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 $\alpha_i = w_0a_1w_1a_2w_2\cdots a_iw_i$, 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. $\alpha_i = w_0a_1w_1a_2w_2\cdots a_iw_i$ 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 $\alpha = w_0a_1w_1a_2w_2\cdots$ 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 $\mathit{h\text{-}trace}(\alpha'_{PP}) = \mathit{h\text{-}trace}(\pi_{PP}(\alpha_{PP \times A_1}))$, $\mathit{h\text{-}trace}(\alpha'_{A_2}) = \mathit{h\text{-}trace}(\pi_{A_1}(\alpha_{PP \times A_1}))$, 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

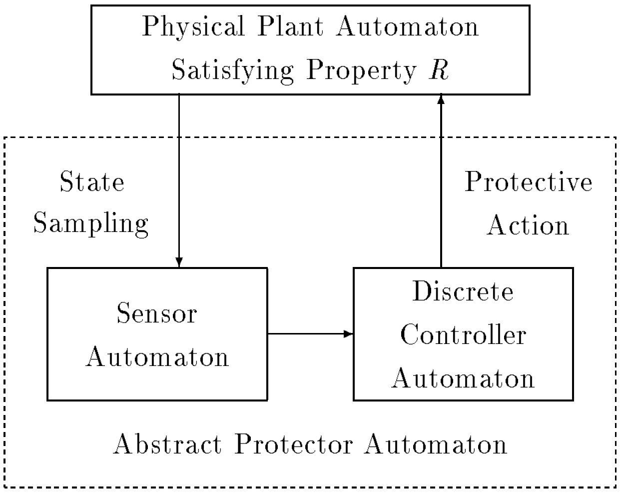
In this section, we define an abstract protector that is parameterized in terms of:

- *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.

> $future_{PP,R,j} : \mathcal{P}(R) \times \mathcal{P}(\mathbb{R}^{\geq 0}) \rightarrow \mathcal{P}(R)$, 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 $T \subseteq \mathbb{R}^{\geq 0}$, $\mathit{future}_{PP,R,j}(p', T) \subseteq \mathit{future}_{PP,R,j}(p, T)$.*

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, $\mathit{no\text{-}op}_{PP,R,j}$, which yields, for a given state in R, the set of input actions on port j of the PP automaton that do not affect the state of the PP automaton, provided they are executed prior to either time-passage, or other input actions on port j.

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

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 $\mathit{no\text{-}op}_{PP,R,j}(p) \neq \emptyset$.*

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, $\textit{very-safe}_{PP,R,G,j}$, 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 $\textit{very-safe}_{PP,R,G,j}$ 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.

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

It is important to note that the set $\textit{very-safe}_{PP,R,G,j}$ 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 $\textit{very-safe}_{PP,R,G,j}$.

Lemma 3.2.3

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

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

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

$\textit{safe}_{PP,R,G,j} \subseteq R$, defined by: $p \in \textit{safe}_{PP,R,G,j}$ if and only if both of the following hold: 1. $\textit{future}_{PP,R,j}(p, 0) \subseteq G$. 2. There exists an input action π on port j, such that for every $p', p'' \in R$ satisfying $p' \in \textit{future}_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in \textit{very-safe}_{PP,R,G,j}$.

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

$safe_{PP,R,G,j} : \Sigma^{in}_{PP_j} \rightarrow \mathcal{P}(R)$, 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 $p \in \mathit{safe}_{PP,R,G,j}(\mathtt{decision}(y))$.*

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-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: $\texttt{snapshot}(y)_j$, 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: $Y_{PP} :\in \boldsymbol{Y_{PP}}$

$\texttt{snapshot}(y)_j$
Pre: $next\text{-}snap_j = now_j$
y is current valuation of Y_{PP}
Eff: $Y_{PP} :\in \boldsymbol{Y_{PP}}$
$next\text{-}snap_j := now_j + d$

Trajectories:
for all $u \in Y_{PP}$
u assumes arbitrary values in $type(u)$ throughout w
$next\text{-}snap_j$ is constant throughout w
for all $t \in T_I$
$w(t).now_j = w(0).now_j + t$
$w(t).now_j \leq w(t).next\text{-}snap_j$

automaton using a $\texttt{snapshot}(y)_j$ output action. The $Sensor_j$ automaton keeps track of the appropriate times for scheduling each $\texttt{snapshot}(y)_j$ 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 $\texttt{snapshot}(y)_j$, 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 $\texttt{snapshot}(y)_j$ 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)
$\texttt{snapshot}(y)_j$, 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

$\texttt{snapshot}(y)_j$

Eff: if $y \in safe_{PP,R,G,j}$ then

$send_j :\in \{\phi \in \Sigma^{in}_{PP_j} \mid \forall\, p, p', p'' \in R$ such that
$p \lceil Y_{PP} = y$, $p' \in future_{PP,R,j}(p, 0)$, and $p' \xrightarrow{\phi}_{PP} p''$,
it is the case that $p'' \in delay\text{-}safe_{PP,R,G,j}(d)\}$

else

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

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-safe*. Therefore, from Lemma 3.2.5, part 1, it follows that $p'' \in$ *delay-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*$(\texttt{decision}(y))$.

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* $s.ppstate \in \mathit{delay\text{-}safe}_{PP,R,G,j}(s.next\text{-}snap_j - s.now_j)$.
2. *If* $s.send_j = \phi$*, for some* $\phi \in \Sigma^{in}_{PP_j}$*, then*
   (a) $\mathit{future}_{PP,R,j}(s.ppstate, 0) \subseteq G$*, and*
   (b) *For every* $p', p'' \in R$ *such that* $p' \in \mathit{future}_{PP,R,j}(s.ppstate, 0)$ *and* $p' \xrightarrow{\phi}_{PP} p''$*, it is the case that* $p'' \in \mathit{delay\text{-}safe}_{PP,R,G,j}(d)$.

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 $s.ppstate \in \mathit{delay\text{-}safe}(s.next\text{-}snap_j - s.now_j)$. 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 $s.ppstate \in \mathit{delay\text{-}safe}(0)$, which by Lemma 3.2.5, part 2, is equivalent to $s.ppstate \in \mathit{safe}$. 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 $s, s' \in \mathit{states}(PP \times Abs_j)$ such that $s.ppstate, s'.ppstate \in R$ and $\pi \in \Sigma_{PP \times Abs_j}$. We consider cases:

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

   From the effects of the $\mathtt{snapshot}(y)_j$ 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) $\mathit{future}(s'.ppstate, 0) \subseteq G$, and (b) for every $p', p'' \in R$ such that $p' \in \mathit{future}(s'.ppstate, 0)$ and $p' \xrightarrow{s'.send_j}_{PP} p''$, it is the case that $p'' \in \mathit{delay\text{-}safe}(d)$.

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

   For condition (a), since $s.ppstate \in \mathit{safe}$, it is the case that $\mathit{future}(s.ppstate, 0) \subseteq G$. Since the $\mathtt{snapshot}(y)_j$ 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 $\mathit{future}(s'.ppstate, 0) \subseteq G$, as needed.

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

$p' \in$ *future*$(s.ppstate, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in$ *delay-safe*(d). As a result of the step, it is the case that $s'.send_j = null$ and $s'.next\text{-}snap_j - s'.now_j = d$. Moreover, the invariant for the state s implies that $s'.ppstate \in$ *delay-safe*(d). Since $s'.next\text{-}snap_j - s'.now_j = d$ and $s'.ppstate \in$ *delay-safe*(d), it follows that $s'.ppstate \in$ *delay-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 $s.send_j = null$, then the invariant for s implies that $s.ppstate \in$ *delay-safe*(t), where $t = s.next\text{-}snap_j - s.now_j$; that is, *future*$(s.ppstate, [0, t]) \subseteq G$ and *future*$(s.ppstate, t) \subseteq$ *safe*. However, Lemma 3.2.2 implies that *future*$(s'.ppstate, t) \subseteq$ *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 *future*$(s'.ppstate, [0, t]) \subseteq G$ and *future*$(s'.ppstate, t) \subseteq$ *safe*, where $t = s'.next\text{-}snap_j - s'.now_j$. These two conditions imply that $s'.ppstate \in$ *delay-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 *future*$(s.ppstate, 0) \subseteq G$ and 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). However, Lemma 3.2.2 implies that *future*$(s'.ppstate, 0) \subseteq$ *future*$(s.ppstate, 0)$. Therefore, it follows that *future*$(s'.ppstate, 0) \subseteq G$ and 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). 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 $s, s' \in$ *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 $send_j = null$ throughout the trajectory w. Therefore, the invariant for the state s implies that $s.ppstate \in$ *delay-safe*$(s.next\text{-}snap_j - s.now_j)$; that is, *future*$(s.ppstate, [0, s.next\text{-}snap_j - s.now_j]) \subseteq G$ and *future*$(s.ppstate, s.next\text{-}snap_j - s.now_j) \subseteq$ *safe*. We must show that $s'.ppstate \in$ *delay-safe*$(s'.next\text{-}snap_j - s'.now_j)$; that is, *future*$(s'.ppstate, [0, s'.next\text{-}snap_j - s'.now_j]) \subseteq G$ and *future*$(s'.ppstate, s'.next\text{-}snap_j - s'.now_j) \subseteq$ *safe*. It suffices to show that *future*$(s'.ppstate, [0, s'.next\text{-}snap_j - s'.now_j]) \subseteq$

future$(s.ppstate, [0, s.next\text{-}snap_j - s.now_j])$ and *future*$(s'.ppstate, s'.next\text{-}snap_j - s'.now_j) \subseteq$ *future*$(s.ppstate, s.next\text{-}snap_j - s.now_j)$.

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

Using similar arguments, it can be shown that *future*$(s'.ppstate, s'.next\text{-}snap_j - s'.now_j) \subseteq$ *future*$(s.ppstate, s.next\text{-}snap_j - s.now_j)$. ■

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 $safe_{PP,R,G,j}$ via an execution that only involves states in R, it is the case that $s.ppstate \in G$.*

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

Theorem 3.2.9 *$Abs(PP, S, R, G, j, d)$ guarantees G in PP from $safe_{PP,R,G,j}$ 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 $s.ppstate \in G$, 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. ■