VEHICLES automaton, for $i, i' \in I, i \neq i'$. Each of these MERGE-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, is an implementation of a particular instantiation of the abstract protector automaton of Section 3.2 and guarantees that the vehicles i and i' do not collide into each other.

7.1 Augmented Physical Plant: MERGE-VEHICLES

In this section we augment the model for the system of n vehicles to involve a merge of two sections of track. We replace 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. We replace the `brake` and `unbrake` input actions of the VEHICLES automaton with `protect` input actions which allow single protectors to instruct multiple 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 set of track locations in the VEHICLES automaton was a line. In the case of a binary merge, the set of locations is a Y-shaped track — two incoming branches and one outgoing branch. We define the set of locations L as follows:

$$L = (\{\texttt{left}, \texttt{right}\} \times \mathbb{R}^{<0}) \cup (\{\texttt{out}\} \times \mathbb{R}^{\geq 0})$$

Each location l, for $l \in L$, is comprised of two components; the first component represents the branch of the track on which the vehicle is traveling and the second component represents the position of the vehicle with respect to the merge point. The locations on the top branch of the merge have the label `left` and a *negative* real number as their respective components. Similarly, the locations on the bottom branch of the merge have the label `right` and a *negative* real number as their respective components. The locations on the merged section of the track are specified by the label `out` and a *non-negative* real number. The point $\langle \texttt{out}, 0\rangle$ is the first point on the merged section of the track that no two vehicles can occupy simultaneously. For notational brevity, we use $l.b$ and $l.x$ to denote the branch and the position components of the location l, respectively.

We define a partial order on L, as follows. If $\langle b_1, x_1\rangle$ and $\langle b_2, x_2\rangle$ are locations in L then $\langle b_1, x_1\rangle \leq \langle b_2, x_2\rangle$ if and only if $x_1 \leq x_2$ and either $b_1 = b_2$, or $b_2 = \texttt{out}$. In other words, two locations are *incomparable* if one specifies a location on the `left` branch and the other specifies a location on the `right` branch; otherwise, they are *comparable* and their order is given by the ordering on their real component.

A closed interval in L is specified with an ordered pair of locations that are comparable, *e.g.*, $[\langle \texttt{left}, -1\rangle, \langle \texttt{out}, 2.5\rangle]$, and contains all locations between them. Addition with non-negative scalars on L is defined as follows: if $\langle b, x\rangle$ is a location in L and $y \in \mathbb{R}^{\geq 0}$, then $\langle b, x\rangle + y$ is equal to $\langle b, x + y\rangle$ if $x + y$ is negative, and $\langle \texttt{out}, x + y\rangle$ otherwise. It is important to note that for all $y \in \mathbb{R}^{\geq 0}$, $\langle b, x\rangle + y$ exists and $\langle b, x\rangle \leq (\langle b, x\rangle + y)$.

The automaton MERGE-VEHICLES of Figure 7.1 models a physical system of n vehicles traveling on a track involving a Y-shaped merge. The MERGE-VEHICLES automaton is the result of augmenting the VEHICLES automaton of Chapter 4 to allow for the Y-shaped track topology.

In the new model, each of the position components x_i of the state of the VEHICLES automaton is replaced with the corresponding location component l_i. This entails simply replacing the occurrences of x_i with $l_i.x$. The derived variables *stop-dist*$_i$, *max-range*$_i(t)$, and *max-vel*$_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined for the VEHICLES automaton in Section 4.3, carry over unchanged to the MERGE-VEHICLES automaton. The derived variables E_i, O_i, and $C_i(t)$,

Figure 7.1 The MERGE-VEHICLES automaton.

Actions:

Input:
- e, the environment action (stuttering)
- $\mathtt{protect}(C)_j$, for all $C \in \mathcal{P}(I), j \in J$

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

Variables

Internal:
- $\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$
- $brake(i) \in \mathtt{Bool}$, for all $i \in I$, initially $\mathtt{False}$
- $brake\text{-}req(i, j) \in \mathtt{Bool}$, for all $i \in I, j \in J$, initially $\mathtt{False}$

Output:
- $l_i \in L$, for all $i \in I$, initially $l_i \in L$
- $\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$
- $collided(i, i') \in \mathtt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\mathtt{False}$

subject to $VALID$

Discrete Transitions:

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

$\mathtt{colliding\text{-}pair}(i, i')$
  Pre: $\neg collided(i, i')$
    $\wedge (E_i \cap E_{i'} \neq \emptyset)$
    $\wedge (l_i < \min(E_i \cap E_{i'}))$
  Eff: $collided(i, i') := \mathtt{True}$
    if $(l_i.b \neq l_{i'}.b)$
      $\wedge (l_i.b \neq \mathtt{out}) \wedge (l_{i'}.b \neq \mathtt{out})$
    then
      $collided(i', i) := \mathtt{True}$

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

$\mathtt{brick\text{-}wall}(i)$
  Pre: $\mathtt{True}$
  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 $t \in T_I$
    $w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i\ ds$
    $w(t).l_i.x = w(0).l_i.x + \int_0^t w(s).\dot{x}_i\ ds$
    if $\neg w.collided(i, i')$
      $\wedge (w(t).E_i \cap w(t).E_{i'} \neq \emptyset)$
      $\wedge (w(t).l_i < \min(w(t).E_i \cap w(t).E_{i'}))$
    then
      $t = w.ltime$
subject to $VALID$

for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined for the VEHICLES automaton in Sections 4.1 and 4.3, extend to the MERGE-VEHICLES automaton by replacing the position variables with their location counterparts.

In the VEHICLES automaton, a collision between two vehicles is recorded solely by the trailing vehicle — as if it is the only vehicle blamed for the collision. The rationale behind this approach is that the trailing vehicle is the only vehicle that is capable of preventing a collision through braking; that is, the trailing vehicle is liable for the collision. This rationale carries over to the MERGE-VEHICLES automaton with the exception that in the MERGE-VEHICLES automaton it is possible for two vehicles to collide sideways while merging. In such situations, it is not clear which vehicle is liable for the collision and, therefore, the collision is recorded by both vehicles involved in the collision. This is done by augmenting the effects of the $\texttt{colliding-pair}(i, i')$ actions, for $i, i' \in I, i \neq i'$, so that a $\texttt{colliding-pair}(i, i')$ action sets both the variables *collided*(i, i') and *collided*(i', i) to $\texttt{True}$ when the vehicles i and i' are colliding sideways while merging. If indeed the vehicles i and i' are colliding sideways while merging, although both of the actions $\texttt{colliding-pair}(i, i')$ and $\texttt{colliding-pair}(i', i)$ are enabled, only one of them is actually executed and neither of them is enabled thereafter. The interpretation of the *collided*(i, i') variables, for $i, i' \in I, i \neq i'$, still remains the same; that is, each of the variables *collided*(i, i'), for $i, i' \in I, i \neq i'$, denotes whether the vehicle i has collided into the vehicle i'. However, if *collided*$(i, i') = \texttt{True}$ and *collided*$(i', i) = \texttt{False}$, then it follows that the vehicle i has collided into the vehicle i' from behind, where as, if *collided*$(i, i') = \texttt{True}$ and *collided*$(i', i) = \texttt{True}$, then it follows that the vehicles i and i' have collided sideways while merging.

The $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$ actions of the VEHICLES automaton, for $i \in I$ and $j \in J$, are replaced by the $\texttt{protect}(C)_j$ actions, for $C \in \mathcal{P}(I)$ and $j \in J$. These actions enable a protector j to instruct each of the vehicles in the set of vehicles C to apply its "emergency" brakes. If a vehicle i is a member of C then it is requested to brake by the protector j, emulating a $\texttt{brake}(i)_j$ action of the VEHICLES automaton; otherwise, any previous request of the protector j to brake the vehicle i is revoked, emulating an $\texttt{unbrake}(i)_j$ action of the VEHICLES automaton.

As in the case of the VEHICLES automaton, the set of input actions of the MERGE-VEHICLES automaton includes the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(I)$ and $j \in J$; that is, the MERGE-VEHICLES automaton allows each protector j, for $j \in J$, to brake any subset of the vehicles. However, it is often the case that a protector j, for some $j \in J$, need not schedule but a subset of the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(I)$. In such cases, the protector j is specified as having only the output actions that it is capable of scheduling and the remaining input actions of the MERGE-VEHICLES automaton on port j are ignored.

The remaining state variables and discrete actions of the VEHICLES automaton as well as the notational shorthand *collided*$(i, *)$, *collided*$(*, i)$, and *collided*$(*, i, *)$, for all $i \in I$, defined for the VEHICLES automaton in Section 4.1, carry over to the MERGE-VEHICLES automaton unchanged.

In the case of the trajectories of the MERGE-VEHICLES automaton, it is important to note that due to the nature of the set of locations L, as the vehicles travel past the merge point, the branch component of their location variables changes from either `left`, or `right` to `out`.

Finally, we redefine the set *VALID* to account for the new track topology.

VALID $\subseteq$ *states*(MERGE-VEHICLES), defined as the set of states of the MERGE-VEHICLES automaton that satisfy the following conditions: 1. $\nexists\ i, i' \in I, i \neq i'$, such that the set $E_i \cap E_{i'}$ is a positive length closed interval of L. 2. $\dot{x}_i \geq 0$, for all $i \in I$. 3. If $\neg collided(*, i, *)$ then $\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$, for all $i \in I$. 4. If $\neg collided(*, i, *) \wedge brake(i)$ then if $\dot{x}_i = 0$ then $\ddot{x}_i = 0$ else $\ddot{x}_i = \ddot{c}_{brake}$, for all $i \in I$.

The MERGE-VEHICLES automaton complies with the assumptions made about the *PP* automaton in Section 3.2.1. The MERGE-VEHICLES automaton has neither input variables, nor output actions, on any of its ports (Axioms 3.2.1 and 3.2.2, respectively). Moreover, each of the actions $\mathtt{protect}(C_j)_j$, for $j \in J$ and $C_j = \{i \mid brake\text{-}req(i, j) = \mathtt{True}\}$, is a no-op input action on port j for any $R \subseteq$ *VALID*. Therefore, the set of no-op input actions on each port $j \in J$ and any $R \subseteq$ *VALID* is non-empty (Axiom 3.2.3).

Henceforth, we assume that the sets *disjoint-extents*(i, i'), *disjoint-owned-tracks*(i, i'), and *disjoint-claimed-tracks*(i, i', t), for $i, i' \in I, i \neq i'$ and $t \in \mathbb{R}^{\geq 0}$, defined for the VEHICLES automaton in Section 4.3, have been extended to the MERGE-VEHICLES automaton to incorporate the redefinitions of the derived variables used in their definitions. Moreover, we assume that the Lemmas 4.4.1, 4.4.2, 4.4.3, 4.4.4, and 4.4.5 extend to the MERGE-VEHICLES automaton in the obvious way.

7.2 Auxiliary Sets for the MERGE-VEHICLES Automaton

This section presents several auxiliary sets for the MERGE-VEHICLES automaton that are comprised of states that satisfy particular properties. While their formal definitions appear in Table 7.1, their informal descriptions follow.

comparable(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* comprised of the states in which the locations of the vehicles i and i' are comparable.

incomparable(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* comprised of the states in which the locations of the vehicles i and i' are not comparable.

yield-comparable(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *comparable*(i, i') comprised of the states in which, in the case of a claim overlap between the vehicles i and i', the vehicle i must yield to the vehicle i'. When the locations of the vehicles i and i' are comparable, the vehicle i must yield to the vehicle i' if the location of the vehicle i is strictly less than the location of the vehicle i'.

yield-incomparable(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *incomparable*(i, i') comprised of the states in which, in the case of a claim overlap between the vehicles i and i', the vehicle i must yield to the vehicle i'. When the locations of the vehicles i and i' are not comparable, the vehicle i must yield to the vehicle i' if either *only* the vehicle i' owns the merge point, or the vehicle i is traveling on the `left` branch and neither or both vehicles own the merge point.

yield(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* comprised of the states in which, in the case of a claim overlap between the vehicles i and i', the vehicle i must yield to the vehicle i' in order to prevent a potential collision between the vehicles i and i'.

Since the above definitions only depend on the output variables of the MERGE-VEHICLES automaton, we often use the above sets to classify states of the output state set $Y_{\text{MERGE-VEHICLES}}$.

The following lemma describes some properties of the sets defined above.

Lemma 7.2.1 *For all $i, i' \in I, i \neq i'$, the following hold:*

1. *VALID* $=$ *comparable*$(i, i') \cup$ *incomparable*(i, i').
2. *comparable*$(i, i') =$ *yield-comparable*$(i, i') \cup$ *yield-comparable*(i', i).
3. *yield-comparable*$(i, i') \cap$ *yield-comparable*$(i', i) = \emptyset$.
4. *incomparable*$(i, i') =$ *yield-incomparable*$(i, i') \cup$ *yield-incomparable*(i', i).
5. *yield-incomparable*$(i, i') \cap$ *yield-incomparable*$(i', i) = \emptyset$.

Proof: We prove each of the conditions separately:

1. This follows directly from the definition of *comparable*(i, i') and *incomparable*(i, i'), for $i, i' \in I, i \neq i'$.
2. For all $i, i' \in I, i \neq i'$, the sets *yield-comparable*(i, i') and *yield-comparable*(i', i) are both subsets of the set *comparable*(i, i'). Therefore, it suffices to show that any state p in the set *comparable*(i, i'), for some $i, i' \in I, i \neq i'$, is either in the set *yield-comparable*(i, i'), or in the set *yield-comparable*(i', i).

Table 7.1 Auxiliary sets for the MERGE-VEHICLES automaton.

$comparable(i,i') \subseteq VALID$, for $i, i' \in I, i \neq i'$, defined by

$$comparable(i,i') = \{p \in VALID \mid (p.l_i.b = p.l_{i'}.b) \vee (p.l_i.b = \texttt{out}) \vee (p.l_{i'}.b = \texttt{out})\}$$

$incomparable(i,i') \subseteq VALID$, for $i, i' \in I, i \neq i'$, defined by

$$incomparable(i,i') = VALID - comparable(i,i')$$

$yield\text{-}comparable(i,i') \subseteq comparable(i,i')$, for $i, i' \in I, i \neq i'$, defined by

$$yield\text{-}comparable(i,i') = \{p \in comparable(i,i') \mid p.l_i < p.l_{i'}\}$$

$yield\text{-}incomparable(i,i') \subseteq incomparable(i,i')$, for $i, i' \in I, i \neq i'$, defined by

$$\begin{aligned} yield\text{-}incomparable(i,i') = \{p \in incomparable(i,i') \mid & ((\langle \texttt{out}, 0\rangle \notin p.O_i \wedge \langle \texttt{out}, 0\rangle \in p.O_{i'}) \\ & \vee ((\langle \texttt{out}, 0\rangle \in p.O_i \wedge \langle \texttt{out}, 0\rangle \in p.O_{i'} \\ & \quad \wedge p.l_i.b = \texttt{left}) \\ & \vee ((\langle \texttt{out}, 0\rangle \notin p.O_i \wedge \langle \texttt{out}, 0\rangle \notin p.O_{i'} \\ & \quad \wedge p.l_i.b = \texttt{left})\} \end{aligned}$$

$yield(i,i') \subseteq VALID$, for $i, i' \in I, i \neq i'$, defined by

$$yield(i,i') = yield\text{-}comparable(i,i') \cup yield\text{-}incomparable(i,i')$$

Let the state p be any state in $comparable(i,i')$, for some $i, i' \in I, i \neq i'$. Since $comparable(i,i') \subseteq VALID$, it is the case that $p \in VALID$. Therefore, the sections of the track occupied by the vehicles i and i' do not have a positive length closed interval overlap. It follows that it is not possible for their locations to coincide; that is, for any $p \in comparable(i,i')$, it is the case that $p.l_i \neq p.l_{i'}$. Therefore, regarding the ordering of the locations of the vehicles i and i', there are only two viable cases:

(a) $p.l_i < p.l_{i'}$. In this case, $p \in yield\text{-}comparable(i,i')$.

(b) $p.l_{i'} < p.l_i$. In this case, $p \in yield\text{-}comparable(i',i)$.

3. If $p \in yield\text{-}comparable(i,i')$ then it is the case that $p.l_i < p.l_{i'}$. It follows that $p \notin yield\text{-}comparable(i',i)$. Similarly, if $p \in yield\text{-}comparable(i',i)$ then it is the case that $p.l_{i'} < p.l_i$. It follows that $p \notin yield\text{-}comparable(i,i')$. This suffices.

4. For all $i, i' \in I, i \neq i'$, the sets $yield\text{-}incomparable(i,i')$ and $yield\text{-}incomparable(i',i)$ are both subsets of the set $incomparable(i,i')$. Therefore, it suffices to show that any

state p in the set *incomparable*(i, i'), for some $i, i' \in I, i \neq i'$, is either in the set *yield-incomparable*(i, i'), or in the set *yield-incomparable*(i', i).

Let the state p be any state in *incomparable*(i, i'), for some $i, i' \in I, i \neq i'$, and without loss of generality let the vehicle i be the vehicle traveling on the `left` incoming edge. Regarding the ownership of the merge point by each of the vehicles, there are four cases:

(a) $\langle \texttt{out}, 0\rangle \in p.O_i \wedge \langle \texttt{out}, 0\rangle \in p.O_{i'}$. In this case, $p \in$ *yield-incomparable*(i, i') and $p \notin$ *yield-incomparable*(i', i).

(b) $\langle \texttt{out}, 0\rangle \notin p.O_i \wedge \langle \texttt{out}, 0\rangle \notin p.O_{i'}$. Similarly to above, $p \in$ *yield-incomparable*(i, i') and $p \notin$ *yield-incomparable*(i', i).

(c) $\langle \texttt{out}, 0\rangle \notin p.O_i \wedge \langle \texttt{out}, 0\rangle \in p.O_{i'}$. In this case, $p \in$ *yield-incomparable*(i, i') and $p \notin$ *yield-incomparable*(i', i).

(d) $\langle \texttt{out}, 0\rangle \in p.O_i \wedge \langle \texttt{out}, 0\rangle \notin p.O_{i'}$. In this case, $p \notin$ *yield-incomparable*(i, i') and $p \in$ *yield-incomparable*(i', i).

5. This condition follows from the analysis in the proof of condition 4.

■

7.3 Protection System MERGE-PROT-PAIR$_{\{i,i'\}}$

Each MERGE-PROT-PAIR$_{\{i,i'\}}$ automaton, for $i, i' \in I, i \neq i'$, is a vehicle-pair collision protector and 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. Each of the MERGE-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, is an implementation of the abstract protector of Section 3.2 specialized to particular definitions of the parameters PP, S, R, G, j, and d.

The physical plant automaton, PP, is defined to be the MERGE-VEHICLES automaton of Figure 7.1. The port j and the sampling period d are defined to be the port and sampling period with which the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ communicates with the MERGE-VEHICLES automaton. They are assumed arbitrary and are fixed for the rest of the chapter. The set of "good" states G is defined to be the set of states in which the vehicles i and i' have not collided into each other, *i.e.*, $G = \mathit{VALID} - P_{collided(i,i')} - P_{collided(i',i)}$. In this chapter, we use the notation $G_{\{i,i'\}}$ to denote the definition of G that is specific to the MERGE-PROT-PAIR$_{\{i,i'\}}$ protector. The set R is defined to be the set $R = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. This definition restricts the states of the MERGE-VEHICLES automaton to states in which all the vehicles are abiding by the speed limit and in which the vehicles of all other vehicle pairs $\{i'', i'''\}$, for $i'', i''' \in I, i'' \neq$

$i''', \{i'', i'''\} \neq \{i, i'\}$, have not collided into each other. The set S is defined to be the set *safe* as defined in Section 3.2.1; that is, the set of states of the *PP* automaton for which a single input action of *PP* on port j can guarantee that, provided no new input actions on port j are allowed, all subsequently R-reachable states will be in G. Once again, the definition of the set *safe* is specialized to the above definitions of the automaton *PP*, the sets R and G, and the port j. In this chapter, we use the notation $R_{\{i,i'\}}$ and $S_{\{i,i'\}}$ to refer to the above definitions of the sets R and S.

The MERGE-PROT-PAIR$_{\{i,i'\}}$ protector automaton is an implementation of the abstract protector automaton $Abs($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. More precisely, as is the case for the abstract protector Abs_j, we define the MERGE-PROT-PAIR$_{\{i,i'\}}$ automaton to be the composition of a sensor and a discrete controller automaton. These automata are implementations of their abstract equivalents of Figures 3.2 and 3.3 specialized, however, to the above definitions of the parameters *PP*, S, R, G, j, and d. The sensor automaton is precisely the specialization of the sensor automaton of Figure 3.2 to the above definitions of the parameters *PP*, *etc.* The discrete controller automaton is defined in Figure 7.2.

The braking strategy of the MERGE-PROT-PAIR$_{\{i,i'\}}$ protector is as follows. The protector is allowed to brake the vehicles i and i' only if the sections of the track they claim in time d 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; 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. However, if both of the vehicles were instructed to brake, it would be possible to reach a *bottleneck* state — a state in which both of the incoming vehicles involved in the claim overlap are instructed to brake thereafter and, subsequently, none of the trailing incoming vehicles would be capable of proceeding.

The braking strategy considers the case in which both the vehicles i and i' own the merge point. Although this situation is a valid state of the MERGE-VEHICLES automaton, in the following section it is shown that such states are excluded from the reachable state set of the composition of the MERGE-VEHICLES automaton and all the MERGE-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$. It is also important to note that, according to the braking strategy and provided that the sections of track owned by the vehicles i and i' are disjoint, if the locations of the vehicles i and i' are not comparable, then the section of the track owned by the vehicle to be braked is entirely upstream of the merge point.

Figure 7.2 Discrete controller automaton for the protector MERGE-PROT-PAIR$_{\{i,i'\}}$.

Actions:	Input:	e, the environment action (stuttering)
		$\mathtt{snapshot}(y)_j$, for each valuation y of $Y_{\text{MERGE-VEHICLES}}$
	Output:	$\mathtt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$
Variables:	Internal:	$send_j \in \mathcal{P}(\{i, i'\}) \cup null$, initially $null$

Discrete Transitions:

$\mathtt{snapshot}(y)_j$
Eff: if $y \notin \mathit{disjoint\text{-}claimed\text{-}tracks}(i, i', d)$ then
 if $y \in \mathit{yield}(i, i')$ then
  $send_j := \{i\}$
 else
  $send_j := \{i'\}$
else
 $send_j := \emptyset$

$\mathtt{protect}(C)_j$
Pre: $send_j = C$
Eff: $send_j := null$

Trajectories:
$w.send_j \equiv null$

It is important to note that the abstract protector automaton $Abs($MERGE-VEHICLES, $S_{\{i,i'\}}$, $R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$ complies with the assumptions made about the abstract protector in Section 3.2.1. In particular, since the vehicle location variables, the vehicle velocity variables, and the *collided* variables are output variables of the MERGE-VEHICLES automaton, the set *safe* is $Y_{\text{MERGE-VEHICLES}}$-determinable and actions that guarantee safety can be determined from the output variables of the MERGE-VEHICLES automaton (Axioms 3.2.4 and 3.2.5, respectively). Moreover, the sets $R_{\{i,i'\}}$ and $G_{\{i,i'\}}$ are $Y_{\text{MERGE-VEHICLES}}$-determinable (Axioms 3.2.6 and 3.2.7, respectively) and the set of start states $S_{\{i,i'\}}$ is a subset of the set *safe* (Axiom 3.2.8), since $S_{\{i,i'\}}$ is defined to be the set *safe*.

In Section 3.1 it was shown that the abstract protector Abs_j guarantees that the physical plant PP remains within G starting from S given R. Similarly, the MERGE-PROT-PAIR$_{\{i,i'\}}$ automaton guarantees that the MERGE-VEHICLES automaton remains within $G_{\{i,i'\}}$ starting from $S_{\{i,i'\}}$ given $R_{\{i,i'\}}$. This is shown in the following section.

7.4 Correctness of MERGE-PROT-PAIR$_{\{i,i'\}}$

The main result to be shown is that MERGE-PROT-PAIR$_{\{i,i'\}} \leq Abs($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. Since both MERGE-PROT-PAIR$_{\{i,i'\}}$ and $Abs($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$ involve the composition of the same sensor automaton with distinct discrete controller automata, Theorem 2.7.4 applies. Therefore, it suffices to show that the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ of Figure 7.2 implements the discrete controller automaton $DC($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$ of Figure 3.3. From Theorem 2.6.1, this follows by showing that there exists a simulation relation between the states of the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ and the discrete controller automaton $DC($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. We first give some set definitions, then prove some lemmas, and finally show the existence of such a simulation relation.

In this section, we use the notation $future_{\{i,i'\}}$, $safe_{\{i,i'\}}$, $very\text{-}safe_{\{i,i'\}}$, and $delay\text{-}safe_{\{i,i'\}}$ to denote the specialization of the function *future*, the sets *safe* and *very-safe*, and the function *delay-safe*, which are defined in Section 3.2.1, to the automaton MERGE-VEHICLES, the sets $R_{\{i,i'\}}$ and $G_{\{i,i'\}}$, and the the port j of the MERGE-PROT-PAIR$_{\{i,i'\}}$ protector. Moreover, since the environment action of the MERGE-VEHICLES automaton is stuttering, its consideration is omitted in all inductive proofs involving the PP automaton.

We proceed by defining several sets that are used in the correctness proof of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$. For reference, their formal definitions appear in Table 7.2.

Let $W_{\{i,i'\}}$ be the subset of $R_{\{i,i'\}} \cap G_{\{i,i'\}}$ comprised of the states in which the section of the track owned by the vehicle i does not overlap the section of track owned by the vehicle i'; that is, $W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap disjoint\text{-}owned\text{-}tracks(i, i')$.

Let $V_{(i,i')}$ be the subset of $W_{\{i,i'\}}$ comprised of the states in which the vehicle i is being instructed to brake by the protector j and either the locations of the vehicles i and i' are comparable and $l_i < l_{i'}$, *i.e.*, the vehicle i is trailing the vehicle i', or the locations of the vehicles i and i' are incomparable and the section of the track owned by the vehicle i is entirely upstream of the merge point $\langle \mathtt{out}, 0\rangle$. Moreover, let $V_{\{i,i'\}}$ be defined as $V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$.

Let $T_{\{i,i'\}}(t)$, where $t \in \mathbb{R}^{\geq 0}$, be the subset of $R_{\{i,i'\}} \cap G_{\{i,i'\}}$ comprised of the states in which the section of the track claimed in time t by the vehicle i does not overlap the section of the track claimed in time t by the vehicle i'; that is, $T_{\{i,i'\}}(t) = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap disjoint\text{-}claimed\text{-}tracks(i, i', t)$.

The following lemma defines the relation among the sets $G_{\{i,i'\}}$, $W_{\{i,i'\}}$, $V_{\{i,i'\}}$, and $T_{\{i,i'\}}(t)$, for $t \in \mathbb{R}^{\geq 0}$.

Lemma 7.4.1 *For all $t, t' \in \mathbb{R}^{\geq 0}$, $t \leq t'$, the following hold:*

Table 7.2 Sets used in the correctness proof of MERGE-PROT-PAIR$_{\{i,i'\}}$.

$W_{\{i,i'\}} \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$$

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

$$V_{(i,i')} = \{p \in W_{\{i,i'\}} \cap P_{B_{ij}} \mid (p \in \mathit{comparable}(i, i') \wedge p.l_i < p.l_{i'}) \\ \vee (p \in \mathit{incomparable}(i, i') \wedge \max(p.O_i) < \langle \mathtt{out}, 0\rangle)\}$$

$V_{\{i,i'\}} \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$$

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

$$T_{\{i,i'\}}(t) = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}claimed\text{-}tracks}(i, i', t)$$

---

1. $T_{\{i,i'\}}(t) \subseteq W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$.
2. $V_{\{i,i'\}} \subseteq W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$.
3. $T_{\{i,i'\}}(t') \subseteq T_{\{i,i'\}}(t)$.
4. $T_{\{i,i'\}}(0) = W_{\{i,i'\}}$.

Proof: Follow directly from the definitions of the sets $V_{\{i,i'\}}$, $W_{\{i,i'\}}$, and $T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and Lemma 4.4.2. ■

In the following three lemmas, we show that any state $R_{\{i,i'\}}$-reachable from a state in $V_{(i,i')}$ through an execution fragment that involves no input actions on port j, is in $W_{\{i,i'\}}$. In the first lemma, we show that if the final state of such an execution fragment is in $G_{\{i,i'\}}$ and the section of track owned by the vehicle i has not grown since the beginning of the execution fragment, then the final state of the execution fragment is in $W_{\{i,i'\}}$. In the second lemma, we show that the final state of any such execution fragment is in $G_{\{i,i'\}}$ and the section of track owned by the vehicle i does not grow throughout the execution fragment. Finally, the third lemma combines these two results and states formally the desired property.

Lemma 7.4.2 *Let $p \in V_{(i,i')}$ and $p' \in \mathit{future}_{\{i,i'\}}(p, \mathbb{R}^{\geq 0})$. If $p' \in G_{\{i,i'\}}$ and $p'.O_i \subseteq p.O_i$, then $p' \in W_{\{i,i'\}}$.*

Proof: We need to show that $p' \in W_{\{i,i'\}}$; that is, we need to show that the state p' is in the set $R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$. By assumption, it is the case that $p' \in G_{\{i,i'\}}$.

Therefore, it remains to be shown that $p' \in R_{\{i,i'\}}$ and $p' \in$ *disjoint-owned-tracks*(i, i'). We consider these two conditions by cases:

1. $p' \in R_{\{i,i'\}}$.

   This is the case because the function $\mathit{future}_{\{i,i'\}}(p, \mathbb{R}^{\geq 0})$ only considers $R_{\{i,i'\}}$-reachable states.

2. $p' \in$ *disjoint-owned-tracks*(i, i').

   Since $p \in V_{(i,i')}$, there are two possible cases: (i) $p \in$ *comparable*(i, i') and $p.l_i < p.l_{i'}$, and (ii) $p \in$ *incomparable*(i, i') and $\max(p.O_i) < \langle \texttt{out}, 0 \rangle$.

   In the first case, it is as if the vehicle i is trailing the vehicle i' on a single track. Since $p \in V_{(i,i')} \subseteq W_{\{i,i'\}}$, the sections of the track owned by the vehicles i and i' in state p are disjoint. Since $p \in$ *comparable*(i, i') and $p.l_i < p.l_{i'}$, it follows that $\max(p.O_i) < \min(p.O_{i'})$. Moreover, Lemma 4.4.2, part 2, implies that $\max(p.O_i) < p.l_{i'}$. Therefore, because of the non-negative constraint on the vehicle velocities and the assumption that $p'.O_i \subseteq p.O_i$, it follows that $p' \in$ *disjoint-owned-tracks*(i, i').

   In the second case, since $\max(p.O_i) < \langle \texttt{out}, 0 \rangle$, the section of the track owned by the vehicle i in state p is strictly within the incoming directed edge $p.l_i.e$. Since $p'.O_i \subseteq p.O_i$, the same is true for the section of track owned by the vehicle i in state p'. Therefore, since the vehicle i' is traveling on the adjacent incoming branch, it follows that $p' \in$ *disjoint-owned-tracks*(i, i').

■

Lemma 7.4.3 *If $p \in V_{(i,i')}$ and $p' \in \mathit{future}_{\{i,i'\}}(p, \mathbb{R}^{\geq 0})$, then $p' \in G_{\{i,i'\}}$ and $p'.O_i \subseteq p.O_i$.*

Proof: Let α be an execution fragment of the MERGE-VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $V_{(i,i')}$, is only comprised of states in $R_{\{i,i'\}}$, and involves no input actions on port j. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_{\{i,i'\}}$ and $p_{final}.O_i \subseteq p_{init}.O_i$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and therefore, $p_{final} = p_{init}$. From Lemma 7.4.1, part 2, and the fact that $p_{init} \in V_{(i,i')} \subseteq V_{\{i,i'\}}$, it follows that $p_{final} \in G_{\{i,i'\}}$. Moreover, the fact that $p_{final}.O_i \subseteq p_{init}.O_i$ is trivially true.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{final} \in G_{\{i,i'\}}$ and $p_{final}.O_i \subseteq p_{init}.O_i$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{final} \in G_{\{i,i'\}}$ and $p'_{final}.O_i \subseteq p'_{init}.O_i$. Moreover, from Lemma 7.4.2 it follows that $p'_{final} \in W_{\{i,i'\}}$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, we consider all possible actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$, are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and does not affect the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i)$ action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Moreover, since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_i \leq p'_{final}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_i \subseteq p'_{final}.O_i$. However, from the induction hypothesis it is the case that $p'_{final}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.O_i \subseteq p_{init}.O_i$, as needed.
3. the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, affect neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Moreover, since the input actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions $\texttt{brick-wall}(i'')$, for $i'' \in I, i'' \neq i$, do not affect the velocity of the vehicle i, it is the case that $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_i \subseteq p'_{final}.O_i$. However, from the induction hypothesis it is the case that $p'_{final}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.O_i \subseteq p_{init}.O_i$, as needed.
4. the internal actions $\texttt{colliding-pair}(i'', i''')$, for $i'', i''' \in I, i'' \neq i'''$, and the internal actions $\texttt{collision-effects}(i'''')$, for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{final} \in W_{\{i,i'\}}$.

Since $p_{init} \in V_{(i,i')} \subseteq P_{B_{ij}}$ and the execution fragment leading from p_{init} to p'_{final} involves no input actions on port j, it follows that $p'_{final} \in P_{B_{ij}}$. Therefore, in the case of a trajectory from p'_{final} to p_{final}, Lemma 4.4.4, part 1, implies that $p_{final}.O_i \subseteq p'_{final}.O_i$. However, from the induction hypothesis it is the case that $p'_{final}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.O_i \subseteq p_{init}.O_i$. Moreover, since $p'_{final} \in G_{\{i,i'\}}$ and the variables *collided*(i, i') and *collided*(i', i) remain constant throughout the trajectory, it follows that $p_{final} \in G_{\{i,i'\}}$, as needed. ■

Lemma 7.4.4 $future_{\{i,i'\}}(V_{(i,i')}, \mathbb{R}^{\geq 0}) \subseteq W_{\{i,i'\}}$.

Proof: Follows directly from Lemmas 7.4.2 and 7.4.3. ■

In the following lemma, we extend the result of Lemma 7.4.4 to the set $V_{\{i,i'\}}$.

Lemma 7.4.5 $future_{\{i,i'\}}(V_{\{i,i'\}}, \mathbb{R}^{\geq 0}) \subseteq W_{\{i,i'\}}$.

Proof: Follows directly from Lemma 7.4.4 and the fact that $V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$. ■

In the following two lemmas, we use Lemma 7.4.5 to show that $V_{\{i,i'\}} \subseteq$ *very-safe*$_{\{i,i'\}}$ and $V_{\{i,i'\}} \subseteq$ *delay-safe*$_{\{i,i'\}}(t)$, for any $t \in \mathbb{R}^{\geq 0}$, respectively.

Lemma 7.4.6 $V_{\{i,i'\}} \subseteq$ *very-safe*$_{\{i,i'\}}$.

Proof: Follows directly from Lemma 7.4.5 and Lemma 7.4.1, part 1. ■

Lemma 7.4.7 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $V_{\{i,i'\}} \subseteq$ delay-safe$_{\{i,i'\}}(t)$.*

Proof: Follows directly from Lemma 7.4.6 and Lemma 3.2.5, part 1. ■

In the next three lemmas and the subsequent corollary, we show that the sets $W_{\{i,i'\}}$ and *safe*$_{\{i,i'\}}$ are equal. First, we show that any state that is $R_{\{i,i'\}}$-reachable from a state p in $W_{\{i,i'\}}$ through an execution fragment that involves no input actions on port j and has a limit time equal to zero, is in the set $W_{\{i,i'\}}$. Then, we show that $W_{\{i,i'\}} \subseteq$ *safe*$_{\{i,i'\}}$ and *safe*$_{\{i,i'\}} \subseteq W_{\{i,i'\}}$. Finally, the subsequent corollary states that $W_{\{i,i'\}} =$ *safe*$_{\{i,i'\}}$.

Lemma 7.4.8 $future_{\{i,i'\}}(W_{\{i,i'\}}, 0) \subseteq W_{\{i,i'\}}$.

Proof: Let α be an execution fragment of the MERGE-VEHICLES automaton of n steps, where $n \in \mathbb{N}$, that: starts in a state in $W_{\{i,i'\}}$, is only comprised of states in $R_{\{i,i'\}}$, involves no input actions on port j, and has a limit time equal to zero. Let p_{init} and p_{final} be the initial and final states of α, respectively. By induction on the length n of the execution fragment α, we show that $p_{final} \in W_{\{i,i'\}}$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of no steps and, therefore, $p_{final} = p_{init}$. Since $p_{init} \in W_{\{i,i'\}}$, it follows that $p_{final} \in W_{\{i,i'\}}$.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{\mathit{final}} \in W_{\{i,i'\}}$. Let α' be the part of the execution fragment α comprised of the first k steps. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}}$. Since the final state of α is reached from the final state of α' by a single step, the inductive step involves the consideration of all possible steps leading from p'_{final} to p_{final}.

To complete the induction, we consider all possible discrete actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$, are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and affects neither the velocity of the vehicle i', nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i)$ action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. Moreover, since the $\texttt{brick-wall}(i)$ action does not affect the velocity of the vehicle i', it is the case that $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$ and $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it follows that $p_{\mathit{final}} \in$ *disjoint-owned-tracks*(i, i').

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.
3. the $\texttt{brick-wall}(i')$ action sets the velocity of the vehicle i' to zero and affects neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i')$ action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_{i'} \leq p'_{\mathit{final}}.\dot{x}_{i'}$. Moreover, since the $\texttt{brick-wall}(i')$ action does not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$ and $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it follows that $p_{\mathit{final}} \in$ *disjoint-owned-tracks*(i, i').

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.
4. the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, affect neither the velocities of the vehicles i and i', nor the *collided*(i, i') and *collided*(i', i) variables.

From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

Moreover, since the input actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, do not affect the velocities of the vehicles i and i', it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$ and $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$ and $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$, it is the case that $p_{\mathit{final}} \in \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$.

Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.

5. the internal actions $\texttt{colliding-pair}(i'', i''')$, for $i'', i''' \in I, i'' \neq i'''$, and the internal actions $\texttt{collision-effects}(i'''')$, for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{\mathit{final}} \in W_{\{i,i'\}}$.

■

Lemma 7.4.9 $W_{\{i,i'\}} \subseteq \mathit{safe}_{\{i,i'\}}$.

Proof: From the definition of *safe* in Section 3.2.1, we must show that any state $p \in W_{\{i,i'\}}$ satisfies: (i) $\mathit{future}_{\{i,i'\}}(p, 0) \subseteq G_{\{i,i'\}}$, and (ii) there exists some input action π on port j such that for every $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in \mathit{future}_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in \mathit{very\text{-}safe}_{\{i,i'\}}$.

(i) Since $p \in W_{\{i,i'\}}$, the first condition follows from Lemma 7.4.8 and Lemma 7.4.1, part 1.

(ii) For the second condition, let π be the action $\texttt{protect}(C)_j$, where $C = \{i\}$, if $p \in \mathit{yield}(i, i')$, and $C = \{i'\}$, otherwise. Without loss of generality, let $p \in \mathit{yield}(i, i')$ and $C = \{i\}$.

Throughout the execution fragment from p to p', the actions $\texttt{colliding-pair}(i'', i''')$, for $i'', i''' \in I, i'' \neq i'''$, and $\texttt{collision-effects}(i'''')$, for $i'''' \in I$, are not enabled. Therefore, since none of the other discrete actions of the MERGE-VEHICLES automaton can increase the velocities of the vehicles i and i', Lemma 4.4.3, part 1, implies that $p'.O_i \subseteq p.O_i$ and $p'.O_{i'} \subseteq p.O_{i'}$. Moreover, since the $\texttt{protect}(\{i\})_j$ action does not affect the velocity of the vehicle i, Lemma 4.4.3, part 1, implies that $p''.O_i \subseteq p.O_i$. Since $p''.O_i \subseteq p.O_i$ and $p \in \mathit{yield}(i, i')$, it is the case that in the state p'' either the locations of the vehicles i and i' are comparable and the vehicle i is trailing the vehicle i', or the locations of the vehicles i and i' are not comparable and the section of track owned by the vehicle i is entirely upstream of the merge point $\langle \texttt{out}, 0 \rangle$.

Moreover, considering the step from p' to p'', the $\texttt{protect}(\{i\})_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables. Therefore, since Lemma 7.4.8 implies that $p' \in W_{\{i,i'\}}$, it follows that $p'' \in R_{\{i,i'\}}$ and $p'' \in G_{\{i,i'\}}$. In addition, since the $\texttt{protect}(\{i\})_j$ action does not affect the velocities of the vehicles i and i', Lemma 4.4.3, part 1, implies that $p''.O_i \subseteq p'.O_i$ and $p''.O_{i'} \subseteq p'.O_{i'}$. Therefore, since $p' \in W_{\{i,i'\}}$, it follows that $p'' \in$ *disjoint-owned-tracks*(i, i'). From the above conditions, it follows that $p'' \in W_{\{i,i'\}}$.

In addition, since the $\texttt{protect}(\{i\})_j$ action sets the variable *brake-req*(i, j) to $\texttt{True}$, it is also the case that $p'' \in P_{B_{ij}}$.

Thus, since $p'' \in W_{\{i,i'\}}$, $p'' \in P_{B_{ij}}$, and either the locations of the vehicles i and i' in the state p'' are comparable and the vehicle i is trailing the vehicle i', or the locations of the vehicles i and i' in the state p'' are not comparable and the section of track owned by the vehicle i is entirely upstream of the merge point $\langle \texttt{out}, 0\rangle$, it follows that $p'' \in V_{(i,i')} \subseteq V_{\{i,i'\}}$.

Finally, Lemma 7.4.6 implies that $p'' \in$ *very-safe*$_{\{i,i'\}}$, as needed. ■

Lemma 7.4.10 *For any $p \in R_{\{i,i'\}}$, if $p \in$ safe$_{\{i,i'\}}$ then $p \in W_{\{i,i'\}}$.*

Proof: We show the contrapositive; that is, for any $p \in R_{\{i,i'\}}$, if $p \notin W_{\{i,i'\}}$ then $p \notin$ *safe*$_{\{i,i'\}}$. Since $W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-owned-tracks*(i, i') and $p \in R_{\{i,i'\}}$, we consider the conditions $p \notin G_{\{i,i'\}}$ and $p \notin$ *disjoint-owned-tracks*(i, i') separately.

1. $p \notin G_{\{i,i'\}}$.

   From Lemma 3.2.4, part 1, it is the case that *safe*$_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Since $p \notin G_{\{i,i'\}}$, it follows that $p \notin$ *safe*$_{\{i,i'\}}$.

2. $p \notin$ *disjoint-owned-tracks*(i, i').

   We must show that $p \notin$ *safe*$_{\{i,i'\}}$. In order for the state $p \in R_{\{i,i'\}}$ to be in the set *safe*$_{\{i,i'\}}$ there must exist some input action π on port j such that for every $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in$ *very-safe*$_{\{i,i'\}}$. Therefore, it suffices to show that for any input action π on port j, there exist $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin$ *very-safe*$_{\{i,i'\}}$.

   Without loss of generality, suppose that the vehicles i and i' are traveling on adjacent branches in the state p, *i.e.*, $p \in$ *incomparable*(i, i'), and let $\pi = \texttt{protect}(\{i, i'\})_j$.

   Since Lemma 3.2.1, part 3, implies that $p \in$ *future*$_{\{i,i'\}}(p, 0)$, consider the case where $p' = p$. Since $p' = p$ and the input action $\texttt{protect}(\{i, i'\})_j$ affects neither the location, nor the velocity of the vehicles i and i', it follows that $p''.l_i = p'.l_i = p.l_i$, $p''.\dot{x}_i = p'.\dot{x}_i = p.\dot{x}_i$, $p''.l_{i'} = p'.l_{i'} = p.l_{i'}$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'} = p.\dot{x}_{i'}$. Therefore, since the section of track owned by any vehicle depends only on its location and its velocity, it is the case that $p''.O_i = p'.O_i = p.O_i$ and $p''.O_{i'} = p'.O_{i'} = p.O_{i'}$. Therefore, since $p \notin$ *disjoint-owned-tracks*(i, i'), $p''.O_i = p.O_i$, and $p''.O_{i'} = p.O_{i'}$, it follows that $p'' \notin$ *disjoint-owned-tracks*(i, i'). Moreover, since the vehicles i and i' are traveling on adjacent branches in state p, $p''.l_i = p.l_i$, $p''.l_{i'} = p.l_{i'}$, and $p'' \notin$ *disjoint-owned-tracks*(i, i'), it follows that $\langle \texttt{out}, 0\rangle \in p''.O_i$ and $\langle \texttt{out}, 0\rangle \in p''.O_{i'}$.

Again, without loss of generality, suppose that the vehicle i' is the first of the vehicles i and i' to reach the merge point $\langle \texttt{out}, 0\rangle$ and that the vehicles i and i' have not collided up until the point in time when the vehicle i' reaches the merge point. Moreover, consider the evolution of the MERGE-VEHICLES automaton following the state p'' in which a `brick-wall`(i') action is executed at the exact instant in time when the location of the vehicle i' equals the merge point $\langle \texttt{out}, 0\rangle$ and the vehicles i and i' move forward and remain stationary thereafter, respectively. Since $\langle \texttt{out}, 0\rangle \in p''.O_i$, it follows that at some state of such an evolution the action `colliding-pair`(i, i') is enabled and, subsequently, executed. The state of the MERGE-VEHICLES automaton following the execution of the action `colliding-pair`(i, i') would, therefore, not be in $G_{\{i,i'\}}$. It follows that $p'' \notin$ *very-safe*$_{\{i,i'\}}$ which implies that $p \notin$ *safe*$_{\{i,i'\}}$.

Using similar analyses, it can be shown that for any $p \in R_{\{i,i'\}}$ and any input action π on port j, there exist $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin$ *very-safe*$_{\{i,i'\}}$. It follows that $p \notin$ *safe*$_{\{i,i'\}}$, as needed.

■

Corollary 7.4.11 $W_{\{i,i'\}} =$ *safe*$_{\{i,i'\}}$.

Proof: Follows directly from Lemmas 7.4.9 and 7.4.10. ■

In the next few lemmas, we show that any state p in the set $T_{\{i,i'\}}(t)$, for any $t \in \mathbb{R}^{\geq 0}$, is in the set *delay-safe*$_{\{i,i'\}}(t)$; that is, any state $R_{\{i,i'\}}$-reachable from p within an amount of time t through an execution fragment that involves no input actions on port j, is in the set $G_{\{i,i'\}}$ and any state $R_{\{i,i'\}}$-reachable from the state p in exactly an amount of time t through an execution fragment that involves no input actions on port j, is in the set *safe*$_{\{i,i'\}}$.

Lemma 7.4.12 *Let $p \in T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in$ future$_{\{i,i'\}}(p, t)$, where $t \in [0, \tau]$. If $p' \in G_{\{i,i'\}}$, $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$, and $p'.C_{i'}(\tau - t) \subseteq p.C_{i'}(\tau)$, then $p' \in T_{\{i,i'\}}(\tau - t)$.*

Proof: We need to show that $p' \in R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-claimed-tracks*$(i, i', \tau - t)$. Since $p' \in G_{\{i,i'\}}$, it remains to be shown that $p' \in R_{\{i,i'\}}$ and $p' \in$ *disjoint-claimed-tracks*$(i, i', \tau - t)$. We consider these two conditions by cases:

1. $p' \in R_{\{i,i'\}}$.

   This is the case because the function $future_{\{i,i'\}}(p,t)$ only considers $R_{\{i,i'\}}$-reachable states.

2. $p' \in \textit{disjoint-claimed-tracks}(i,i',\tau-t)$.

   Since $p \in \textit{disjoint-claimed-tracks}(i,i',\tau)$, $p'.C_i(\tau-t) \subseteq p.C_i(\tau)$, and $p'.C_{i'}(\tau-t) \subseteq p.C_{i'}(\tau)$, it follows that $p' \in \textit{disjoint-claimed-tracks}(i,i',\tau-t)$, as needed.

■

Lemma 7.4.13 *For all $p \in T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in future_{\{i,i'\}}(p,t)$, where $t \in [0,\tau]$, it is the case that $p' \in G_{\{i,i'\}}$, $p'.C_i(\tau-t) \subseteq p.C_i(\tau)$, and $p'.C_{i'}(\tau-t) \subseteq p.C_{i'}(\tau)$.*

Proof: Let $\tau \in \mathbb{R}^{\geq 0}$ and α be an execution fragment of the MERGE-VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $T_{\{i,i'\}}(\tau)$, is only comprised of states in $R_{\{i,i'\}}$, involves no input actions on port j, and has a limit time t that lies in the interval $[0,\tau]$. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_{\{i,i'\}}$, $p_{final}.C_i(\tau-t) \subseteq p_{init}.C_i(\tau)$, and $p_{final}.C_{i'}(\tau-t) \subseteq p_{init}.C_{i'}(\tau)$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n=0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$ and $\alpha.ltime = 0$, *i.e.*, $t=0$. From Lemma 7.4.1, part 1, and the fact that $p_{init} \in T_{\{i,i'\}}(\tau)$, it follows that $p_{final} \in G_{\{i,i'\}}$. Moreover, since $t=0$, the conditions $p_{final}.C_i(\tau-t) \subseteq p_{init}.C_i(\tau)$ and $p_{final}.C_{i'}(\tau-t) \subseteq p_{init}.C_{i'}(\tau)$ are trivially true.

The inductive step involves showing that if α is an execution fragment of length $n=k+1$, for some $k \in \mathbb{N}$, with $\alpha.ltime = t$, where $t \in [0,\tau]$, then $p_{final} \in G_{\{i,i'\}}$, $p_{final}.C_i(\tau-t) \subseteq p_{init}.C_i(\tau)$, and $p_{final}.C_{i'}(\tau-t) \subseteq p_{init}.C_{i'}(\tau)$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories and let $\alpha'.ltime = t'$, where $t' \in [0,t]$. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{final} \in G_{\{i,i'\}}$, $p'_{final}.C_i(\tau-t') \subseteq p'_{init}.C_i(\tau)$, and $p'_{final}.C_{i'}(\tau-t') \subseteq p'_{init}.C_{i'}(\tau)$. Moreover, from Lemma 7.4.12 it follows that $p'_{final} \in T_{\{i,i'\}}(\tau-t')$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, keeping in mind that the limit times of α' and α are equal, *i.e.*, $t'=t$, we consider all possible actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i,i'\})$, are not enabled because α involves no input actions on port j.

2. the `brick-wall`(i) action sets the velocity of the vehicle i to zero and affects neither the velocity of the vehicle i', nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the `brick-wall`(i) action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_i \leq p'_{final}.\dot{x}_i$. Moreover, since the `brick-wall`(i) action does not affect the velocity of the vehicle i', it is the case that $p_{final}.\dot{x}_{i'} = p'_{final}.\dot{x}_{i'}$. From Lemma 4.4.3, part 2, it follows that $p_{final}.C_i(\tau - t) \subseteq p'_{final}.C_i(\tau - t')$ and $p_{final}.C_{i'}(\tau - t) \subseteq p'_{final}.C_{i'}(\tau - t')$. However, from the induction hypothesis we have $p'_{final}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$ and $p'_{final}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{final}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$, as needed.

3. the `brick-wall`(i') action sets the velocity of the vehicle i' to zero and affects neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the `brick-wall`(i') action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_{i'} \leq p'_{final}.\dot{x}_{i'}$. Moreover, since the `brick-wall`(i') action does not affect the velocity of the vehicle i, it is the case that $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$. From Lemma 4.4.3, part 2, it follows that $p_{final}.C_i(\tau - t) \subseteq p'_{final}.C_i(\tau - t')$ and $p_{final}.C_{i'}(\tau - t) \subseteq p'_{final}.C_{i'}(\tau - t')$. However, from the induction hypothesis we have $p'_{final}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$ and $p'_{final}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{final}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$, as needed.

4. the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, affect neither the velocities of the vehicles i and i', nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Moreover, since the input actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, do not affect the velocities of the vehicles i and i', it is the case that $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$ and $p_{final}.\dot{x}_{i'} = p'_{final}.\dot{x}_{i'}$. From Lemma 4.4.3, part 2, it follows that $p_{final}.C_i(\tau - t) \subseteq p'_{final}.C_i(\tau - t')$ and $p_{final}.C_{i'}(\tau - t) \subseteq p'_{final}.C_{i'}(\tau - t')$. However, from the induction hypothesis we have $p'_{final}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$ and $p'_{final}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Therefore, since $p_{init} =$

p'_{init}, it follows that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{final}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$, as needed.

5. the internal actions `colliding-pair`(i'', i'''), for $i'', i''' \in I, i'' \neq i'''$, and the internal actions `collision-effects`(i''''), for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{final} \in T_{\{i,i'\}}(\tau - t')$.

In the case of a trajectory, Lemma 4.4.4, part 2, applies and it follows that $p_{final}.C_i(\tau - t) \subseteq p'_{final}.C_i(\tau - t')$ and $p_{final}.C_{i'}(\tau - t) \subseteq p'_{final}.C_{i'}(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{final}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$ and $p'_{final}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{final}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{final}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$. Moreover, since $p'_{final} \in G_{\{i,i'\}}$ and the *collided*(i, i') and *collided*(i', i) variables remain constant throughout the trajectory, it follows that $p_{final} \in G_{\{i,i'\}}$, as needed. ■

Lemma 7.4.14 *For $\tau \in \mathbb{R}^{\geq 0}$ and $t \in [0, \tau]$, it is the case that $future_{\{i,i'\}}(T_{\{i,i'\}}(\tau), t) \subseteq T_{\{i,i'\}}(\tau - t)$.*

Proof: Follows directly from Lemmas 7.4.12 and 7.4.13. ■

Corollary 7.4.15 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $future_{\{i,i'\}}(T_{\{i,i'\}}(t), 0) \subseteq T_{\{i,i'\}}(t)$.*

Proof: Follows directly from Lemma 7.4.14. ■

Lemma 7.4.16 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $T_{\{i,i'\}}(t) \subseteq delay\text{-}safe_{\{i,i'\}}(t)$.*

Proof: From the definition of *delay-safe* in Section 3.2.1, we must show that:

1. $future_{\{i,i'\}}(T_{\{i,i'\}}(t), [0, t]) \subseteq G_{\{i,i'\}}$, and
2. $future_{\{i,i'\}}(T_{\{i,i'\}}(t), t) \subseteq safe_{\{i,i'\}}$.

The first condition follows directly from Lemma 7.4.14 and Lemma 7.4.1, part 1. Moreover, Lemma 7.4.14 and Lemma 7.4.1, part 4, imply that $future_{\{i,i'\}}(T_{\{i,i'\}}(t), t) \subseteq W_{\{i,i'\}}$. Therefore, the second condition follows from Lemma 7.4.9. ■

In the following lemma, we show that the MERGE-PROT-PAIR$_{\{i,i'\}}$ protector implements the *Abs*(MERGE-VEHICLES, $S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d$) protector. Since the protector automata MERGE-PROT-PAIR$_{\{i,i'\}}$ and Abs_j involve the composition of the same sensor automaton with distinct controller automata, it suffices to show that the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ implements the discrete controller automaton *DC*(MERGE-VEHICLES, $S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d$).

Lemma 7.4.17 MERGE-PROT-PAIR$_{\{i,i'\}} \leq Abs($MERGE-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$.

Proof: Both the MERGE-PROT-PAIR$_{\{i,i'\}}$ and the Abs_j protectors involve the composition of the same sensor automaton with distinct controller automata. From Theorem 2.7.4, it suffices to show that the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ implements DC_j. This is shown by a simulation from the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ to DC_j.

The mapping between the states of the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ and DC_j is almost the identity. In the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$, the variable $send_j$ is equal to either a member of $\mathcal{P}(\{i, i'\})$, or the value *null*. In DC_j, these valuations simply map to either the actions $\texttt{protect}(C)_j$, where C is the member of $\mathcal{P}(\{i, i'\})$ that corresponds to the valuation of the variable $send_j$ of the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$, or the value *null*, respectively.

The start states for the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ and DC_j are the states in which $send_j = null$. These are related to each other according to the mapping discussed above.

Furthermore, since the trajectories in both discrete controller automata are identical, we need only consider their discrete transitions. We analyze the actions of the implementation by cases, letting p denote any complete state of the MERGE-VEHICLES automaton that corresponds to the output state y, *i.e.*, $p \in VALID$ and $p \lceil Y_{\text{MERGE-VEHICLES}} = y$.

1. The $\texttt{snapshot}(y)_j$ action of the implementation sets $send_j$ to an element of $\mathcal{P}(\{i, i'\})$. In order to show that the behavior of the implementation is allowed by the specification, we must show that the input action $\texttt{snapshot}(y)_j$ of the implementation sets the value of the $send_j$ variable in such a way that the subsequently enabled action π of the implementation (i) guarantees that for all $p', p'' \in R_{\{i,i'\}}$ such that $p' \in future_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in delay\text{-}safe_{\{i,i'\}}(d)$, if $p \in safe_{\{i,i'\}}$, and (ii) is an arbitrary output action of the implementation, otherwise.

   First, consider the case in which $p \in safe_{\{i,i'\}}$. Since Corollary 7.4.11 implies that $p \in W_{\{i,i'\}}$, the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ according to whether the state p is in $T_{\{i,i'\}}(d)$, or not.

   On one hand, if $p \notin T_{\{i,i'\}}(d)$ then the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to either $\{i\}$, or $\{i'\}$ according to the strategy described in Section 7.3. Therefore, the $\texttt{snapshot}(y)_j$ action enables either the $\texttt{protect}(\{i\})_j$ action, or the $\texttt{protect}(\{i'\})_j$ action. Since $p \in W_{\{i,i'\}}$, Lemma 7.4.8 implies that $p' \in W_{\{i,i'\}}$. Moreover, since the $\texttt{protect}(\{i\})_j$ and $\texttt{protect}(\{i'\})_j$ actions affect neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_{\{i,i'\}}$, $p'' \in G_{\{i,i'\}}$, $p''.\dot{x}_i = p'.\dot{x}_i$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'}$. Therefore, since $p' \in W_{\{i,i'\}}$, Lemma 4.4.3, part 1, implies that $p'' \in$ *disjoint-owned-tracks*(i, i'). From the above conditions, it follows that $p'' \in W_{\{i,i'\}}$. Moreover, since the $\texttt{protect}(\{i\})_j$ and $\texttt{protect}(\{i'\})_j$ actions set the *brake-req*(i, j) and *brake-req*(i', j) variables, respectively, to `True`, it follows that $p'' \in V_{\{i,i'\}}$. Finally, Lemma 7.4.7 implies that $p'' \in$ *delay-safe*$_{\{i,i'\}}(d)$, as needed.

On the other hand, if $p \in T_{\{i,i'\}}(d)$ then the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to $\emptyset$ and the $\texttt{protect}(\emptyset)_j$ action is enabled. Since $p \in T_{\{i,i'\}}(d)$, Corollary 7.4.15 implies that $p' \in T_{\{i,i'\}}(d)$. Moreover, since the $\texttt{protect}(\emptyset)_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_{\{i,i'\}}$, $p'' \in G_{\{i,i'\}}$, $p''.\dot{x}_i = p'.\dot{x}_i$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'}$. Therefore, since $p' \in T_{\{i,i'\}}(d)$, Lemma 4.4.3, part 2, implies that $p'' \in$ *disjoint-claimed-tracks*(i, i', d). From the above conditions, it follows that $p'' \in T_{\{i,i'\}}(d)$. Finally, Lemma 7.4.16 implies that $p'' \in$ *delay-safe*$_{\{i,i'\}}(d)$, as needed.

Next, consider the case in which $p \notin safe_{\{i,i'\}}$. In this case, the $\texttt{snapshot}(y)_j$ action of the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to either $\{i\}$, $\{i'\}$, or $\emptyset$ and, subsequently, enables either the $\texttt{protect}(\{i\})_j$ action, the $\texttt{protect}(\{i'\})_j$ action, or the $\texttt{protect}(\emptyset)_j$ action, respectively. However, when $p \notin safe_{\{i,i'\}}$, the DC_j automaton sets the variable $send_j$ arbitrarily and, subsequently, enables an arbitrary output action. Therefore, the behavior of the discrete controller automaton of the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ is allowed by that of the DC_j automaton.

Therefore, the effects of the $\texttt{snapshot}(y)_j$ action of the implementation are allowed by its specification.

2. The $\texttt{protect}(C)_j$ actions, for $C \in \mathcal{P}(\{i, i'\})$, have identical effects in both discrete controller automata. When the $send_j$ variable matches either the set C, or the $\texttt{protect}(C)_j$ action, respectively, the action $\texttt{protect}(C)_j$ is executed and the $send_j$ variable is set to *null* in both discrete controller automata.

3. The environment action in both discrete controller automata is stuttering. It follows that the mapping between the states of the discrete controller automaton of MERGE-PROT-PAIR$_{\{i,i'\}}$ and the DC_j automaton prior to and succeeding the execution of the environment action remains the same.

■

Corollary 7.4.18 *The protector* MERGE-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the automaton* MERGE-VEHICLES *remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Proof: Follows directly from Lemma 7.4.17 and Theorem 3.2.9. ■

Table 7.3 Formal definitions of MERGE-PROT, $G_{\text{MERGE-PROT}}$, $S_{\text{MERGE-PROT}}$, and $R_{\text{MERGE-PROT}}$.

$$\text{MERGE-PROT} \equiv \prod_{i,i' \in I, i \neq i'} \text{MERGE-PROT-PAIR}_{\{i,i'\}}$$

$$G_{\text{MERGE-PROT}} \equiv \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$$

$$S_{\text{MERGE-PROT}} \equiv \bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$$

$$R_{\text{MERGE-PROT}} \equiv P_{not\text{-}overspeed}$$

7.5 Protection System MERGE-PROT

We now define the collision protector MERGE-PROT. While considering the automaton MERGE-PROT, we restrict the states of the MERGE-VEHICLES automaton to $P_{not\text{-}overspeed}$ as defined in Section 4.2, *i.e.*, $R_{\text{MERGE-PROT}} = P_{not\text{-}overspeed}$. Let $G_{\text{MERGE-PROT}}$ and $S_{\text{MERGE-PROT}}$ be the intersection of $G_{\{i,i'\}}$ and $S_{\{i,i'\}}$, for all $\{i, i'\}$, where $i, i' \in I, i \neq i'$, respectively, and MERGE-PROT be the composition of MERGE-PROT-PAIR$_{\{i,i'\}}$, for all $\{i, i'\}$, where $i, i' \in I, i \neq i'$. The protector MERGE-PROT guarantees that MERGE-VEHICLES remains within $G_{\text{MERGE-PROT}}$ starting from $S_{\text{MERGE-PROT}}$ given $R_{\text{MERGE-PROT}}$. For reference, the formal definitions of the MERGE-PROT automaton and the sets $G_{\text{MERGE-PROT}}$, $S_{\text{MERGE-PROT}}$, and $R_{\text{MERGE-PROT}}$ are shown in Table 7.3.

Lemma 7.5.1 *The protector* MERGE-PROT *guarantees that the* MERGE-VEHICLES *automaton remains within* $G_{\text{MERGE-PROT}}$ *from* $S_{\text{MERGE-PROT}}$ *given* $R_{\text{MERGE-PROT}}$.

In the following proof, we show that all the states of an execution of $PP \times$ MERGE-PROT starting from $S_{\text{MERGE-PROT}}$ given $R_{\text{MERGE-PROT}}$ are in $G_{\text{MERGE-PROT}}$. This is done by applying Theorem 3.1.8 and showing that the second condition of the theorem does not hold.

Proof: Let α be any execution of the system $PP \times$ MERGE-PROT starting from a state in $S_{\text{MERGE-PROT}}$ and in which all states are in $R_{\text{MERGE-PROT}}$.

From Theorem 3.1.8, one of the following holds:

1. Every state in α is in $G_{\text{MERGE-PROT}} = \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$.
2. α can be written as $\alpha_1 \frown \alpha_2$, where

(a) All state occurrences in α_1 except possibly the last state occurrence are in the set $G_{\text{MERGE-PROT}} = \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$.

(b) If the last state occurrence in α_1 is in $\overline{G_{\{i,i'\}}}$, for some $i, i' \in I, i \neq i'$, then there exists $i'', i''' \in I, i'' \neq i''', \{i'', i'''\} \neq \{i, i'\}$, such that the last state occurrence in α_1 is in $\overline{G_{\{i'',i'''\}}}$.

(c) All state occurrences in α_2 except possibly the first state occurrence are in the set $\bigcap_{\{i'',i'''\} \in N} \mathit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$, for some $N \subseteq \{\{i,i'\} \mid i,i' \in I, i \neq i'\}$, where $|N| \geq 2$.

We proceed by showing that it is not possible to decompose α as $\alpha_1 \frown \alpha_2$ while satisfying the three aforementioned conditions.

The violation of $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ can only occur through the violation of at least one of the conditions $G_{\{i,i'\}}$, where $i, i' \in I, i \neq i'$. Moreover, each of these conditions are violated only through the execution of a `colliding-pair` action. Without loss of generality, suppose that the first condition that is violated in α is the condition $G_{\{i,i'\}}$, for some $i, i' \in I, i \neq i'$, and that such a violation has resulted through a `colliding-pair`(i,i') action. Let p and p' be the states of the MERGE-VEHICLES automaton prior to and succeeding this `colliding-pair`(i,i') action, *i.e.*, $p, p' \in R_{\text{MERGE-PROT}}$ such that $p \xrightarrow{\pi} p'$, where $\pi =$ `colliding-pair`(i,i'). Since the `colliding-pair`(i,i') action only sets the *collided*(i,i') variable to `True`, it follows that $p' \in \overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. Now, we attempt to decompose α as $\alpha_1 \frown \alpha_2$:

1. Suppose we split α at any state preceding the state p. Then the state p is in α_2. Since p' is the first state in which one of the conditions $G_{\{i'',i'''\}}$, for $i'', i''' \in I, i'' \neq i'''$, is violated, it is the case that $p \in \bigcap_{i'',i''' \in I, i'' \neq i'''} G_{\{i'',i'''\}}$ and there does not exist $N \subseteq \{\{i'',i'''\} \mid i'',i''' \in I, i'' \neq i'''\}$ such that $|N| \geq 2$ and $p \in \bigcap_{\{i'',i'''\} \in N} \mathit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

2. Suppose we split α at the state p. Then the state p' is in α_2. Since p' is the first state in which one of the conditions $G_{\{i'',i'''\}}$, for $i'', i''' \in I, i'' \neq i'''$, is violated and since the state p' is in $\overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$, it follows that there does not exist $N \subseteq \{\{i'',i'''\} \mid i'',i''' \in I, i'' \neq i'''\}$ such that $|N| \geq 2$ and $p' \in \bigcap_{\{i'',i'''\} \in N} \mathit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

3. Suppose we split α at the state p'. Then p' is the last state of α_1 and the first state of α_2. However, $p' \in \overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. Therefore, the second condition is violated and this decomposition of α is not valid.

4. Suppose we split α at any state succeeding p'. Then the state p' is in α_1. Since $p' \in \overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$, it follows that the state p' is not

in the set $\bigcap_{i'',i''' \in I, i'' \neq i'''} G_{\{i'',i'''\}}$. Therefore, the first condition is violated and this decomposition of α is not valid.

Therefore, the execution α cannot be decomposed into any such α_1 and α_2. It follows that the first clause of Theorem 3.1.8 must hold; that is, every state in α is in $G_{\text{MERGE-PROT}}$. This implies that the protector MERGE-PROT guarantees $G_{\text{MERGE-PROT}}$ in the MERGE-VEHICLES automaton starting from $S_{\text{MERGE-PROT}}$ given $R_{\text{MERGE-PROT}}$. ■

Chapter 8

Example 4: Collision Avoidance on a General Graph of Tracks

In this chapter, we consider a general track topology involving binary merges and diverges. We first augment the model of the PRT 2000™ to involve a track topology consisting of multiple branches interconnected by Y-shaped merges and diverges — the new model is referred to as the GRAPH-VEHICLES automaton. Then we define the protector GRAPH-PROT that guarantees that none of the vehicles of the GRAPH-VEHICLES automaton collide, assuming that they are all abiding by the speed limit. 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 $\{i, i'\}$ of vehicles of the GRAPH-VEHICLES automaton, for $i, i' \in I, i \neq i'$. Each of these GRAPH-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, is an implementation of a particular instantiation of the abstract protector automaton of Section 3.2 and guarantees that the vehicles i and i' do not collide into each other.

8.1 Augmented Physical Plant: GRAPH-VEHICLES

In this section we augment the model for the system of n vehicles to involve a track topology involving binary merges and diverges. This is done by extending the definition of the location of a vehicle to support a graph of tracks and by introducing an additional internal discrete action which is used to update the location variables of the vehicles as they cross the junction points in the track topology.

The track topology is represented by a directed graph $G = (V, E)$, where V and E denote the sets of vertices and edges of the graph G, respectively. The vertices and edges of the graph G correspond, respectively, to the junctions and branches of the track topology. Any edge e of the graph G is specified by an ordered pair of vertices that denote the initial and the final vertices of the directed edge e, *i.e.*, $e = \langle v_{init}, v_{final} \rangle$. We use the notation $e.v_{init}$ and $e.v_{final}$ to denote the initial and final vertices of the edge e, respectively. The function $length : E \rightarrow \mathbb{R}^{\geq 0}$ maps an edge to its length. Moreover, the functions $in(v)$, $out(v)$, and $e(v)$ map the vertex v of the graph G to its sets of incoming edges, outgoing edges, and both incoming and outgoing edges, respectively; that is, $in : V \rightarrow \mathcal{P}(E)$, $out : V \rightarrow \mathcal{P}(E)$, and $e : V \rightarrow \mathcal{P}(E)$, with $e(v) = in(v) \cup out(v)$, for all $v \in V$.

The graph G, as defined above, is assumed to satisfy the following conditions:

- All the edges of the graph G are of sufficient length to rule out collisions among vehicles that are neither on identical, nor on contiguous edges; that is, if d_{max} is the maximum sampling period of all the protectors under consideration, the length of each edge in the graph G is greater than $\Delta x_{max} = \dot{c}_{max} d_{max} - \dot{c}_{max}^2 / 2\ddot{c}_{brake}$ — the maximum distance a vehicle can travel if left free for d_{max} time units and instructed to brake thereafter, under the assumption that the vehicle does not collide and is abiding by the speed limit. This restriction rules out the possibility of a vehicle having a d_{max} time unit claim overlap with a vehicle that is more than one edge upstream or downstream.
- All the merges and diverges of the graph G are Y-shaped; that is, for each vertex v in the graph G, it is the case that $\langle |in(v)|, |out(v)| \rangle \in \{\langle 1, 1\rangle, \langle 2, 1\rangle, \langle 1, 2\rangle\}$.
- All cycles must contain at least three edges. This condition ensures that the ordering of the locations of vehicles traveling on successive branches of the track topology is well defined.

Any point on the graph G is represented by a pair consisting of the directed edge of the graph G and the distance of the particular point from the initial vertex of the directed edge. The formal definition of the set L of locations is as follows:

$$L = \{\langle e, x\rangle \mid e \in E \text{ and } x \in [0, length(e)]\}$$

The set of locations is constrained by the length of the edges of the graph G; that is, for $l \in L$ and $l = \langle e, x\rangle$, it is the case that $x \in [0, length(e)]$. We use the notation $l.e$ and $l.x$ to denote the edge and position components of the location l, respectively. It is important to note that, in this representation scheme, the vertices of the graph G have non-unique representations; that is, for all edges $e, e' \in E$, with $e.v_{final} = e'.v_{init}$, it is the case that the location $l = \langle e, length(e)\rangle$ is identical to the location $l' = \langle e', 0\rangle$. Finally, two locations in L are *comparable* if they are locations either on identical, or on successive edges, *i.e.*, the locations $l, l' \in L$ are comparable only if either $l.e = l'.e$, or $l.e.v_{final} = l'.e.v_{init}$ or $l.e.v_{init} = l'.e.v_{final}$.

Addition of a non-negative scalar y to a location $l \in L$, where $l = \langle e, x \rangle$, maps the location l to the set of locations that can be reached from the location l by traveling a distance y downstream. The set $\langle e, x \rangle + y$ always exists and is defined to be either the singleton $\{\langle e, x + y \rangle\}$, if $x + y \leq \mathit{length}(e)$, or the set $\bigcup_{\varepsilon \,\in\, out(e.v_{final})} (\langle \varepsilon, 0 \rangle + (x + y - \mathit{length}(e)))$, otherwise. This definition handles the cases in which the locations $\langle e, x \rangle + y$ extend past a single split or merge, or even multiple splits and/or merges in the track topology.

It is important to note that addition of a location l with a non-negative scalar that is bounded by the minimum distance from the location l to the closest second junction downstream results in a set of locations in which each location l' is comparable to the location l and satisfies the inequality $l \leq l'$; that is, for all $l \in L$, where $l = \langle e, x \rangle$, and $y \in [0, \mathit{length}(e) - x + \min_{\varepsilon \,\in\, out(e.v_{final})} \mathit{length}(\varepsilon)]$, the location l is comparable to all locations in $l + y$ and, moreover, $l \leq l'$, for all $l' \in l + y$. In particular, since the length of each edge of the graph G is assumed to be greater than Δx_{max}, addition of a location l with a non-negative scalar $y \leq \Delta x_{max}$ results in a set of locations in which each location l' is comparable to the location l and satisfies the inequality $l \leq l'$.

A closed interval in L is specified with an ordered pair of comparable locations and contains all locations between them, *e.g.*, $[\langle e_1, x_1 \rangle, \langle e_2, x_2 \rangle]$. The partial ordering on comparable locations in L is as follows: $\langle e_1, x_1 \rangle \leq \langle e_2, x_2 \rangle$ if and only if either $x_1 \leq x_2$ and $e_1 = e_2$, or $e_1.v_{final} = e_2.v_{init}$.

Due to the fact that the extent of a vehicle may extend beyond a split in the track topology, we redefine the notion of the section of the track occupied by a particular vehicle as the union of the intervals extending from the current position of the vehicle to a point on the track that is a distance c_{len} downstream; that is, the extent of a vehicle $i \in I$ is the set $E_i = \bigcup_{l'_i \,\in\, l_i + c_{len}} [l_i, l'_i]$.

In view of breaking the right-of-way symmetry when vehicles approach a merge in the track topology, we must define a prioritization scheme. In Chapter 7, the prioritization was based on the configuration of the merge; namely, the vehicle traveling on the `right` branch of the merge had priority over a vehicle traveling on the `left` branch. In the case of the graph of tracks, the notion of either `left`, or `right` is not well defined. Therefore, we associate a unique priority index to each edge of the graph and give priority to vehicles traveling on the edge whose priority index is greater. Let the function *priority* be an injection from the set of edges E of the graph G, to the set of natural numbers $\mathbb{N}$; that is, $\mathit{priority} : E \to \mathbb{N}$, where for any $e, e' \in E, e \neq e'$, it is the case that $\mathit{priority}(e) \neq \mathit{priority}(e')$.

The new model of the physical system, called GRAPH-VEHICLES, is presented in Figure 8.1. The GRAPH-VEHICLES automaton is the result of augmenting the MERGE-VEHICLES automaton of Chapter 7 so as to involve a general track topology consisting of Y-shaped merges and diverges. Each of the `reset-location`(i) actions, for $i \in I$, is enabled when the vehicle i has reached the final point of the directed edge on which it is traveling, *i.e.*, the vehicle i

Figure 8.1 The GRAPH-VEHICLES automaton.

Actions:

Input:
- e, the environment action (stuttering)
- $\texttt{protect}(C)_j$, for all $C \in \mathcal{P}(I), j \in J$

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

Variables

Internal:
- $\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$
- $\mathit{brake}(i) \in \texttt{Bool}$, for all $i \in I$, initially $\texttt{False}$
- $\mathit{brake\text{-}req}(i, j) \in \texttt{Bool}$, for all $i \in I, j \in J$, initially $\texttt{False}$

Output:
- $l_i \in L$, for all $i \in I$, initially $l_i \in L$
- $\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$
- $\mathit{collided}(i, i') \in \texttt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\texttt{False}$

subject to VALID

Discrete Transitions:

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

$\texttt{colliding-pair}(i, i')$
Pre: $\neg \mathit{collided}(i, i')$
  $\wedge (E_i \cap E_{i'} \neq \emptyset)$
  $\wedge (l_i < \min(E_i \cap E_{i'}))$
Eff: $\mathit{collided}(i, i') := \texttt{True}$
  if $(l_i.e \neq l_{i'}.e)$
    $\wedge (l_i.e.v_{\mathit{final}} = l_{i'}.e.v_{\mathit{final}})$
  then
    $\mathit{collided}(i', i) := \texttt{True}$

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

$\texttt{reset-location}(i)$
Pre: $l_i.x = \mathit{length}(l_i.e)$
Eff: $l_i.e :\in \mathit{out}(l_i.e)$
  $l_i.x := 0$

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

Trajectories:

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

is located on a vertex of the graph G. At that point in time, its location is nondeterministically set to the initial point of an arbitrary outgoing edge of the vertex on which the vehicle i is located.

The remaining state variables, derived variables, and discrete actions of either the VEHICLES automaton of Chapter 4, or the MERGE-VEHICLES automaton of Chapter 7 as well as the notational shorthand *collided*$(i, *)$, *collided*$(*, i)$, and *collided*$(*, i, *)$, for all $i \in I$, defined for the VEHICLES automaton in Section 4.1, carry over to the GRAPH-VEHICLES automaton unchanged.

As in the case of the MERGE-VEHICLES automaton, the set of input actions of the GRAPH-VEHICLES automaton includes the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(I)$ and $j \in J$; that is, the GRAPH-VEHICLES automaton allows each protector j, for $j \in J$, to brake any subset of the vehicles. However, it is often the case that a protector j, for some $j \in J$, need not schedule but a subset of the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(I)$. In such cases, the protector j is specified as having only the output actions that it is capable of scheduling and the remaining input actions of the GRAPH-VEHICLES automaton on port j are ignored.

The *VALID* set of the GRAPH-VEHICLES automaton is the redefinition of the *VALID* set of the VEHICLES automaton to account for the new track topology representation.

VALID $\subseteq$ *states*(GRAPH-VEHICLES), defined as the set of states of the GRAPH-VEHICLES automaton that satisfy the following conditions:

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

The GRAPH-VEHICLES automaton complies with the assumptions made about the *PP* automaton in Section 3.2.1. The GRAPH-VEHICLES automaton has neither input variables, nor output actions, on any of its ports (Axioms 3.2.1 and 3.2.2, respectively). Moreover, each of the actions $\texttt{protect}(C_j)_j$, for $j \in J$ and $C_j = \{i \mid \textit{brake-req}(i, j) = \texttt{True}\}$, is a no-op input action on port j for any $R \subseteq$ *VALID*. Therefore, the set of no-op input actions on each port $j \in J$ and any $R \subseteq$ *VALID* is non-empty (Axiom 3.2.3).

8.2 Auxiliary Derived Variables and Auxiliary Sets for the GRAPH-VEHICLES Automaton

In this section, we define auxiliary derived variables and sets for the GRAPH-VEHICLES automaton. Most of these variables and sets carry over from either the VEHICLES, or the MERGE-VEHICLES automata. In such cases, the variables and sets are redefined only when their extension to the GRAPH-VEHICLES automaton is not obvious.

As in Chapter 7, we assume that the variables $\textit{stop-dist}_i$, $\textit{max-range}_i(t)$, and $\textit{max-vel}_i(t)$, defined for the VEHICLES automaton in Section 4.3, extend to involve location instead of position variables in the obvious way.

As in the case of the extents of the vehicles of the GRAPH-VEHICLES automaton, we redefine the sections of track owned and claimed by the vehicles in the GRAPH-VEHICLES automaton. While their formal definitions appear in Table 8.1, their informal interpretations follow.

O_i, for $i \in I$, is the section of track that the vehicle i "owns". A vehicle i owns all track intervals that extend from the current position of the vehicle i to the points on the track that the vehicle i can reach even if it is braked immediately. Due to the possibility of such track intervals extending beyond a split in the track topology, the variable O_i is the union of all the intervals that the vehicle i owns.

$C_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, is the section of track that the vehicle i "claims" within t time units. A vehicle i claims within t time units all track intervals that extend from the current position of the vehicle i to the points on the track that the vehicle i can reach if braked after t time units and assuming worst-case vehicle behavior up to the point in time when it is braked. Due to the possibility of such track intervals extending beyond a split in the track topology, the variable $C_i(t)$ is the union of all the intervals that the vehicle i claims within t time units.

Henceforth, we assume that the sets $\textit{disjoint-extents}(i, i')$, $\textit{disjoint-owned-tracks}(i, i')$, and $\textit{disjoint-claimed-tracks}(i, i', t)$, for $i, i' \in I, i \neq i'$ and $t \in \mathbb{R}^{\geq 0}$, defined for the VEHICLES automaton in Section 4.3, have been extended to the GRAPH-VEHICLES automaton to incorporate the redefinitions of the derived variables used in their definitions. Moreover, we assume that the Lemmas 4.4.1, 4.4.2, 4.4.3, 4.4.4, and 4.4.5 extend to the GRAPH-VEHICLES automaton in the obvious way.

Several auxiliary sets for the GRAPH-VEHICLES automaton are described below. Their formal definitions appear in Table 8.2.

$\textit{successive}(i, i')$, for $i, i' \in I, i \neq i'$, is the subset of $\textit{VALID}$ that consists of the states in which the vehicles i and i' are traveling in succession either on the same, or on

Table 8.1 Auxiliary derived variables for the GRAPH-VEHICLES automaton.

$O_i \subseteq L$, for all $i \in I$, defined by

$$O_i = \bigcup_{l'_i \, \in \, l_i + (stop\text{-}dist_i + c_{len})} [l_i, l'_i]$$

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

$$C_i(t) = \bigcup_{l'_i \, \in \, l_i + (max\text{-}range_i(t) - max\text{-}vel_i(t)^2/(2\ddot{c}_{brake}) + c_{len})} [l_i, l'_i]$$

successive directed edges; that is, states in which either the vehicle i is downstream of the vehicle i', or the vehicle i' is downstream of the vehicle i.

adjacent(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* that consists of the states in which the vehicles i and i' are traveling on different tracks that lead to the same junction; that is, the edges on which the vehicles i and i' are traveling are distinct and have the same final vertex.

proximate(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* that consists of the states in which the vehicles i and i' are traveling either in succession as defined by the set *successive*(i, i'), or on adjacent tracks as defined by the set *adjacent*(i, i').

remote(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* that consists of the states in which the vehicles i and i' are traveling neither in succession as defined by the set *successive*(i, i'), nor on adjacent tracks as defined by the set *adjacent*(i, i').

yield-successive(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *successive*(i, i') that consists of the states in which, in the case of a claim overlap among the vehicles i and i', the vehicle i must yield to the vehicle i'. When the vehicles i and i' are traveling in succession, the vehicle i must yield to the vehicle i' if the vehicle i is trailing the vehicle i'. The vehicle i is said to be trailing the vehicle i' if the location of the vehicle i is strictly less than the location of the vehicle i'.

yield-adjacent(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *adjacent*(i, i') that consists of the states in which, in the case of a claim overlap among the vehicles i and i', the vehicle i must yield to the vehicle i'. When the vehicles i and i' are traveling on adjacent incoming tracks, the vehicle i must yield to the vehicle i' if either *only* the vehicle i' owns the upcoming merge point, or the vehicle i' has priority and neither or both vehicles own the merge point.

yield(i, i'), for $i, i' \in I, i \neq i'$, is the subset of *VALID* that consists of the states in which,

Table 8.2 Auxiliary sets for the GRAPH-VEHICLES automaton.

$successive(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$\begin{aligned} successive(i,i') = \{p \in VALID \mid\ & (p.l_i.e = p.l_{i'}.e) \\ & \vee (p.l_i.e.v_{final} = p.l_{i'}.e.v_{init}) \\ & \vee (p.l_{i'}.e.v_{final} = p.l_i.e.v_{init})\} \end{aligned}$$

$adjacent(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$adjacent(i,i') = \{p \in VALID \mid (p.l_i.e \neq p.l_{i'}.e) \wedge (p.l_i.e.v_{final} = p.l_{i'}.e.v_{final})\}$$

$proximate(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$proximate(i,i') = successive(i,i') \cup adjacent(i,i')$$

$remote(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$remote(i,i') = VALID - proximate(i,i')$$

$yield\text{-}successive(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$yield\text{-}successive(i,i') = \{p \in successive(i,i') \mid p.l_i < p.l_{i'}\}$$

$yield\text{-}adjacent(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$\begin{aligned} yield\text{-}adjacent(i,i') = \{p \in adjacent(i,i') \mid\ & ((\langle p.l_i.e, length(p.l_i.e)\rangle \notin p.O_i \\ & \quad \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \in p.O_{i'}) \\ & \vee ((\langle p.l_i.e, length(p.l_i.e)\rangle \in p.O_i \\ & \quad \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \in p.O_{i'} \\ & \quad \wedge priority(p.l_i.e) < priority(p.l_{i'}.e)) \\ & \vee ((\langle p.l_i.e, length(p.l_i.e)\rangle \notin p.O_i \\ & \quad \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \notin p.O_{i'} \\ & \quad \wedge priority(p.l_i.e) < priority(p.l_{i'}.e))\} \end{aligned}$$

$yield(i,i') \subseteq VALID$, for $i,i' \in I, i \neq i'$, defined by

$$yield(i,i') = yield\text{-}successive(i,i') \cup yield\text{-}adjacent(i,i')$$

in the case of a claim overlap among the vehicles i and i', the vehicle i must yield to the vehicle i' in order to prevent a potential collision between the vehicles i and i'.

The following lemma describes some properties of the sets defined above.

Lemma 8.2.1 *For all $i, i' \in I, i \neq i'$, the following hold:*

1. *$VALID = proximate(i, i') \cup remote(i, i')$.*
2. *$proximate(i, i') \cap remote(i, i') = \emptyset$.*
3. *$successive(i, i') = yield\text{-}successive(i, i') \cup yield\text{-}successive(i', i)$.*
4. *$yield\text{-}successive(i, i') \cap yield\text{-}successive(i', i) = \emptyset$.*
5. *$adjacent(i, i') = yield\text{-}adjacent(i, i') \cup yield\text{-}adjacent(i', i)$.*
6. *$yield\text{-}adjacent(i, i') \cap yield\text{-}adjacent(i', i) = \emptyset$.*

Proof: We prove each of the conditions separately:

1. The condition that $VALID = proximate(i, i') \cup remote(i, i')$, for each $i, i' \in I, i \neq i'$, follows from the definition of the sets $proximate(i, i')$ and $remote(i, i')$.
2. As for the first condition, the condition that $proximate(i, i') \cap remote(i, i') = \emptyset$, for each $i, i' \in I, i \neq i'$, follows from the definition of the sets $proximate(i, i')$ and $remote(i, i')$.
3. For all $i, i' \in I, i \neq i'$, the sets $yield\text{-}successive(i, i')$ and $yield\text{-}successive(i', i)$ are both subsets of the set $successive(i, i')$. Therefore, it suffices to show that any state p in the set $successive(i, i')$, for some $i, i' \in I, i \neq i'$, is either in the set $yield\text{-}successive(i, i')$, or in the set $yield\text{-}successive(i', i)$.

   Let the state p be any state in $successive(i, i')$, for some $i, i' \in I, i \neq i'$. Since $successive(i, i') \subseteq VALID$, it is the case that $p \in VALID$. Therefore, the sections of the track occupied by the vehicles i and i' do not have a positive length closed interval overlap. It follows that it is not possible for their locations to coincide; that is, for any $p \in successive(i, i')$, it is the case that $p.l_i \neq p.l_{i'}$. Therefore, regarding the ordering of the locations of the vehicles i and i', there are only two viable cases:

   (a) $p.l_i < p.l_{i'}$. In this case, $p \in yield\text{-}successive(i, i')$.

   (b) $p.l_{i'} < p.l_i$. In this case, $p \in yield\text{-}successive(i', i)$.

4. If $p \in yield\text{-}successive(i, i')$ then it is the case that $p.l_i < p.l_{i'}$. It follows that $p \notin yield\text{-}successive(i', i)$. Similarly, if $p \in yield\text{-}successive(i', i)$ then it is the case that $p.l_{i'} < p.l_i$. It follows that $p \notin yield\text{-}successive(i, i')$. This suffices.

5. For all $i, i' \in I, i \neq i'$, the sets *yield-adjacent*(i, i') and *yield-adjacent*(i', i) are both subsets of the set *adjacent*(i, i'). Therefore, it suffices to show that any state p in the set *adjacent*(i, i'), for some $i, i' \in I, i \neq i'$, is either in the set *yield-adjacent*(i, i'), or in the set *yield-adjacent*(i', i).

   Let the state p be any state in *adjacent*(i, i'), for some $i, i' \in I, i \neq i'$, and without loss of generality let the vehicle i' be the vehicle traveling on the incoming edge of greater priority, *i.e.*, $priority(p.l_i.e) < priority(p.l_{i'}.e)$. Regarding the ownership of the merge point by each of the vehicles, there are four cases:

   (a) $\langle \mathtt{out}, 0\rangle \in p.O_i \wedge \langle \mathtt{out}, 0\rangle \in p.O_{i'}$. In this case, $p \in$ *yield-adjacent*(i, i') and $p \notin$ *yield-adjacent*(i', i).

   (b) $\langle \mathtt{out}, 0\rangle \notin p.O_i \wedge \langle \mathtt{out}, 0\rangle \notin p.O_{i'}$. Similarly to above, $p \in$ *yield-adjacent*(i, i') and $p \notin$ *yield-adjacent*(i', i).

   (c) $\langle \mathtt{out}, 0\rangle \notin p.O_i \wedge \langle \mathtt{out}, 0\rangle \in p.O_{i'}$. In this case, $p \in$ *yield-adjacent*(i, i') and $p \notin$ *yield-adjacent*(i', i).

   (d) $\langle \mathtt{out}, 0\rangle \in p.O_i \wedge \langle \mathtt{out}, 0\rangle \notin p.O_{i'}$. In this case, $p \notin$ *yield-adjacent*(i, i') and $p \in$ *yield-adjacent*(i', i).

6. This condition follows from the analysis in the proof of condition 5.

■

8.3 Protection System GRAPH-PROT-PAIR$_{\{i,i'\}}$

The GRAPH-PROT-PAIR$_{\{i,i'\}}$ automata, for $i, i' \in I, i \neq i'$, are vehicle-pair collision protectors and guarantee 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. Each of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, is an implementation of the abstract protector of Section 3.2 specialized to particular definitions of the parameters PP, S, R, G, j, and d.

The physical plant automaton, PP, is defined to be the GRAPH-VEHICLES automaton of Figure 8.1. The port j and the sampling period d are defined to be the port and sampling period with which the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ communicates with the GRAPH-VEHICLES automaton. While the port j is assumed arbitrary, the sampling period d is restricted to the set $(0, d_{max}]$, where d_{max} is the maximum protector sampling period presented in Section 8.1. The set of "good" states G is defined to be the set of states in which the vehicles i and i' have not collided into each other, *i.e.*, $G = VALID - P_{collided(i,i')} - P_{collided(i',i)}$. In this chapter, we use the notation $G_{\{i,i'\}}$ to denote the definition of G that is specific to the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protector. The set R is defined to be the set $R = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. This definition restricts the states of the GRAPH-VEHICLES automaton to states in which all the vehicles are abiding by the speed limit and in which the vehicles of all other vehicle pairs $\{i'', i'''\}$, for $i'', i''' \in I, i'' \neq i''', \{i'', i'''\} \neq \{i, i'\}$, have not collided into each other. The set S is defined to be the set *safe* defined in Section 3.2.1; that is, the set of states of the PP automaton for which a single input action of PP on port j can guarantee that, provided no new input actions on port j are allowed, all subsequently R-reachable states will be in G. Once again, the definition of the set *safe* is specialized to the above definitions of the automaton PP, the sets R and G, and the port j. In this chapter, we use the notation $R_{\{i,i'\}}$ and $S_{\{i,i'\}}$ to refer to the above definitions of the sets R and S.

The GRAPH-PROT-PAIR$_{\{i,i'\}}$ protector automaton is an implementation of the abstract protector automaton Abs(GRAPH-VEHICLES, $S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d$). More precisely, as is the case for the abstract protector Abs_j, we define the GRAPH-PROT-PAIR$_{\{i,i'\}}$ automaton to be the composition of a sensor and a discrete controller automaton. These automata are implementations of their abstract equivalents of Figures 3.2 and 3.3 specialized, however, to the above definitions of the parameters PP, S, R, G, j, and d. The sensor automaton is precisely the specialization of the sensor automaton of Figure 3.2 to the above definitions of the parameters PP, etc. The discrete controller automaton is defined in Figure 8.2.

The braking strategy of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protector is as follows. The protector is allowed to brake the vehicles i and i' only if the sections of the track they claim in d 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.

It is important to note that the abstract protector automaton Abs(GRAPH-VEHICLES, $S_{\{i,i'\}}$, $R_{\{i,i'\}}, G_{\{i,i'\}}, j, d$) complies with the assumptions made about the abstract protector in Section 3.2.1. In particular, since the vehicle location variables, the vehicle velocity variables, and the *collided* variables are output variables of the GRAPH-VEHICLES automaton, the set *safe* is $Y_{\text{GRAPH-VEHICLES}}$-determinable and actions that guarantee safety can be determined from the output variables of the GRAPH-VEHICLES automaton (Axioms 3.2.4 and 3.2.5, respectively). Moreover, the sets $R_{\{i,i'\}}$ and $G_{\{i,i'\}}$ are $Y_{\text{GRAPH-VEHICLES}}$-determinable (Axioms 3.2.6 and 3.2.7, respectively) and the set of start states $S_{\{i,i'\}}$ is a subset of the set *safe* (Axiom 3.2.8), since $S_{\{i,i'\}}$ is defined to be the set *safe*.

In Section 3.1 it was shown that the abstract protector Abs_j guarantees that the physical

Figure 8.2 Discrete controller automaton for the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$.

Actions: Input: e, the environment action (stuttering)
`snapshot`$(y)_j$, for each valuation y of $Y_{\text{GRAPH-VEHICLES}}$
Output: `protect`$(C)_j$, for $C \in \mathcal{P}(\{i,i'\})$
Variables: Internal: $send_j \in \mathcal{P}(\{i,i'\}) \cup null$, initially $null$

Discrete Transitions:

`snapshot`$(y)_j$
Eff: if $y \notin disjoint\text{-}claimed\text{-}tracks(i, i', d)$ then
if $y \in yield(i, i')$ then
$send_j := \{i\}$
else
$send_j := \{i'\}$
else
$send_j := \emptyset$

`protect`$(C)_j$
Pre: $send_j = C$
Eff: $send_j := null$

Trajectories:
$w.send_j \equiv null$

---

plant PP remains within G starting from S given R. Similarly, the GRAPH-PROT-PAIR$_{\{i,i'\}}$ automaton guarantees that the GRAPH-VEHICLES automaton remains within $G_{\{i,i'\}}$ starting from $S_{\{i,i'\}}$ given $R_{\{i,i'\}}$. This is shown in the following section.

8.4 Correctness of GRAPH-PROT-PAIR$_{\{i,i'\}}$

The main result to be shown is that GRAPH-PROT-PAIR$_{\{i,i'\}} \leq Abs($GRAPH-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. Since both GRAPH-PROT-PAIR$_{\{i,i'\}}$ and $Abs($GRAPH-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$ involve the composition of the same sensor automaton with distinct discrete controller automata, Theorem 2.7.4 applies. Therefore, it suffices to show that the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ of Figure 8.2 implements the discrete controller automaton $DC($GRAPH-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$ of Figure 3.3. From Theorem 2.6.1, this follows by showing that there exists a simulation relation between the states of the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$

Table 8.3 Sets used in the correctness proof of GRAPH-PROT-PAIR$_{\{i,i'\}}$.

$W_{\{i,i'\}} \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$$

$B_{\{i,i'\}} \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$B_{\{i,i'\}} = W_{\{i,i'\}} \cap P_{B_{ij}} \cap P_{B_{i'j}}$$

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

$$V_{(i,i')} = \{p \in W_{\{i,i'\}} \cap P_{B_{ij}} \mid (p \in \mathit{successive}(i, i') \wedge p.l_i < p.l_{i'}) \\ \vee (p \in \mathit{adjacent}(i, i') \wedge p.O_i \subseteq [p.l_i, \langle p.l_i.e, \mathit{length}(p.l_i.e)\rangle])\}$$

$V_{\{i,i'\}} \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$$

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

$$T_{\{i,i'\}}(t) = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}claimed\text{-}tracks}(i, i', t)$$

and the discrete controller automaton $DC($GRAPH-VEHICLES$, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. We first give some set definitions, then prove some lemmas, and finally show the existence of such a simulation relation.

In this section, we use the notation $\mathit{future}_{\{i,i'\}}$, $\mathit{safe}_{\{i,i'\}}$, $\mathit{very\text{-}safe}_{\{i,i'\}}$, and $\mathit{delay\text{-}safe}_{\{i,i'\}}$ to denote the specialization of the function *future*, the sets *safe* and *very-safe*, and the function *delay-safe*, which are defined in Section 3.2.1, to the automaton GRAPH-VEHICLES, the sets $R_{\{i,i'\}}$ and $G_{\{i,i'\}}$, and the the port j of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protector. Moreover, since the environment action of the GRAPH-VEHICLES automaton is stuttering, its consideration is omitted in all inductive proofs involving the PP automaton.

We proceed by defining several sets that are used in the correctness proof of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$. For reference, their formal definitions appear in Table 8.3.

Let $W_{\{i,i'\}}$ be the subset of $R_{\{i,i'\}} \cap G_{\{i,i'\}}$ comprised of the states in which the section of the track owned by the vehicle i does not overlap the section of track owned by the vehicle i'; that is, $W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$.

Let $B_{\{i,i'\}}$ be the subset of of $W_{\{i,i'\}}$ comprised of the states in which the vehicles i and i' are both being instructed to brake by the protector j; that is, $B_{\{i,i'\}} = W_{\{i,i'\}} \cap P_{B_{ij}} \cap P_{B_{i'j}}$.

Let $V_{(i,i')}$ be the subset of $W_{\{i,i'\}}$ comprised of the states in which the vehicle i is being instructed to brake by the protector j and either the vehicles i and i' are traveling in succession and $l_i < l_{i'}$, *i.e.*, the vehicle i is trailing the vehicle i', or the vehicles i and i' are adjacent and the section of the track owned by the vehicle i is entirely upstream of the merge point $\langle \mathtt{out}, 0\rangle$. Moreover, let $V_{\{i,i'\}}$ be defined as $V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$.

Let $T_{\{i,i'\}}(t)$, where $t \in \mathbb{R}^{\geq 0}$, be the subset of $R_{\{i,i'\}} \cap G_{\{i,i'\}}$ comprised of the states in which the section of the track claimed in time t by the vehicle i does not overlap the section of the track claimed in time t by the vehicle i'; that is, $T_{\{i,i'\}}(t) = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-claimed-tracks*(i, i', t).

The following lemma defines the relation among the sets $G_{\{i,i'\}}$, $W_{\{i,i'\}}$, $B_{\{i,i'\}}$, $V_{\{i,i'\}}$, and $T_{\{i,i'\}}(t)$, for $t \in \mathbb{R}^{\geq 0}$.

Lemma 8.4.1 *For all $t, t' \in \mathbb{R}^{\geq 0}$, $t \leq t'$, the following hold:*

1. $T_{\{i,i'\}}(t) \subseteq W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$.
2. $V_{\{i,i'\}} \subseteq W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$.
3. $B_{\{i,i'\}} \subseteq W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$.
4. $T_{\{i,i'\}}(t') \subseteq T_{\{i,i'\}}(t)$.
5. $T_{\{i,i'\}}(0) = W_{\{i,i'\}}$.

Proof: Follow directly from the definitions of the sets $W_{\{i,i'\}}$, $B_{\{i,i'\}}$, $V_{\{i,i'\}}$, and $T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and Lemma 4.4.2. ■

In the next two lemmas, we show that any state p in the set $B_{\{i,i'\}}$ is in the set *very-safe*$_{\{i,i'\}}$; that is, any state $R_{\{i,i'\}}$-reachable from p through an execution fragment that involves no input actions on port j, is in the set $G_{\{i,i'\}}$. In the first lemma, we show that any state that is $R_{\{i,i'\}}$-reachable from p through an execution fragment that involves no input actions on port j, is in the set $W_{\{i,i'\}}$. In the second lemma, we show that $B_{\{i,i'\}} \subseteq$ *very-safe*$_{\{i,i'\}}$.

Lemma 8.4.2 *future*$_{\{i,i'\}}(B_{\{i,i'\}}, \mathbb{R}^{\geq 0}) \subseteq W_{\{i,i'\}}$.

Proof: Let α be an execution fragment of the GRAPH-VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $B_{\{i,i'\}}$, is only comprised of states in $R_{\{i,i'\}}$, and involves no input actions on port j. Let p_{init} and p_{final} be the initial and final states of α, respectively. By induction on the length n of the execution fragment α, we show that $p_{final} \in W_{\{i,i'\}}$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$. Since $p_{init} \in B_{\{i,i'\}}$, Lemma 8.4.1, part 3, implies that $p_{final} \in W_{\{i,i'\}}$.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{final} \in W_{\{i,i'\}}$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{final} \in W_{\{i,i'\}}$. Since the final state of α is reached from the final state of α' by a single step, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, we consider all possible discrete actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$, are not enabled because α involves no input actions on port j.

2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and affects neither the velocity of the vehicle i', nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i)$ action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_i \leq p'_{final}.\dot{x}_i$. Moreover, since the $\texttt{brick-wall}(i)$ action does not affect the velocity of the vehicle i', it is the case that $p_{final}.\dot{x}_{i'} = p'_{final}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_i \subseteq p'_{final}.O_i$ and $p_{final}.O_{i'} \subseteq p'_{final}.O_{i'}$. Therefore, since $p'_{final} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it follows that $p_{final} \in$ *disjoint-owned-tracks*(i, i').

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{final} \in W_{\{i,i'\}}$.

3. the $\texttt{brick-wall}(i')$ action sets the velocity of the vehicle i' to zero and affects neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i')$ action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{final}.\dot{x}_{i'} \leq p'_{final}.\dot{x}_{i'}$. Moreover, since the $\texttt{brick-wall}(i')$ action does not affect the velocity of the vehicle i, it is the case that $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_{i'} \subseteq p'_{final}.O_{i'}$ and $p_{final}.O_i \subseteq p'_{final}.O_i$. Therefore, since $p'_{final} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it follows that $p_{final} \in$ *disjoint-owned-tracks*(i, i').

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{final} \in W_{\{i,i'\}}$.

4. the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\texttt{reset-location}(i''')$, for $i''' \in I$, affect neither the velocities of the vehicles i and i', nor the *collided*(i, i') and *collided*(i', i) variables.

From the induction hypothesis, it is the case that $p'_{final} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\texttt{reset-location}(i''')$, for $i''' \in I$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{final} \in G_{\{i,i'\}}$.

Moreover, since the input actions $\texttt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions $\texttt{brick-wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\texttt{reset-location}(i''')$, for $i''' \in I$, do not affect the velocities of the vehicles i and i', it is the case that $p_{final}.\dot{x}_i = p'_{final}.\dot{x}_i$ and $p_{final}.\dot{x}_{i'} = p'_{final}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{final}.O_i \subseteq p'_{final}.O_i$ and $p_{final}.O_{i'} \subseteq p'_{final}.O_{i'}$. Therefore, since $p'_{final} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it is the case that $p_{final} \in$ *disjoint-owned-tracks*(i, i').

Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{final} \in W_{\{i,i'\}}$.

5. the internal actions $\texttt{colliding-pair}(i'', i''')$, for $i'', i''' \in I, i'' \neq i'''$, and the internal actions $\texttt{collision-effects}(i'''')$, for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{final} \in W_{\{i,i'\}}$.

Since, $p_{init} \in B_{(i,i')} \subseteq P_{B_{ij}} \cap P_{B_{i'_j}}$ and the execution fragment leading from p_{init} to p'_{final} involves no input actions on port j, it follows that $p'_{final} \in P_{B_{ij}} \cap P_{B_{i'_j}}$. Therefore, in the case of a trajectory from p'_{final} to p_{final}, Lemma 4.4.4, part 1, implies that $p_{final}.O_i \subseteq p'_{final}.O_i$ and $p_{final}.O_{i'} \subseteq p'_{final}.O_{i'}$. Therefore, since $p'_{final} \in W_{\{i,i'\}} \subseteq$ *disjoint-owned-tracks*(i, i'), it follows that $p_{final} \in$ *disjoint-owned-tracks*(i, i'). Moreover, since $p'_{final} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$ and the variables *collided*(i, i') and *collided*(i', i) remain constant throughout the trajectory, it follows that $p_{final} \in G_{\{i,i'\}}$. Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{final} \in W_{\{i,i'\}}$, as needed. ■

Lemma 8.4.3 $B_{\{i,i'\}} \subseteq$ *very-safe*$_{\{i,i'\}}$.

Proof: Follows directly from Lemma 8.4.2, Lemma 8.4.1, part 3, and the definition of *very-safe* in Section 3.2.1. ■

In the next three lemmas and the subsequent corollary, we show that the sets $W_{\{i,i'\}}$ and *safe*$_{\{i,i'\}}$ are equal. First, we show that any state that is $R_{\{i,i'\}}$-reachable from a state p in $W_{\{i,i'\}}$ through an execution fragment that involves no input actions on port j and has a limit time equal to zero, is in the set $W_{\{i,i'\}}$. Then, we show that $W_{\{i,i'\}} \subseteq$ *safe*$_{\{i,i'\}}$ and *safe*$_{\{i,i'\}} \subseteq W_{\{i,i'\}}$. Finally, the subsequent corollary states that $W_{\{i,i'\}} =$ *safe*$_{\{i,i'\}}$.

Lemma 8.4.4 *future*$_{\{i,i'\}}(W_{\{i,i'\}}, 0) \subseteq W_{\{i,i'\}}$.

Proof: Let α be an execution fragment of the GRAPH-VEHICLES automaton of n steps, where $n \in \mathbb{N}$, that: starts in a state in $W_{\{i,i'\}}$, is only comprised of states in $R_{\{i,i'\}}$, involves no input actions on port j, and has a limit time equal to zero. Let p_{init} and p_{final} be the initial and final states of α, respectively. By induction on the length n of the execution fragment α, we show that $p_{\mathit{final}} \in W_{\{i,i'\}}$.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of no steps and, therefore, $p_{\mathit{final}} = p_{init}$. Since $p_{init} \in W_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{\mathit{final}} \in W_{\{i,i'\}}$. Let α' be the part of the execution fragment α comprised of the first k steps. The induction hypothesis involves the assertion that if p'_{final} is the final state of α', then it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}}$. Since the final state of α is reached from the final state of α' by a single step, the inductive step involves the consideration of all possible steps leading from p'_{final} to p_{final}.

To complete the induction, we consider all possible discrete actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i,i'\})$, are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and affects neither the velocity of the vehicle i', nor the $\mathit{collided}(i,i')$ and $\mathit{collided}(i',i)$ variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i)$ action does not affect the $\mathit{collided}(i,i')$ and $\mathit{collided}(i',i)$ variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. Moreover, since the $\texttt{brick-wall}(i)$ action does not affect the velocity of the vehicle i', it is the case that $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$ and $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq \mathit{disjoint\text{-}owned\text{-}tracks}(i,i')$, it follows that $p_{\mathit{final}} \in \mathit{disjoint\text{-}owned\text{-}tracks}(i,i')$.

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.
3. the $\texttt{brick-wall}(i')$ action sets the velocity of the vehicle i' to zero and affects neither the velocity of the vehicle i, nor the $\mathit{collided}(i,i')$ and $\mathit{collided}(i',i)$ variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i')$ action does not affect the $\mathit{collided}(i,i')$ and $\mathit{collided}(i',i)$ variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_{i'} \leq p'_{\mathit{final}}.\dot{x}_{i'}$. Moreover, since the $\texttt{brick-wall}(i')$ action does not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$ and $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$, it follows that $p_{\mathit{final}} \in \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$.

Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.

4. the actions $\mathtt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, $\mathtt{brick\text{-}wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\mathtt{reset\text{-}location}(i''')$, for $i''' \in I$, affect neither the velocities of the vehicles i and i', nor the $\mathit{collided}(i, i')$ and $\mathit{collided}(i', i)$ variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Therefore, since the actions $\mathtt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, $\mathtt{brick\text{-}wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\mathtt{reset\text{-}location}(i''')$, for $i''' \in I$, do not affect the $\mathit{collided}(i, i')$ and $\mathit{collided}(i', i)$ variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Moreover, since the input actions $\mathtt{protect}(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions $\mathtt{brick\text{-}wall}(i'')$, for $i'' \in I - \{i, i'\}$, and $\mathtt{reset\text{-}location}(i''')$, for $i''' \in I$, do not affect the velocities of the vehicles i and i', it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$ and $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$ and $p_{\mathit{final}}.O_{i'} \subseteq p'_{\mathit{final}}.O_{i'}$. Therefore, since $p'_{\mathit{final}} \in W_{\{i,i'\}} \subseteq \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$, it is the case that $p_{\mathit{final}} \in \mathit{disjoint\text{-}owned\text{-}tracks}(i, i')$.

   Finally, since all states in α are, by definition, restricted to the set $R_{\{i,i'\}}$, it follows that $p_{\mathit{final}} \in W_{\{i,i'\}}$.

5. the internal actions $\mathtt{colliding\text{-}pair}(i'', i''')$, for $i'', i''' \in I, i'' \neq i'''$, and the internal actions $\mathtt{collision\text{-}effects}(i'''')$, for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{\mathit{final}} \in W_{\{i,i'\}}$.

■

Lemma 8.4.5 $W_{\{i,i'\}} \subseteq \mathit{safe}_{\{i,i'\}}$.

Proof: From the definition of *safe* in Section 3.2.1, we must show that any state $p \in W_{\{i,i'\}}$ satisfies: (i) $\mathit{future}_{\{i,i'\}}(p, 0) \subseteq G_{\{i,i'\}}$, and (ii) there exists some input action π on port j such that for every $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in \mathit{future}_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in \mathit{very\text{-}safe}_{\{i,i'\}}$.

(i) Since $p \in W_{\{i,i'\}}$, the first condition follows from Lemma 8.4.4 and Lemma 8.4.1, part 1.

(ii) For the second condition, let π be the action $\mathtt{protect}(\{i, i'\})_j$.

From Lemma 8.4.4, it follows that $p' \in W_{\{i,i'\}}$. Now, considering the step from p' to p'', since the $\mathtt{protect}(\{i, i'\})_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_{\{i,i'\}}$, $p'' \in G_{\{i,i'\}}$, $p''.\dot{x}_i = p'.\dot{x}_i$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'}$. Therefore, Lemma 4.4.3, part 1, implies that $p''.O_i \subseteq p'.O_i$ and $p''.O_{i'} \subseteq p'.O_{i'}$. Since

$p' \in W_{\{i,i'\}}$, it follows that $p'' \in$ *disjoint-owned-tracks*(i, i'). From the above conditions, it follows that $p'' \in W_{\{i,i'\}}$.

Moreover, since the `protect`$(\{i, i'\})_j$ action sets the internal variables *brake-req*(i, j) and *brake-req*(i', j) to `True`, it is the case that $p'' \in P_{B_{ij}} \cap P_{B_{i'j}}$. Since $p'' \in W_{\{i,i'\}}$, it follows that $p'' \in B_{\{i,i'\}}$.

Finally, Lemma 8.4.3 implies that $p'' \in$ *very-safe*$_{\{i,i'\}}$, as needed. ■

Lemma 8.4.6 *For any $p \in R_{\{i,i'\}}$, if $p \in$ safe$_{\{i,i'\}}$ then $p \in W_{\{i,i'\}}$.*

Proof: We show the contrapositive; that is, for any $p \in R_{\{i,i'\}}$, if $p \notin W_{\{i,i'\}}$ then $p \notin$ *safe*$_{\{i,i'\}}$. Since $W_{\{i,i'\}} = R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-owned-tracks*(i, i') and $p \in R_{\{i,i'\}}$, we consider the conditions $p \notin G_{\{i,i'\}}$ and $p \notin$ *disjoint-owned-tracks*(i, i') separately.

1. $p \notin G_{\{i,i'\}}$.

   From Lemma 3.2.4, part 1, it is the case that *safe*$_{\{i,i'\}} \subseteq G_{\{i,i'\}}$. Since $p \notin G_{\{i,i'\}}$, it follows that $p \notin$ *safe*$_{\{i,i'\}}$.

2. $p \notin$ *disjoint-owned-tracks*(i, i').

   We must show that $p \notin$ *safe*$_{\{i,i'\}}$. In order for the state $p \in R_{\{i,i'\}}$ to be in the set *safe*$_{\{i,i'\}}$ there must exist some input action π on port j such that for every $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in$ *very-safe*$_{\{i,i'\}}$. Therefore, it suffices to show that for any input action π on port j, there exist $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin$ *very-safe*$_{\{i,i'\}}$.

   Using similar analyses to those presented in the proofs of Lemmas 6.2.9 and 7.4.10, it can be shown that for any $p \in R_{\{i,i'\}}$ and any input action π on port j, there exist $p', p'' \in R_{\{i,i'\}}$ satisfying $p' \in$ *future*$_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, such that $p'' \notin$ *very-safe*$_{\{i,i'\}}$. It follows that $p \notin$ *safe*$_{\{i,i'\}}$, as needed.

■

Corollary 8.4.7 $W_{\{i,i'\}} =$ *safe*$_{\{i,i'\}}$.

Proof: Follows directly from Lemmas 8.4.5 and 8.4.6. ■

In the following three lemmas, we show that any state $R_{\{i,i'\}}$-reachable from a state in $V_{(i,i')}$ through an execution fragment that involves no input actions on port j and has a limit time that lies in the interval $[0, d_{max}]$, is in the set $W_{\{i,i'\}}$. In the first lemma, we show that if the final state of such an execution fragment is in $G_{\{i,i'\}}$ and the section of track owned by the vehicle i has not grown since the beginning of the execution fragment, then the final state of the execution fragment is in $W_{\{i,i'\}}$. In the second lemma, we show that the final state of any such execution fragment is in $G_{\{i,i'\}}$ and the section of track owned by the vehicle i does not grow throughout the execution fragment. The third lemma states the desired property which follows directly from the first two lemmas.

Lemma 8.4.8 *Let $p \in V_{(i,i')}$ and $p' \in future_{\{i,i'\}}(p,[0,d_{max}])$. If $p' \in G_{\{i,i'\}}$ and $p'.O_i \subseteq p.O_i$, then $p' \in W_{\{i,i'\}}$.*

Proof: We need to show that $p' \in W_{\{i,i'\}}$; that is, we need to show that the state p' is in the set $R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-owned-tracks*(i,i'). Since $p' \in G_{\{i,i'\}}$, by assumption, it remains to be shown that $p' \in R_{\{i,i'\}}$ and $p' \in$ *disjoint-owned-tracks*(i,i'). We consider these two conditions by cases:

1. $p' \in R_{\{i,i'\}}$.

   This is the case because the function $future_{\{i,i'\}}(p, \mathbb{R}^{\geq 0})$ only considers $R_{\{i,i'\}}$-reachable states.

2. $p' \in$ *disjoint-owned-tracks*(i,i').

   Since $p \in V_{(i,i')}$, there are two possible scenarios: (i) $p \in$ *successive*(i,i') and $p.l_i < p.l_{i'}$, (ii) $p \in$ *adjacent*(i,i') and $p.O_i \subseteq [p.l_i, \langle p.l_i.e, length(p.l_i.e)\rangle]$.

   In the first case, it is as if the vehicle i is trailing the vehicle i' on a single track. Since $p \in V_{(i,i')} \subseteq W_{\{i,i'\}}$, the sections of the track owned by the vehicles i and i' in state p are disjoint. Now, consider the section of track owned by the vehicle i in the state p'. Since $p'.O_i \subseteq p.O_i$, it follows that $p.l_i = \min(p.O_i) \leq p'.l_i = \min(p'.O_i)$ and there exist locations in $p.O_i$ that are at least as downstream as any of the locations in $p'.O_i$. Next, consider the section of track owned by the vehicle i' in the state p'. Because of the non-negative constraint on the vehicle velocities it follows that the location $p'.l_{i'} = \min(p'.O_{i'})$ is either equal to, or downstream of the location $p.l_{i'} = \min(p.O_{i'})$. Moreover, the sections of track owned by the vehicle i' in state p' could only range from the location $p'.l_{i'}$ up to the locations that are a distance Δx_{max} downstream from the location $p.l_{i'}$. Therefore, because of the constraint on the length of the edges in the track topology and the constraint on the minimum number of edges comprising a cycle in the track topology, it follows that $p' \in$ *disjoint-owned-tracks*(i,i').

   In the second case, since $p.O_i \subseteq [p.l_i, \langle p.l_i.e, length(p.l_i.e)\rangle]$, the section of the track owned by the vehicle i in state p is strictly within the incoming directed edge $p.l_i.e$. Since $p'.O_i \subseteq p.O_i$, the same is true for the section of track owned by the vehicle i in state p'. Similarly to above, because of the non-negative constraint on the vehicle velocities it follows that the location $p'.l_{i'} = \min(p'.O_{i'})$ is either equal to, or downstream of the location $p.l_{i'} = \min(p.O_{i'})$. Moreover, the sections of track owned by the vehicle i' in state p' could only range from the location $p'.l_{i'}$ up to the locations that are a distance Δx_{max} downstream from the location $p.l_{i'}$. Therefore, because of the constraint on the length of the edges in the track topology, the constraint on the minimum number of edges comprising a cycle in the track topology, the fact that the vehicles are traveling on adjacent tracks in state p, and the fact that the section of track owned by the vehicle i remains within the incoming branch, it follows that $p' \in$ *disjoint-owned-tracks*(i, i').

■

Lemma 8.4.9 *If $p \in V_{(i,i')}$ and $p' \in$ future$_{\{i,i'\}}(p, [0, d_{max}])$, then it is the case that $p' \in G_{\{i,i'\}}$ and $p'.O_i \subseteq p.O_i$.*

Proof: Let α be an execution fragment of the GRAPH-VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $V_{(i,i')}$, is only comprised of states in $R_{\{i,i'\}}$, involves no input actions on port j, and has a limit time that lies in the interval $[0, d_{max}]$. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{final} \in G_{\{i,i'\}}$ and $p_{final}.O_i \subseteq p_{init}.O_i$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{final} = p_{init}$. From Lemma 8.4.1, part 2, and the fact that $p_{init} \in V_{(i,i')} \subseteq V_{\{i,i'\}}$, it follows that $p_{final} \in G_{\{i,i'\}}$. Moreover, the fact that $p_{final}.O_i \subseteq p_{init}.O_i$ is trivially true.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, then $p_{final} \in G_{\{i,i'\}}$ and $p_{final}.O_i \subseteq p_{init}.O_i$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{final} \in G_{\{i,i'\}}$ and $p'_{final}.O_i \subseteq p'_{init}.O_i$. Moreover, from Lemma 8.4.8 it follows that $p'_{final} \in W_{\{i,i'\}}$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, we consider all possible actions by cases:

1. the actions `protect`$(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$, are not enabled because α involves no input actions on port j.
2. the `brick-wall`(i) action sets the velocity of the vehicle i to zero and does not affect the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{final} \in G_{\{i,i'\}}$. Therefore, since the `brick-wall`(i) action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

Moreover, since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$, as needed.

3. the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, `brick-wall`(i''), for $i'' \in I, i'' \neq i$, and `reset-location`(i'''), for $i''' \in I$, affect neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_{\{i,i'\}}$. Therefore, since the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, `brick-wall`(i''), for $i'' \in I, i'' \neq i$, and `reset-location`(i'''), for $i''' \in I$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Moreover, since the input actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions `brick-wall`(i''), for $i'' \in I, i'' \neq i$ and `reset-location`(i'''), for $i''' \in I$, do not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 1, it follows that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{init}.O_i$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$, as needed.

4. the internal actions `colliding-pair`(i'', i'''), for $i'', i''' \in I, i'' \neq i'''$, and the internal actions `collision-effects`(i''''), for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{\mathit{final}} \in W_{\{i,i'\}}$.

Since $p'_{init} \in V_{(i,i')} \subseteq P_{B_{ij}}$ and the execution fragment leading from p'_{init} to p'_{final} involves no input actions on port j, it follows that $p'_{\mathit{final}} \in P_{B_{ij}}$. Therefore, in the case of a trajectory from p'_{final} to p_{final}, Lemma 4.4.4, part 1, implies that $p_{\mathit{final}}.O_i \subseteq p'_{\mathit{final}}.O_i$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.O_i \subseteq p'_{init}.O_i$. Since $p_{init} = p'_{init}$, it follows that $p_{\mathit{final}}.O_i \subseteq p_{init}.O_i$. Moreover, since $p'_{\mathit{final}} \in G_{\{i,i'\}}$ and the variables *collided*(i, i') and *collided*(i', i) remain constant throughout the trajectory, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$, as needed. ■

Lemma 8.4.10 $\mathit{future}_{\{i,i'\}}(V_{(i,i')}, [0, d_{max}]) \subseteq W_{\{i,i'\}}$.

Proof: Follows directly from Lemmas 8.4.8 and 8.4.9. ■

In the following lemma, we extend the result of Lemma 8.4.10 to the set $V_{\{i,i'\}}$.

Lemma 8.4.11 $\mathit{future}_{\{i,i'\}}(V_{\{i,i'\}}, [0, d_{max}]) \subseteq W_{\{i,i'\}}$.

Proof: Follows directly from Lemma 8.4.10 and the fact that $V_{\{i,i'\}} = V_{(i,i')} \cup V_{(i',i)}$. ■

The following lemma states that any state p in the set $V_{\{i,i'\}}$ is in the set *delay-safe*$_{\{i,i'\}}(d)$; that is, any state $R_{\{i,i'\}}$-reachable from p within an amount of time d through an execution fragment that involves no input actions on port j, is in the set $G_{\{i,i'\}}$ and any state $R_{\{i,i'\}}$-reachable from p in exactly an amount of time d through an execution fragment that involves no input actions on port j, is in the set *safe*$_{\{i,i'\}}$.

Lemma 8.4.12 $V_{\{i,i'\}} \subseteq$ *delay-safe*$_{\{i,i'\}}(d)$.

Proof: Follows from Lemma 3.2.4, part 1, Lemma 8.4.11, Corollary 8.4.7, and the fact that $d \leq d_{max}$. ■

In the next few lemmas, we show that any state p in the set $T_{\{i,i'\}}(t)$, for any $t \in \mathbb{R}^{\geq 0}$, is in the set *delay-safe*$_{\{i,i'\}}(t)$; that is, any state $R_{\{i,i'\}}$-reachable from p within an amount of time t through an execution fragment that involves no input actions on port j, is in the set $G_{\{i,i'\}}$ and any state $R_{\{i,i'\}}$-reachable from p in exactly an amount of time t through an execution fragment that involves no input actions on port j, is in the set *safe*$_{\{i,i'\}}$.

Lemma 8.4.13 *Let $p \in T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in$ future$_{\{i,i'\}}(p,t)$, where $t \in [0,\tau]$. If $p' \in G_{\{i,i'\}}$, $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$, and $p'.C_{i'}(\tau - t) \subseteq p.C_{i'}(\tau)$, then $p' \in T_{\{i,i'\}}(\tau - t)$.*

Proof: We need to show that $p' \in R_{\{i,i'\}} \cap G_{\{i,i'\}} \cap$ *disjoint-claimed-tracks*$(i, i', \tau - t)$. Since $p' \in G_{\{i,i'\}}$, by assumption, it remains to be shown that $p' \in R_{\{i,i'\}}$ and $p' \in$ *disjoint-claimed-tracks*$(i, i', \tau - t)$. We consider these two conditions by cases:

1. $p' \in R_{\{i,i'\}}$.

   This is the case because the function *future*$_{\{i,i'\}}(p,t)$ only considers $R_{\{i,i'\}}$-reachable states.

2. $p' \in$ *disjoint-claimed-tracks*$(i, i', \tau - t)$.

   Since $p \in T_{\{i,i'\}}(\tau)$, it is the case that $p \in$ *disjoint-claimed-tracks*(i, i', τ). Therefore, since $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$ and $p'.C_{i'}(\tau - t) \subseteq p.C_{i'}(\tau)$, it follows that $p' \in$ *disjoint-claimed-tracks*$(i, i', \tau - t)$, as needed.

■

Lemma 8.4.14 *For all $p \in T_{\{i,i'\}}(\tau)$, where $\tau \in \mathbb{R}^{\geq 0}$, and $p' \in$ future$_{\{i,i'\}}(p,t)$, where $t \in [0,\tau]$, it is the case that $p' \in G_{\{i,i'\}}$, $p'.C_i(\tau - t) \subseteq p.C_i(\tau)$, and $p'.C_{i'}(\tau - t) \subseteq p.C_{i'}(\tau)$.*

Proof: Let $\tau \in \mathbb{R}^{\geq 0}$ and α be an execution fragment of the GRAPH-VEHICLES automaton of n steps and trajectories, where $n \in \mathbb{N}$, that: starts in a state in $T_{\{i,i'\}}(\tau)$, is only comprised of states in $R_{\{i,i'\}}$, involves no input actions on port j, and has a limit time t that lies in the interval $[0, \tau]$. Letting p_{init} and p_{final} be the initial and final states of α, respectively, we must show that $p_{\mathit{final}} \in G_{\{i,i'\}}$, $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$, and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$. The proof is by induction on the length n of the execution fragment α.

For the base case, consider the execution fragment α of length $n = 0$; that is, α is an execution fragment that consists of a single point trajectory and, therefore, $p_{\mathit{final}} = p_{init}$ and $\alpha.\mathit{ltime} = 0$, *i.e.*, $t = 0$. From Lemma 8.4.1, part 1, and the fact that $p_{init} \in T_{\{i,i'\}}(\tau)$, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$. Moreover, since $t = 0$, the facts that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$ are trivially true.

The inductive step involves showing that if α is an execution fragment of length $n = k+1$, for some $k \in \mathbb{N}$, with $\alpha.\mathit{ltime} = t$, where $t \in [0, \tau]$, then $p_{\mathit{final}} \in G_{\{i,i'\}}$, $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$, and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$. Let α' be the part of the execution fragment α comprised of the first k steps and trajectories and let $\alpha'.\mathit{ltime} = t'$, where $t' \in [0, t]$. The induction hypothesis involves the assertion that if p'_{init} and p'_{final} are the initial and final states of α', respectively, then it is the case that $p'_{\mathit{final}} \in G_{\{i,i'\}}$, $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$, and $p'_{\mathit{final}}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Moreover, from Lemma 8.4.13 it follows that $p'_{\mathit{final}} \in T_{\{i,i'\}}(\tau - t')$. Since the final state of α is reached from the final state of α' by a single step or trajectory, the inductive step involves the consideration of all possible steps and trajectories leading from p'_{final} to p_{final}.

In the case of a step, keeping in mind that the limit times of α' and α are equal, *i.e.*, $t' = t$, we consider all possible discrete actions by cases:

1. the actions $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$, are not enabled because α involves no input actions on port j.
2. the $\texttt{brick-wall}(i)$ action sets the velocity of the vehicle i to zero and affects neither the velocity of the vehicle i', nor the $\mathit{collided}(i, i')$ and $\mathit{collided}(i', i)$ variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_{\{i,i'\}}$. Therefore, since the $\texttt{brick-wall}(i)$ action does not affect the $\mathit{collided}(i, i')$ and $\mathit{collided}(i', i)$ variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_i \leq p'_{\mathit{final}}.\dot{x}_i$. Moreover, since the $\texttt{brick-wall}(i)$ action does not affect the velocity of the vehicle i', it is the case that $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 2, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p'_{\mathit{final}}.C_{i'}(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{init}.C_i(\tau)$ and $p'_{\mathit{final}}.C_{i'}(\tau - t') \subseteq p'_{init}.C_{i'}(\tau)$. Therefore, since $p_{init} = p'_{init}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{init}.C_i(\tau)$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{init}.C_{i'}(\tau)$, as needed.

3. the `brick-wall`(i') action sets the velocity of the vehicle i' to zero and affects neither the velocity of the vehicle i, nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_{\{i,i'\}}$. Therefore, since the `brick-wall`(i') action does not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Since the vehicle velocities are restricted to be non-negative, it is the case that $p_{\mathit{final}}.\dot{x}_{i'} \leq p'_{\mathit{final}}.\dot{x}_{i'}$. Moreover, since the `brick-wall`(i') action does not affect the velocity of the vehicle i, it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$. From Lemma 4.4.3, part 2, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p'_{\mathit{final}}.C_{i'}(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$ and $p'_{\mathit{final}}.C_{i'}(\tau - t') \subseteq p'_{\mathit{init}}.C_{i'}(\tau)$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{\mathit{init}}.C_i(\tau)$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{\mathit{init}}.C_{i'}(\tau)$, as needed.

4. the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, and `reset-location`(i'''), for $i''' \in I$, affect neither the velocities of the vehicles i and i', nor the *collided*(i, i') and *collided*(i', i) variables.

   From the induction hypothesis, it is the case that $p'_{\mathit{final}} \in G_{\{i,i'\}}$. Therefore, since the actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, and `reset-location`(i'''), for $i''' \in I$, do not affect the *collided*(i, i') and *collided*(i', i) variables, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$.

   Moreover, since the input actions `protect`$(C)_{j'}$, for $C \in \mathcal{P}(I)$ and $j' \in J, j' \neq j$, and the internal actions `brick-wall`(i''), for $i'' \in I - \{i, i'\}$, and `reset-location`(i'''), for $i''' \in I$, do not affect the velocities of the vehicles i and i', it is the case that $p_{\mathit{final}}.\dot{x}_i = p'_{\mathit{final}}.\dot{x}_i$ and $p_{\mathit{final}}.\dot{x}_{i'} = p'_{\mathit{final}}.\dot{x}_{i'}$. From Lemma 4.4.3, part 2, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p'_{\mathit{final}}.C_{i'}(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$ and $p'_{\mathit{final}}.C_{i'}(\tau - t') \subseteq p'_{\mathit{init}}.C_{i'}(\tau)$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{\mathit{init}}.C_i(\tau)$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{\mathit{init}}.C_{i'}(\tau)$, as needed.

5. the internal actions `colliding-pair`(i'', i'''), for $i'', i''' \in I, i'' \neq i'''$, and the internal actions `collision-effects`(i''''), for $i'''' \in I$, are not enabled because α is only comprised of states in $R_{\{i,i'\}}$ and $p'_{\mathit{final}} \in T_{\{i,i'\}}(\tau - t')$.

In the case of a trajectory, Lemma 4.4.4, part 2, implies that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p'_{\mathit{final}}.C_i(\tau - t')$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p'_{\mathit{final}}.C_{i'}(\tau - t')$. However, from the induction hypothesis it is the case that $p'_{\mathit{final}}.C_i(\tau - t') \subseteq p'_{\mathit{init}}.C_i(\tau)$ and $p'_{\mathit{final}}.C_{i'}(\tau - t') \subseteq p'_{\mathit{init}}.C_{i'}(\tau)$. Therefore, since $p_{\mathit{init}} = p'_{\mathit{init}}$, it follows that $p_{\mathit{final}}.C_i(\tau - t) \subseteq p_{\mathit{init}}.C_i(\tau)$ and $p_{\mathit{final}}.C_{i'}(\tau - t) \subseteq p_{\mathit{init}}.C_{i'}(\tau)$. Moreover, since $p'_{\mathit{final}} \in G_{\{i,i'\}}$ and the *collided*(i, i') and *collided*(i', i) variables remain constant throughout the trajectory, it follows that $p_{\mathit{final}} \in G_{\{i,i'\}}$, as needed. ■

Lemma 8.4.15 *For $\tau \in \mathbb{R}^{\geq 0}$ and $t \in [0, \tau]$, it is the case that $future_{\{i,i'\}}(T_{\{i,i'\}}(\tau), t) \subseteq T_{\{i,i'\}}(\tau - t)$.*

Proof: Follows directly from Lemmas 8.4.13 and 8.4.14. ■

Corollary 8.4.16 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $future_{\{i,i'\}}(T_{\{i,i'\}}(t), 0) \subseteq T_{\{i,i'\}}(t)$.*

Proof: Follows directly from Lemma 8.4.15. ■

Lemma 8.4.17 *For any $t \in \mathbb{R}^{\geq 0}$, it is the case that $T_{\{i,i'\}}(t) \subseteq delay\text{-}safe_{\{i,i'\}}(t)$.*

Proof: From the definition of *delay-safe* in Section 3.2.1, we must show that:

1. $future_{\{i,i'\}}(T_{\{i,i'\}}(t), [0, t]) \subseteq G_{\{i,i'\}}$, and
2. $future_{\{i,i'\}}(T_{\{i,i'\}}(t), t) \subseteq safe_{\{i,i'\}}$.

The first condition follows directly from Lemma 8.4.15 and Lemma 8.4.1, part 1. Moreover, Lemma 8.4.15 and Lemma 8.4.1, part 5, imply that $future_{\{i,i'\}}(T_{\{i,i'\}}(t), t) \subseteq W_{\{i,i'\}}$. Therefore, the second condition follows from Lemma 8.4.5. ■

In the following lemma, we show that the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ implements the protector $Abs(\text{GRAPH-VEHICLES}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$. Since the protector automata GRAPH-PROT-PAIR$_{\{i,i'\}}$ and Abs_j involve the composition of the same sensor automaton with distinct controller automata, it suffices to show that the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ implements the discrete controller automaton $DC(\text{GRAPH-VEHICLES}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$.

Lemma 8.4.18 GRAPH-PROT-PAIR$_{\{i,i'\}} \leq Abs(\text{GRAPH-VEHICLES}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j, d)$.

Proof: Both the GRAPH-PROT-PAIR$_{\{i,i'\}}$ and the Abs_j protectors involve the composition of the same sensor automaton with distinct controller automata. From Theorem 2.7.4, it suffices to show that the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ implements DC_j. This is shown by a simulation from the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ to DC_j.

The mapping between the states of the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ and DC_j is almost the identity. In the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$, the variable $send_j$ is equal to either a member of $\mathcal{P}(\{i, i'\})$, or the value *null*. In DC_j, these valuations simply map to either the actions $\texttt{protect}(C)_j$, where

C is the member of $\mathcal{P}(\{i, i'\})$ that corresponds to the the valuation of the variable $send_j$ of the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$, or the value *null*, respectively.

The start states for the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ and DC_j are the states in which $send_j = null$. These are related to each other according to the mapping discussed above.

Furthermore, since the trajectories in both discrete controller automata are identical, we need only consider their discrete transitions. We analyze the actions of the implementation by cases, letting p denote any complete state of the GRAPH-VEHICLES automaton that corresponds to the output state y, *i.e.*, $p \in VALID$ and $p \lceil Y_{\text{GRAPH-VEHICLES}} = y$.

1. The $\texttt{snapshot}(y)_j$ action of the implementation sets $send_j$ to an element of $\mathcal{P}(\{i, i'\})$. In order to show that the behavior of the implementation is allowed by the specification, we must show that the input action $\texttt{snapshot}(y)_j$ of the implementation sets the value of the $send_j$ variable in such a way that the subsequently enabled action π of the implementation (i) guarantees that for all $p', p'' \in R_{\{i,i'\}}$ such that $p' \in future_{\{i,i'\}}(p, 0)$ and $p' \xrightarrow{\pi} p''$, it is the case that $p'' \in delay\text{-}safe_{\{i,i'\}}(d)$, if $p \in safe_{\{i,i'\}}$, and (ii) is an arbitrary output action of the implementation, otherwise.

   First, consider the case in which $p \in safe_{\{i,i'\}}$. Since Corollary 8.4.7 implies that $p \in W_{\{i,i'\}}$, the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ according to whether the state p is in $T_{\{i,i'\}}(d)$, or not.

   On one hand, if $p \notin T_{\{i,i'\}}(d)$ then the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to either $\{i\}$, or $\{i'\}$ according to the strategy described in Section 8.3. Therefore, the $\texttt{snapshot}(y)_j$ action enables either the $\texttt{protect}(\{i\})_j$ action, or the $\texttt{protect}(\{i'\})_j$ action. Since $p \in W_{\{i,i'\}}$, Lemma 8.4.4 implies that $p' \in W_{\{i,i'\}}$. Moreover, since the $\texttt{protect}(\{i\})_j$ and $\texttt{protect}(\{i'\})_j$ actions affect neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_{\{i,i'\}}$, $p'' \in G_{\{i,i'\}}$, $p''.\dot{x}_i = p'.\dot{x}_i$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'}$. Therefore, since $p' \in W_{\{i,i'\}}$, Lemma 4.4.3, part 1, implies that $p'' \in disjoint\text{-}owned\text{-}tracks(i, i')$. From the above conditions, it follows that $p'' \in W_{\{i,i'\}}$. Moreover, since the $\texttt{protect}(\{i\})_j$ and $\texttt{protect}(\{i'\})_j$ actions set the $brake\text{-}req(i, j)$ and $brake\text{-}req(i', j)$ variables, respectively, to $\texttt{True}$, it follows that $p'' \in V_{\{i,i'\}}$. Finally, Lemma 8.4.12 implies that $p'' \in delay\text{-}safe_{\{i,i'\}}(d)$, as needed.

   On the other hand, if $p \in T_{\{i,i'\}}(d)$ then the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to $\emptyset$ and the $\texttt{protect}(\emptyset)_j$ action is enabled. Since $p \in T_{\{i,i'\}}(d)$, Corollary 8.4.16 implies that $p' \in T_{\{i,i'\}}(d)$. Moreover, since the $\texttt{protect}(\emptyset)_j$ action affects neither the velocity of any of the vehicles, nor any of the *collided* variables, it follows that $p'' \in R_{\{i,i'\}}$, $p'' \in G_{\{i,i'\}}$, $p''.\dot{x}_i = p'.\dot{x}_i$, and $p''.\dot{x}_{i'} = p'.\dot{x}_{i'}$. Therefore, since $p' \in T_{\{i,i'\}}(d)$, Lemma 4.4.3, part 2, implies that $p'' \in$ *disjoint-claimed-tracks*(i, i', d). From the above conditions, it follows that $p'' \in T_{\{i,i'\}}(d)$. Finally, Lemma 8.4.17 implies that $p'' \in$ *delay-safe*$_{\{i,i'\}}(d)$, as needed.

Next, consider the case in which $p \notin$ *safe*$_{\{i,i'\}}$. In this case, the $\texttt{snapshot}(y)_j$ action of the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ sets the variable $send_j$ to either $\{i\}$, $\{i'\}$, or $\emptyset$ and, subsequently, enables either the $\texttt{protect}(\{i\})_j$ action, the $\texttt{protect}(\{i'\})_j$ action, or the $\texttt{protect}(\emptyset)_j$ action, respectively. However, when $p \notin$ *safe*$_{\{i,i'\}}$, the DC_j automaton sets the variable $send_j$ arbitrarily and, subsequently, enables an arbitrary output action. Therefore, the behavior of the discrete controller automaton of the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$ is allowed by that of the DC_j automaton.

Therefore, the effects of the $\texttt{snapshot}(y)_j$ action of the implementation are allowed by its specification.

2. The $\texttt{protect}(C)_j$ actions, for $C \in \mathcal{P}(\{i, i'\})$, have identical effects in both discrete controller automata. When the $send_j$ variable matches either the set C, or the $\texttt{protect}(C)_j$ action, the action $\texttt{protect}(C)_j$ is executed and the $send_j$ variable is set to *null* in both discrete controller automata.

3. The environment action in both discrete controller automata is stuttering. It follows that the mapping between the states of the discrete controller automaton of GRAPH-PROT-PAIR$_{\{i,i'\}}$ and the DC_j automaton prior to and succeeding the execution of the environment action remains the same.

■

Corollary 8.4.19 *The protector* GRAPH-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the automaton* GRAPH-VEHICLES *remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Proof: Follows directly from Lemma 8.4.18 and Theorem 3.2.9. ■

8.5 Protection System GRAPH-PROT

We now define the collision protector GRAPH-PROT. While considering the automaton GRAPH-PROT, we restrict the states of the GRAPH-VEHICLES automaton to $P_{not\text{-}overspeed}$ as defined in Section 4.2, *i.e.*, $R_{\text{GRAPH-PROT}} = P_{not\text{-}overspeed}$. Let $G_{\text{GRAPH-PROT}}$ and $S_{\text{GRAPH-PROT}}$ be the intersection of $G_{\{i,i'\}}$ and $S_{\{i,i'\}}$, for all $\{i, i'\}$, where $i, i' \in I, i \neq i'$, respectively, and GRAPH-PROT be the composition of GRAPH-PROT-PAIR$_{\{i,i'\}}$, for all $\{i, i'\}$, where $i, i' \in I, i \neq i'$. The protector GRAPH-PROT guarantees that GRAPH-VEHICLES remains within $G_{\text{GRAPH-PROT}}$ starting from $S_{\text{GRAPH-PROT}}$ given $R_{\text{GRAPH-PROT}}$. For reference, the formal definitions of the GRAPH-PROT automaton and the sets $G_{\text{GRAPH-PROT}}$, $S_{\text{GRAPH-PROT}}$, and $R_{\text{GRAPH-PROT}}$ are shown in Table 8.4.

Table 8.4 Formal definitions of GRAPH-PROT, $G_{\text{GRAPH-PROT}}$, $S_{\text{GRAPH-PROT}}$, and $R_{\text{GRAPH-PROT}}$.

$$\text{GRAPH-PROT} \equiv \prod_{i,i' \in I, i \neq i'} \text{GRAPH-PROT-PAIR}_{\{i,i'\}}$$

$$G_{\text{GRAPH-PROT}} \equiv \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$$

$$S_{\text{GRAPH-PROT}} \equiv \bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$$

$$R_{\text{GRAPH-PROT}} \equiv P_{\textit{not-overspeed}}$$

Lemma 8.5.1 *The protector* GRAPH-PROT *guarantees that the* GRAPH-VEHICLES *automaton remains within* $G_{\text{GRAPH-PROT}}$ *starting from* $S_{\text{GRAPH-PROT}}$ *given* $R_{\text{GRAPH-PROT}}$.

In the following proof, we show that all the states of an execution of $PP \times$ GRAPH-PROT starting from $S_{\text{GRAPH-PROT}}$ given $R_{\text{GRAPH-PROT}}$ are in $G_{\text{GRAPH-PROT}}$. This is done by applying Theorem 3.1.8 and showing that the second condition of the theorem does not hold.

Proof: Let α be any execution of the system $PP \times$ GRAPH-PROT starting from a state in $S_{\text{GRAPH-PROT}}$ and in which all states are in $R_{\text{GRAPH-PROT}}$.

From Theorem 3.1.8, one of the following holds:

1. Every state in α is in $G_{\text{GRAPH-PROT}} = \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$.
2. α can be written as $\alpha_1 \frown \alpha_2$, where
   (a) All state occurrences in α_1 except possibly the last state occurrence are in the set $G_{\text{GRAPH-PROT}} = \bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$.
   (b) If the last state occurrence in α_1 is in $\overline{G_{\{i,i'\}}}$, for some $i, i' \in I, i \neq i'$, then there exists $i'', i''' \in I, i'' \neq i''', \{i'', i'''\} \neq \{i, i'\}$, such that the last state occurrence in α_1 is in $\overline{G_{\{i'',i'''\}}}$.
   (c) All state occurrences in α_2 except possibly the first state occurrence are in the set $\bigcap_{\{i'',i'''\} \in N} \textit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$, for some $N \subseteq \{\{i, i'\} \mid i, i' \in I, i \neq i'\}$, where $|N| \geq 2$.

We proceed by showing that it is not possible to decompose α as $\alpha_1 \frown \alpha_2$ while satisfying the three aforementioned conditions.

The violation of $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ can only occur through the violation of at least one of the conditions $G_{\{i,i'\}}$, where $i, i' \in I, i \neq i'$. Moreover, each of these conditions are violated only through the execution of a `colliding-pair` action. Without loss of generality, suppose that the first condition that is violated in α is the condition $G_{\{i,i'\}}$, for some $i, i' \in I, i \neq i'$, and that such a violation has resulted through a $\texttt{colliding-pair}(i, i')$ action. Let p and p' be the states of the system prior to and succeeding this $\texttt{colliding-pair}(i, i')$ action, *i.e.*, $p, p' \in R_{\text{GRAPH-PROT}}$ such that $p \xrightarrow{\pi} p'$, where $\pi = \texttt{colliding-pair}(i, i')$. Since the $\texttt{colliding-pair}(i, i')$ action only sets the $\mathit{collided}(i, i')$ variable to `True`, it follows that the state p' is in the set $\overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. Now, we attempt to decompose α as $\alpha_1 \frown \alpha_2$:

1. Suppose we split α at any state preceding the state p. Then the state p is in α_2. Since p' is the first state in which one of the conditions $G_{\{i'',i'''\}}$, for $i'', i''' \in I, i'' \neq i'''$, is violated, it is the case that $p \in \bigcap_{i'',i''' \in I, i'' \neq i'''} G_{\{i'',i'''\}}$ and there does not exist $N \subseteq \{\{i'', i'''\} \mid i'', i''' \in I, i'' \neq i'''\}$ such that $|N| \geq 2$ and $p \in \bigcap_{\{i'',i'''\} \in N} \mathit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

2. Suppose we split α at the state p. Then the state p' is in α_2. Since p' is the first state in which one of the conditions $G_{\{i'',i'''\}}$, for $i'', i''' \in I, i'' \neq i'''$, is violated and since the state p' is in $\overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$, it follows that there does not exist $N \subseteq \{\{i'', i'''\} \mid i'', i''' \in I, i'' \neq i'''\}$ such that $|N| \geq 2$ and $p' \in \bigcap_{\{i'',i'''\} \in N} \mathit{past}(\overline{G_{\{i'',i'''\}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

3. Suppose we split α at the state p'. Then p' is the last state of α_1 and the first state of α_2. However, $p' \in \overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$. Therefore, the second condition is violated and this decomposition of α is not valid.

4. Suppose we split α at any state succeeding p'. Then the state p' is in α_1. Since $p' \in \overline{G_{\{i,i'\}}} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$, it follows that the state p' is not in the set $\bigcap_{i'',i''' \in I, i'' \neq i'''} G_{\{i'',i'''\}}$. Therefore, the first condition is violated and this decomposition of α is not valid.

Therefore, the execution α cannot be decomposed into any such α_1 and α_2. It follows that the first clause of Theorem 3.1.8 must hold; that is, every state in α is in $G_{\text{GRAPH-PROT}}$. This implies that the protector GRAPH-PROT guarantees $G_{\text{GRAPH-PROT}}$ in the GRAPH-VEHICLES automaton starting from $S_{\text{GRAPH-PROT}}$ given $R_{\text{GRAPH-PROT}}$. ■

Chapter 9

Composing the Overspeed and Collision Avoidance Protection Systems

In the previous chapters, we presented example protectors whose correct operation required that the physical plant automaton at hand satisfied particular properties. For instance, in the case of the VEHICLES automaton of Chapter 4, the overspeed protector OS-PROT of Chapter 5 assumes that none of the vehicles collide among themselves and the collision protector CL-PROT of Chapter 6 assumes that none of the vehicles exceed the speed limit. Similarly, the MERGE-PROT protector of Chapter 7 and the GRAPH-PROT protector of Chapter 8 guarantee that none of the vehicles collide among themselves in the MERGE-VEHICLES and GRAPH-VEHICLES automata, respectively, provided that all the vehicles are abiding by the speed limit. In this chapter, we compose the overspeed and collision protectors for the VEHICLES automaton and show that the resulting protector guarantees that the vehicles in the VEHICLES automaton neither exceed the speed limit, nor collide among themselves. We extend these results to the MERGE-VEHICLES and GRAPH-VEHICLES automata after assuming that the overspeed protector OS-PROT of Chapter 5 extends, virtually unchanged, to the MERGE-VEHICLES and GRAPH-VEHICLES automata.

9.1 Overspeed and Collision Avoidance for the VEHICLES Automaton

In the following lemma, we show that the composition of the protectors OS-PROT and CL-PROT guarantees that the vehicles in the VEHICLES automaton neither exceed the speed limit, nor collide among themselves.

Lemma 9.1.1 *The composition of* OS-PROT *and* CL-PROT *is a protector that guarantees* $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$ *in the* VEHICLES *automaton starting from* $S_{\text{OS-PROT}} \cap S_{\text{CL-PROT}}$.

In the following proof, we show that all the states of an execution of $PP \times$OS-PROT$\times$CL-PROT starting from $S_{\text{OS-PROT}} \cap S_{\text{CL-PROT}}$ are in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$. This is done by applying Theorem 3.1.7 and showing that the second condition of the theorem does not hold.

Proof: Let α be any execution of the system $PP \times$ OS-PROT $\times$ CL-PROT starting from a state in $S_{\text{OS-PROT}} \cap S_{\text{CL-PROT}}$.

From Theorem 3.1.7, one of the following holds:

1. Every state in α is in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$.
2. α can be written as $\alpha_1 \frown \alpha_2$, where
   (a) All state occurrences in α_1 except possibly the last are in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$.
   (b) The last state occurrence in α_1 is in G_i, for some $i \in \{\text{OS-PROT}, \text{CL-PROT}\}$, if and only it is in $G_{i'}$, for $i' \in \{\text{OS-PROT}, \text{CL-PROT}\}, i' \neq i$.
   (c) All state occurrences in α_2 except possibly the first state occurrence of α_2 are in $past(\overline{G_{\text{OS-PROT}}}, \alpha) \cap past(\overline{G_{\text{CL-PROT}}}, \alpha)$.

We proceed by showing that it is not possible to decompose α into α_1 and α_2 as proposed by the second clause of Theorem 3.1.7. Then it trivially follows that the first clause of Theorem 3.1.7 holds; that is, for any such α, all states are in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$.

The violation of $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$ can occur through the violation of either $G_{\text{OS-PROT}}$, or $G_{\text{CL-PROT}}$. On one hand, provided that no collisions have occurred, the violation of $G_{\text{OS-PROT}}$ can only occur within a trajectory of the VEHICLES automaton. On the other hand, the violation of $G_{\text{CL-PROT}}$ can only occur through the execution of a `colliding-pair`(i, i') action, for some $i, i' \in I, i' \neq i$. We analyze each of these cases separately.

1. In the first case, the key point is that the violation of the speed limit by any of the vehicles in the VEHICLES automaton can only occur within a trajectory and that a collision can not be recorded within a trajectory. Therefore, the fact that the speed limit is violated prior to the occurrence of any vehicle collisions would imply that the OS-PROT protector is not working correctly; that is, Corollary 5.3.1 is false.

   Let w be the first trajectory in α containing a state occurrence in $\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}$. Suppose that w is a T_I-trajectory and let T'_I be the subset of T_I consisting of all t such that $(i, t, w(t)) \in past(\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}, \alpha)$. Then T'_I is a non-empty subinterval of T_I that is "upward-closed", *i.e.*, if $t \in T'_I$, $t' \in T_I$, and $t < t'$, then $t' \in T'_I$. Since T'_I is an interval of reals, it has a left endpoint t which might or might not itself be in T'_I. It is important to note that since the *collided*(i, i') variables, for $i, i' \in I, i \neq i'$, remain constant throughout any trajectory of the VEHICLES automaton, it is only possible to violate the $G_{\text{OS-PROT}}$ condition within the trajectory w; that is, all the states in w are in the set $G_{\text{CL-PROT}}$. Therefore, letting $s = w(t)$, all state occurrences in α that precede the state occurrence (i, t, s) are in the set $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$. Now, we attempt to decompose α into α_1 and α_2.

(a) Suppose we split α at any state preceding the state (i, t, s). Then the state (i, t, s) is in α_2. Since (i, t, s) is in $G_{\text{CL-PROT}}$ and all states that precede the state (i, t, s) are in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$, it is the case that $(i, t, s) \notin \mathit{past}(\overline{G_{\text{OS-PROT}}}, \alpha) \cap \mathit{past}(\overline{G_{\text{CL-PROT}}}, \alpha)$. Since the state (i, t, s) is not the first state in α_2, the third condition is violated. Therefore, this decomposition of α is not valid.

(b) Suppose we split α at the state (i, t, s) and suppose that the state (i, t, s) is not the last state of w. Then any state of the trajectory w that succeeds the state (i, t, s) is in α_2. Moreover, since all of the states in w are in $G_{\text{CL-PROT}}$, none of the states in w that succeed (i, t, s) are in $\mathit{past}(\overline{G_{\text{OS-PROT}}}, \alpha) \cap \mathit{past}(\overline{G_{\text{CL-PROT}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

(c) Suppose we split α at the state (i, t, s) and suppose that the state (i, t, s) is the last state of w. Then since w is the first trajectory in α containing an occurrence of a state in $\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}$, it follows that $(i, t, s) \in \overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}$. Moreover, since all the states in w are in $G_{\text{CL-PROT}}$, it is the case that $(i, t, s) \in \overline{G_{\text{OS-PROT}}} \cap G_{\text{CL-PROT}}$. Therefore, the second condition is violated and this decomposition of α is not valid.

(d) Suppose we split α at a state s'' that succeeds the state (i, t, s). Let (i, t', s') be a state of the trajectory w that succeeds the state (i, t, s) and precedes the state s''. The state (i, t', s') is in α_1. By definition of T'_I, it is the case that (i, t', s') is in $\mathit{past}(\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}, \alpha)$. Therefore, the first condition is violated and this decomposition of α is not valid.

2. In the second case, the key point is that a collision can only be recorded by an action and that such an action can not cause the velocity of a vehicle to exceed the speed limit. Therefore, the fact that a collision among the vehicles occurs prior to the violation of the speed limit would imply that the CL-PROT protector is not working correctly, *i.e.*, Lemma 6.3.1 is false.

   Without loss of generality, suppose that the $G_{\text{CL-PROT}}$ condition is violated through a `colliding-pair`(i, i') action, for some $i, i' \in I, i' \neq i$. Let p and p' be the states of the system prior to and succeeding this `colliding-pair`(i, i') action, *i.e.*, $p, p' \in \mathit{VALID}$ such that $p \xrightarrow{\pi} p'$, where $\pi =$ `colliding-pair`(i, i'). Since the `colliding-pair`(i, i') action only sets the *collided*(i, i') variable to `True`, it follows that the state p' is in the set $G_{\text{OS-PROT}} \bigcap \overline{G_{\text{CL-PROT}}}$. Now, we attempt to decompose α into α_1 and α_2.

(a) Suppose we split α at any state preceding the state p. Then the state p is in α_2. Since p' is the first state in $\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}$, it is the case that all the states of α preceding p' are in the set $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$; that is, $p \notin \mathit{past}(\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}, \alpha)$ and $p \notin \mathit{past}(\overline{G_{\text{OS-PROT}}}, \alpha) \cap \mathit{past}(\overline{G_{\text{CL-PROT}}}, \alpha)$. Since p is not the first state in α_2, the third condition is violated. Therefore, this decomposition of α is not valid.

(b) Suppose we split α at the state p. Then the state p' is in α_2. Since p' is the first state in $\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}$, it is the case that all the states of α preceding p' are in the set $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$; that is, $p \notin \mathit{past}(\overline{G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}}, \alpha)$ and, moreover, $p \notin \mathit{past}(\overline{G_{\text{OS-PROT}}}, \alpha) \cap \mathit{past}(\overline{G_{\text{CL-PROT}}}, \alpha)$. Since p' follows from p in a single step and $p' \in G_{\text{OS-PROT}} \cap \overline{G_{\text{CL-PROT}}}$, it is the case that $p' \notin \mathit{past}(\overline{G_{\text{OS-PROT}}}, \alpha) \cap \mathit{past}(\overline{G_{\text{CL-PROT}}}, \alpha)$. Therefore, the third condition is violated and this decomposition of α is not valid.

(c) Suppose we split α at the state p'. Then p' is the last state of α_1 and the first state of α_2. Since $p' \in G_{\text{OS-PROT}} \bigcap \overline{G_{\text{CL-PROT}}}$, the second condition is violated. Therefore, this decomposition of α is not valid.

(d) Suppose we split α at any state succeeding p'. Then the state p' is in α_1. Since $p' \in G_{\text{OS-PROT}} \bigcap \overline{G_{\text{CL-PROT}}}$, the first condition is violated. Therefore, this decomposition of α is not valid.

Therefore, the execution α cannot be decomposed into any such α_1 and α_2. It follows that the first clause of Theorem 3.1.7 must hold; that is, every state in α is in $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$. This implies that the protector OS-PROT $\times$ CL-PROT guarantees $G_{\text{OS-PROT}} \cap G_{\text{CL-PROT}}$ in the VEHICLES automaton starting from $S_{\text{OS-PROT}} \cap S_{\text{CL-PROT}}$. ■

9.2 Overspeed and Collision Avoidance for the MERGE-VEHICLES Automaton

In the following lemma, we state that the composition of the protectors OS-PROT and MERGE-PROT guarantees that the vehicles of the MERGE-VEHICLES automaton neither exceed the speed limit, nor collide among themselves. It is important to note that it is assumed without proof that the protector OS-PROT and the Corollary 5.3.1 extend to the MERGE-VEHICLES automaton. In fact, since the strategy of the OS-PROT protector defined for the VEHICLES automaton in Chapter 5 does not depend on the nature of the track topology, the OS-PROT protector of Chapter 5 extends to the MERGE-VEHICLES automaton virtually unchanged.

Lemma 9.2.1 *The composition of* OS-PROT *and* MERGE-PROT *is a protector that guarantees* $G_{\text{OS-PROT}} \cap G_{\text{MERGE-PROT}}$ *in the* MERGE-VEHICLES *automaton starting from* $S_{\text{OS-PROT}} \cap S_{\text{MERGE-PROT}}$.

Proof: This proof follows precisely the steps of the proof of Lemma 9.1.1. ■

9.3 Overspeed and Collision Avoidance for the GRAPH-VEHICLES Automaton

In the following lemma, we state that the composition of the protectors OS-PROT and GRAPH-PROT guarantees that the vehicles of the GRAPH-VEHICLES automaton neither exceed the speed limit, nor collide among themselves. It is important to note that it is assumed without proof that the protector OS-PROT and the Corollary 5.3.1 extend to the GRAPH-VEHICLES automaton. In fact, since the strategy of the OS-PROT protector defined for the VEHICLES automaton in Chapter 5 does not depend on the nature of the track topology, the OS-PROT protector of Chapter 5 extends to the GRAPH-VEHICLES automaton virtually unchanged.

Lemma 9.3.1 *The composition of* OS-PROT *and* GRAPH-PROT *is a protector that guarantees* $G_{\text{OS-PROT}} \cap G_{\text{GRAPH-PROT}}$ *in the* GRAPH-VEHICLES *automaton starting from* $S_{\text{OS-PROT}} \cap S_{\text{GRAPH-PROT}}$.

Proof: This proof follows precisely the steps of the proof of Lemma 9.1.1. ■

Chapter 10

Conclusions and Future Work

This thesis investigates how the formal modeling and verification techniques of computer science can be used for the analysis of hybrid systems. The motivation behind such research lies in the inherent similarity of the hierarchical and decentralized control strategies of hybrid systems and the formal techniques used for the verification of distributed systems in computer science. The thesis focuses on the development of techniques that use hybrid I/O automata to model automated transportation systems and to verify that their protection subsystems enforce the desired safety properties. The long-term goal of such research is to develop a simple and scalable framework for modeling complex hybrid systems with stringent safety and performance requirements.

10.1 Summary

The thesis is split into two major parts. First, we develop an abstract model of a physical plant that is interacting with several protectors. Second, we specialize the abstract models of the physical system and the protectors to simplified versions of the PRT 2000™ and its overspeed and collision protection subsystems.

As indicated above, the first part of the thesis is devoted to the development of an abstract model of a physical plant and a number of protectors that guarantee particular safety or performance properties. Both the physical plant and the protectors are modeled as hybrid I/O automata. The protector automata communicate with the physical plant automaton through shared variables and discrete actions. If S, R, and G are subsets of the states of the physical plant, then a protector automaton A for the physical plant PP guarantees G 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. It is shown that if two or more protectors do not rely on the correct operation of each other, *i.e.*, if the protectors are independent, then their composition guarantees the properties guaranteed by each of the protectors being composed. On the other hand, if the protectors rely on the correct operation of each other, their composition guarantees the properties guaranteed by each of the protectors being composed only under certain conditions.

The abstract protector is parameterized in terms of the physical plant PP, the start states S, the sets of guarantee G and reliance R, the port j through which it communicates with the physical plant automaton, and the sampling period d. It is defined as the composition of a sensor and a discrete controller, both modeled as hybrid I/O automata. The sensor automaton samples the output variables of the physical plant at intervals of d time units and the discrete controller automaton issues protective actions so as to ensure that the physical plant exhibits the desired safety properties. The correctness of the abstract protector reduces the correctness proof of a protector implementation to a simulation proof among the states of the implementation and the particular instantiation of the abstract protector.

The second part of the thesis involves the proof of correctness of overspeed and collision protectors for a simple model of an automated transportation system involving n vehicles. The overspeed and collision protectors are redefined for three types of track topology: a single track, a track involving a Y-shaped merge, and a general track topology comprised of Y-shaped merges and diverges.

In the case of a single track, the overspeed protector is defined as the composition of n protectors, each of which guarantees that a particular vehicle does not exceed the speed limit, provided that none of the vehicles collide among themselves. Conversely, the collision protector is defined as the composition of n protectors, each of which guarantees that a particular vehicle does not collide into any of the vehicles it trails, provided that none of the vehicles exceed the speed limit and that none of the other vehicles collide into any of the vehicles they respectively trail.

In the cases of the more complicated track topologies, although the overspeed protector remains unchanged, the collision protectors are restructured. They involve the composition of $n(n-1)/2$ protectors, each of which guarantees that a particular unordered pair of vehicles do not collide between themselves, provided that none of the vehicles exceed the speed limit and that the vehicles of all other unordered pairs of vehicles do not collide between themselves.

Due to the correctness proof of the parameterized abstract protector, the proofs of correctness of the overspeed protectors for the individual vehicles and of the collision protectors for either individual, or unordered pairs of vehicles, are straightforward. They simply involve demonstrating the existence of a simulation relation among the states of the particular protector implementations and the particular instantiations of the parameterized abstract protector.

The composition of the overspeed protectors is straightforward due to their independence.

The proof of correctness of the overspeed protection subsystem involves the application of the aforementioned composition theorems for independent protectors. In the case of the collision protectors, since the individual collision protectors rely on the correct operation of each other, the proof of correctness of their composition is more involved. It relies on the careful decomposition of the collision protection subsystem so that the failure of multiple collision protectors at the same instant in time is prohibited. Similarly, the correct operation of the composition of the overspeed and collision protection subsystems relies on the fact that the overspeed protectors and the collision protectors can only fail through trajectories and discrete actions, respectively.

10.2 Evaluation

The contributions of this thesis are twofold. First, we develop an abstract model of an automated transportation system comprised of a physical plant and an arbitrary number of protectors. Second, we specialize the abstract model so as to analyze and verify a particular automated transportation system and its overspeed and collision protection systems.

The abstract models that are developed include the physical plant and a number of protectors. The abstract protector is parameterized in terms of the physical system, its start states, its sets of guarantee and reliance, the port with which it communicates with the physical plant and the sampling period. Therefore, the specification of a particular automated transportation system involves refinement of the abstract model. Moreover, 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, composition of independent protectors is straightforward. The safety properties of the individual protectors are guaranteed by the composed protector. Such compositional assertions also hold for dependent protectors under certain conditions. The use of abstraction, modular decomposition, and composition is hoped to allow the scalability of the formal method analysis and the verification of large and complex hybrid systems.

In this work, we demonstrate how hybrid I/O automaton techniques can be applied to the specification and verification of a very general automated transportation problem. We believe that the techniques developed in this thesis 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 plant with a set of overspeed protectors, one for each vehicle, and assume that no collisions occur in the physical system; conversely, to yield a collision free physical system, we compose the physical system with a set of collision protectors, either one for each vehicle, or one for each unordered pair of vehicles, and assume that none of the vehicles exceed the speed limit. The composition of the physical system in such ways formalizes the independence of the overspeed protectors, the interdependence of the collision protectors and more importantly the interdependence of the overspeed protectors and the collision protectors. 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.

10.3 Future Work

In this thesis, the treatment of automated transportation systems is a case study in the use of hybrid I/O automata to formally model hybrid systems. The focus of the research is in the use of abstraction, modularity, and composition to develop an abstract model of automated transportation systems to be used in the analysis and verification of transportation systems in use or under development. The long-term goal is to see how the formal methods of computer science can be used to formally model hybrid systems in a modular and systematic way and to verify their safety or performance characteristics. However, issues that have yet to be addressed involve the topics of robustness, scalability, tractability, and the use of formal methods as part of the system design process.

The work in this thesis assumes an ideal system; that is, the communication among the various subsystems is assumed to be correct and reliable, and to occur in a timely fashion. Moreover, the sampling of the state of the physical plant is assumed to be exact and the effects of the protective actions are assumed to be precise. Since, these assumptions are far from realistic, future research could involve the development of formal methods for analyzing and verifying automated transportation systems that are robust with respect to communication delays and uncertainty. For example, the treatment of automated transportation systems of this thesis could be extended to allow delays in the communication between the plant and the protectors and uncertainty either in the sampling of the state, or in the effects of the protective actions. The treatment of automated transportation systems could also be extended to allow fault tolerance; for example, allowing the track topology to be dynamic so that vehicles are not allowed to travel on branches of the track that have failed either structurally, or due to unexpected accidents.

In this thesis, we develop formal modeling techniques that are based on abstraction, modularity, and subsystem composition. The motivation behind this approach is the intent to model and verify complex hybrid systems that involve hierarchical and decentralized control schemes. Therefore, it is imperative to examine the scalability and tractability characteristics of the formal modeling techniques developed. The success in modeling the overspeed and collision protectors of an automated transportation system in this thesis indicates that the modeling techniques that are based on hybrid I/O automata are scalable to larger and more complex systems. However, the study and formal analysis of more complex systems remains to be done. In particular, it would be interesting to examine how complex continuous-time dynamics affect the formal modeling tools developed in this thesis. Moreover, the lengthy correctness proofs, which were done by hand in this thesis, expose issues of tractability concerning the analysis and verification of complex transportation systems. In fact, they dictate that computer aided verification methods for hybrid I/O automata be developed.

The formal modeling techniques developed in this thesis are techniques intended for the analysis and verification of automated transportation systems. Future research could investigate the potential of using formal methods of computer science as an integral part of the design of the hierarchical and decentralized control schemes of automated transportation systems and of hybrid systems in general.

References

[1] Jean-Raymond Abrial, Egon Börger, and Hans Langmaack. Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Methods for Semantics and Specification, International Conference and Research Center for Computer Science*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1996. The Methods for Semantics and Specification, International Conference and Research Center for Computer Science took place in Schloss, Dagstuhl, Germany, in June 1995.

[2] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138(1):3–34, February 1995. Preliminary version appeared as Ref. 3.

[3] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The Algorithmic Analysis of Hybrid Systems. In *Proc. 11th International Conference on Analysis and Optimization of Systems, Discrete-Event Systems*, volume 199 of *Lecture Notes in Control and Information Sciences*, pages 331–351. Springer-Verlag, 1994.

[4] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993. Extended version appeared as Ref. 2.

[5] Rajeev Alur and David L. Dill. Automata for Modeling Real-Time Systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[6] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994. Preliminary version appeared as Ref. 5.

[7] 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.

[8] 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.

[9] Jean-Marie Flaus and Ollagnon. Guy. Hybrid Flow Nets of Hybrid Processes Modeling and Control. In Oded Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 213–227. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.

[10] 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.

[11] 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. 10.

[12] Datta N. Godbole and John Lygeros. Longitudinal Control of a Lead Car of a Platoon. *IEEE Transactions on Vehicular Technology*, 43(4):1125–1135, November 1994. Also appeared in *Proc. 13th American Control Conference*, pages 398–402, Baltimore, Maryland, June/July 1994.

[13] Datta N. Godbole, John Lygeros, and Shankar Sastry. Hierarchical Hybrid Control: a Case Study. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 166–190. Springer-Verlag, 1995. Also appeared in *Proc. 33rd IEEE Conference on Decision and Control*, pages 1592–1597, Orlando, Florida, December 1994.

[14] 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.

[15] Constance Heitmeyer and Nancy Lynch. The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems. In *Proc. 15th IEEE Real-Time Systems Symposium*, pages 120–131, San Juan, Puerto Rico, December 1994. IEEE Computer Society Press.

[16] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Temporal Proof Methodologies for Real-Time Systems. In *Proc. 18th Annual Symposium on Principles of Programming Languages*, pages 353–366. ACM Press, 1991.

[17] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed Transition Systems. In J.W. de Bakker, K. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proc.*

REX Workshop "Real-Time: Theory in Practice", volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, 1992. The REX Workshop "Real-Time: Theory in Practice" took place in Mook, The Netherlands, in June 1991.

[18] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Temporal Proof Methodologies for Timed Transition Systems. *Information and Computation*, 112(2):273–337, August 1994. Preliminary versions of Part I and Part II appeared as Refs. 17 and 16, respectively.

[19] Leslie Lamport. The Temporal Logic of Actions. Research Report 79, Digital Equipment Corporation Systems Research Center, Palo Alto, California, December 1991.

[20] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994. Also appeared as Ref. 19.

[21] Gunter Leeb and Nancy A. Lynch. Proving Safety Properties of the Steam Boiler Controller. In J.R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1996. Preliminary version presented as "Using Timed Automata for the Steam Boiler Controller Problem" at the Methods for Semantics and Specification, International Conference and Research Center for Computer Science in Schloss, Dagstuhl, Germany in June 1995.

[22] 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.

[23] John Lygeros and Datta N. Godbole. An Interface between Continuous and Discrete Event Controllers for Vehicle Automation. In *Proc. 13th American Control Conference*, pages 801–805, Baltimore, Maryland, June/July 1994. Also appeared as Ref. 24.

[24] John Lygeros and Datta N. Godbole. An Interface between Continuous and Discrete Event Controllers for Vehicle Automation. Technical Report UCB-ITS-PRR-94-12, Institute of Transportation Studies, University of California, Berkeley, April 1994.

[25] 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.

[26] John Lygeros, Datta N. Godbole, and Shankar Sastry. A Verified Hybrid Controller for Automated Vehicles. Technical Report UCB-ITS-PRR-97-9, Institute of Transportation Studies, University of California, Berkeley, 1997. To appear in the Special Issue on Hybrid Systems of the IEEE Transactions on Automatic Control. Preliminary version appeared as Ref. 25.

[27] John Lygeros, Datta N. Godbole, and Sastry Shankar. A Game Theoretic Approach to Hybrid System Design. 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*. Springer-Verlag, 1996. The DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems took place in New Brunswick, New Jersey, in October 1995.

[28] 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.

[29] 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.

[30] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. Preprint. Preliminary versions appeared as Refs. 28 and 29, June 1997.

[31] Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part I: Untimed Systems. Technical Memo MIT/LCS/TM-486, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1993.

[32] Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part I: Untimed Systems. *Information and Computation*, 121(2):214–233, September 1995. Preliminary version appeared as Ref. 31.

[33] 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, Massachusetts, April 1995.

[34] 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. 33.

[35] Oded Maler, Zohar Manna, and Amir Pnueli. From Timed to Hybrid Systems. In J.W. de Bakker, K. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484. Springer-Verlag, 1992. The REX Workshop "Real-Time: Theory in Practice" took place in Mook, The Netherlands, in June 1991.

[36] Zohar Manna and Amir Pnueli. Verifying Hybrid Systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 4–35. Springer-Verlag, 1993.

[37] Amir Pnueli and Joseph Sifakis, editors. *Special Issue on Hybrid Systems*, volume 138, part 1 of *Theoretical Computer Science*. Elsevier Science Publishers, February 1995.

[38] Thomas Stauner, Olaf Müller, and Max Fuchs. Using HyTech to Verify an Automotive Control System. In Oded Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 139–153. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.

[39] Peter Terwiesch, Erich Scheiben, Anders Jenry Petersen, and Thomas Keller. A Digital Real-Time Simulator for Rail-Vehicle Control System Testing. In Oded Maler, editor,

Proc. International Workshop on Hybrid and Real-Time Systems (HART'97), volume 1201 of *Lecture Notes in Computer Science*, pages 199–212. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.

[40] Adam L. Turk, Scott T. Probst, and Gary J. Powers. Verification of Real Time Chemical Processing Systems. In Oded Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 259–272. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.

[41] Pravin Varaiya. Smart Cars on Smart Roads: Problems of Control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[42] 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.

[43] Henri B. Weinberg. Correctness of Vehicle Control Systems: A Case Study. Master of Science Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1996.