Hierarchical Interface-Based Supervisory Control with Data Events

Version 1.01

by

Ryan J. Leduc

Software Quality Research Laboratory Report No. 44
Dept. of Computing and Software
McMaster University
July 27, 2007

Abstract

Hierarchical Interface-based Supervisory Control (HISC) decomposes a discrete-event system (DES) into a high-level subsystem which communicates with $n \geq 1$ low-level subsystems, through separate interfaces which restrict the interaction of the subsystems. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability. As each clause of the definition can be verified using a single subsystem, the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources.

In this report, we extend the range of the behavior of low-levels that interfaces can model by adding a new type of event, low data events, and by relaxing some restrictions in the HISC definitions. This allows us to have (i) request events that don't need to be followed by an answer event, (ii) to start a low-level on a task and then poll it for completion, (iii) to be able to send additional commands while a low-level is already processing a command (iv) to model low-levels that behave as buffers, and (v) to allow unsolicited information (status etc.) to be sent up from a low-level.

Besides greatly enriching the behavior that can be modelled as interfaces and thus expanding the systems that HISC can effectively be applied to, the changes can enable behavior to be moved from the high-level to the low-levels. We demonstrate this when we discuss the application of our method to a large manufacturing system example based upon the AIP example, where we saw a 3.4 times reduction in computation time and a 6.5 times reduction in memory use. This helps prevent the high-level from growing too large, allowing the HISC method to apply to larger systems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the area of Discrete-Event Systems (DES), two common tasks are to verify that a composite system, based on a Cartesian product of subsystems, is (i) nonblocking and (ii) controllable. The main obstacle to performing these tasks is the combinatorial explosion of the product state space.

The Hierarchical Interface-based Supervisory Control (HISC) framework was proposed by Leduc et al. ([6, 7, 9, 8]) to alleviate the state explosion problem. The HISC approach decomposes a system into a *high-level subsystem* which communicates with $n \geq 1$ parallel *low-level subsystems* through separate interfaces that restrict the interaction of the subsystems. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability. As each clause of the definition can be verified using a single subsystem, the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources.

There has been some very promising recent work by Pena et al. [12], Flordal et al. [4], and Hill et al. [5] in using different forms of equivalence abstractions to verify flat systems in a modular fashion. These methods promise to increase the size of unstructured systems that we can verify. As individual levels (components) of an HISC system are treated as flat systems, these new results have the potential of being able to increase the individual component size that we can handle, if they can be adapted to verify the per component HISC conditions.

In this paper, we set out to extend the range of the behavior of low-levels that interfaces can model. This will have the affect of increasing the systems that HISC can be applied to as well as, in some cases, allow us to model certain low-levels in a more flexible, and intuitive way.

As we will see, extending the range of behavior we can model can have the side effect of allowing us to move behavior from the high-level to one or more low-levels, decreasing the size of the high-level.

As the high-level typically increases in size as we add more low-levels, it is often the bottleneck limiting the size of systems we can handle with the HISC method. The new changes we are introducing can thus result in more natural system design using HISC, as well as a decrease in computation time and memory usage.

The most important type of behavior we wish to be able to add, is the ability to to start a low-level on a task and then poll it for completion. Currently, we can start the low-level on a task by sending it a command (*request event*), but then we must wait for it to complete the task and send us a response (*answer event*). Between the request event and answer event, we can not send any more commands to the low-level such as to add another task to its queue, or even to send it an abort command. Once the low-level has sent an answer event, it can't currently provide the high-level with any additional status information until after the next request event. That means the interface can't change state at this point in response to changes in the low-level state that would effect which request events are currently available, as well as which answer events can follow these request events.

The current inability of the low-level to provide status information between the occurrence of an answer event and a request event is the key limiting factor. If we started a machine on a task, and then polled it to see if the task is complete, we might expect to get two possible responses: *taskDone* or *notDone*. However, if the task completed at the low-level between the reception of the *notDone*, and before the request event to check completion, the interface would still be saying that the answer event *notDone* is possible, even though it is not. This would violate the current HISC interface consistency condition.

What we need is a new type of interface events, called *low data events*, that can occur in between a request and answer event, allowing the low-level to send status information as needed. This will allow the interface automata to stay in a consistent state. Low data events will actually be allowed to occur at any state in the interface, providing the additional ability of allowing the low-level to provide information to the high-level at will.

These new events will not only allow us to model polling behavior, but more importantly allow us to model low-levels that behave, from an input-output perspective, as buffers. This is important as buffers are a very common structure in systems. Being able to represent them using interfaces allows us to move the behavior of such subsystems to a low-level, while still interacting with the system as a buffer (i.e. adding tasks to the subsystems queue individually, and removing them individually once processed).

In addition to these new abilities, we would also like to have request events that don't need to be

followed by an answer event. For instance, starting a task when using polling doesn't require a response, thus simplifying the interface automata. Finally, we would like to be able to send additional commands (additional information, an abort signal etc.) while a low-level is already processing a command.

In this report, we first discuss DES preliminaries and then introduce the HISC approach in Chapter 2. As the HISC method has already been explained and justified in detail in [7], [9] , and [8], we will present it quickly, focussing on our changes. We will discuss the new low data events, and the corresponding modifications to the HISC definitions to allow us to model the new types of behavior. We will then present our nonblocking and controllability results. Next we will discuss how to use the new method to model buffers, followed by an application to the AIP example from [6, 8].

1.1 DES Preliminaries

Supervisory control theory [13, 15, 14] provides a framework for the control of discrete-event systems (DES), systems that are discrete in space and time. For a detailed exposition of DES, see [14]. Below, we present a summary of the terminology that we use in this paper.

Let Σ be a finite set of distinct symbols (*events*), and Σ^* be the set of all finite sequences of events, including ϵ, the *empty string.* Let $L \subseteq \Sigma^*$ be a *language* over Σ. A string $t \in \Sigma^*$ is a prefix of $s \in \Sigma^*$ (written $t \leq s$) if $s = tu$, for some $u \in \Sigma^*$. The *prefix closure* of language L (denoted $\overline{L}$) is defined as $\overline{L} = \{t \in \Sigma^* \mid t \leq s \text{ for some } s \in L\}$. Let $\mathrm{Pwr}(\Sigma)$ denote the power set of Σ. For language L, the eligibility operator $\mathrm{Elig}_L : \Sigma^* \to \mathrm{Pwr}(\Sigma)$ is given by $\mathrm{Elig}_L(s) := \{\sigma \in \Sigma \,|s\sigma \in L\}$ for $s \in \Sigma^*$.

A DES automaton is represented as a 5-tuple $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ where Y is the state set, Σ is the event set, the partial function $\delta : Y \times \Sigma \to Y$ is the transition function, y_o is the initial state, and Y_m is the set of marker states. The function δ is extended to $\delta : Y \times \Sigma^* \to Y$ in the natural way. The notation $\delta(y, s)!$ means that δ is defined for $s \in \Sigma^*$ at state y. For DES $\mathbf{G}$, the language generated is denoted by $L(\mathbf{G})$, and is defined to be $L(\mathbf{G}) := \{s \in \Sigma^* |\ \delta(y_o, s)!\}$. The marked behavior of $\mathbf{G}$, is defined as $L_m(\mathbf{G}) := \{s \in L(G) |\ \delta(y_o, s) \in Y_m\}$. The reachable state subset of DES $\mathbf{G}$, denoted Y_r, is: $Y_r := \{y \in Y |\ (\exists s \in \Sigma^*)\, \delta(y_o, s) = y\}$. A DES $\mathbf{G}$ is reachable if $Y_r = Y$. We will always assume $\mathbf{G}$ is reachable. The coreachable state subset, denoted by Y_{cr}, is $Y_{cr} := \{y \in Y |\ (\exists s \in \Sigma^*)\, \delta(y, s) \in Y_m\}$. A DES is *coreachable* if $Y_{cr} = Y$.

Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$. For $i = 1, 2$, $s \in \Sigma^*$, and $\sigma \in \Sigma$, we define the *natural*

projection $P_i : \Sigma^* \to \Sigma_i^*$ according to:

$$\begin{aligned} P_i(\epsilon) &= \epsilon, \quad P_i(\sigma) = \begin{cases} \epsilon \text{ if } \sigma \notin \Sigma_i \\ \sigma \text{ if } \sigma \in \Sigma_i \end{cases} \\ P_i(s\sigma) &= P_i(s)P_i(\sigma) \end{aligned}$$

The *synchronous product of languages* L_1 and L_2, denoted $L_1||L_2$, is defined to be:

$$L_1||L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$$

where $P_i^{-1} : \mathrm{Pwr}(\Sigma_i^*) \to \mathrm{Pwr}(\Sigma^*)$ is the inverse image function of P_i.

The *synchronous product of DES* $\mathbf{G}_i = (Y_i, \Sigma_i, \delta_i, y_{o_i}, Y_{m_i})$ $(i = 1, 2)$, denoted $\mathbf{G}_1||\mathbf{G}_2$, is defined to be a reachable DES $\mathbf{G}$ with event set $\Sigma = \Sigma_1 \cup \Sigma_2$ and properties:

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1)||L_m(\mathbf{G}_2), \quad L(\mathbf{G}) = L(\mathbf{G}_1)||L(\mathbf{G}_2),$$

For DES, the two main properties we want to check are *nonblocking* and *controllability.* A DES $\mathbf{G}$ is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$. This is equivalent to saying that every reachable state is also coreachable.

For controllability, we assume the standard event partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events.* To control the plant, we define a *supervisor* represented as an automaton $\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m)$.

Definition 1 *Let* $\Sigma := \Sigma_1 \cup \Sigma_S, P_1 : \Sigma^* \to \Sigma_1^*$ *and* $P_S : \Sigma^* \to \Sigma_S^*$. *Define* $\mathcal{L}_{\mathbf{G}_1} := P_1^{-1}(L(\mathbf{G}_1))$ *and* $\mathcal{L}_{\mathbf{S}} := P_S^{-1}(L(\mathbf{S}))$. *A supervisor* $\mathbf{S}$ *is* controllable *for a plant* $\mathbf{G}_1$ *if* $\mathcal{L}_{\mathbf{S}}\Sigma_u \cap \mathcal{L}_{\mathbf{G}_1} \subseteq \mathcal{L}_{\mathbf{S}}$ *or, equivalently,* $(\forall s \in \mathcal{L}_{\mathbf{G}_1} \cap \mathcal{L}_{\mathbf{S}})$ $Elig_{\mathcal{L}_{\mathbf{G}_1}}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{L}_{\mathbf{S}}}(s)$.

Chapter 2

HISC with Low Data Events: Nonblocking

A HISC system currently is a two-level system which includes one *high-level subsystem* and $n \geq 1$ *low-level subsystems*. The high-level subsystem communicates with each low-level subsystem through a separate *interface.*

In HISC there is a master-slave relationship. A high-level subsystem sends a command to a particular low-level subsystem, which then performs the indicated task and returns an answer. Fig. 2.1 shows conceptually the structure and information flow of the system. This style of interaction is enforced by an interface that mediates communication between the two subsystems. All system components, including the interfaces, are modeled as automata.

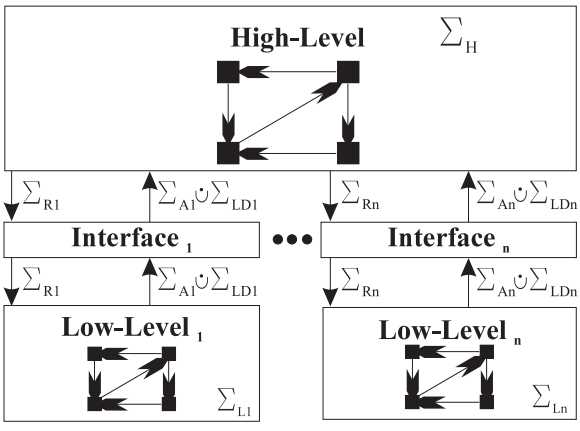


Figure 2.1: Interface Block Diagram

In order to restrict information flow and decouple the subsystems, the system alphabet is partitioned into pairwise disjoint alphabets:

$$\Sigma := \Sigma_H \dot{\cup} \dot{\bigcup_{j=1,\ldots,n}} [\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}] \tag{2.1}$$

The events in Σ_H are called *high-level events* and the events in Σ_{L_j} are the j^{th} *low-level events* ($j = 1, \ldots, n$) as these events appear only in the high-level and j^{th} low-level subsystem models, $\mathbf{G}_H$ and $\mathbf{G}_{L_j}$ respectively. We then have $\mathbf{G}_H$ defined over event set $\Sigma_H \dot{\cup} (\dot{\cup}_{j \in \{1,\ldots,n\}} [\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}])$ and $\mathbf{G}_{L_j}$ defined over event set $\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. We model the j^{th} interface by DES $\mathbf{G}_{I_j}$, which is defined over event set $\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. We define our *flat system* to be $\mathbf{G} = \mathbf{G}_H || \mathbf{G}_{I_1} || \mathbf{G}_{L_1} || \ \ldots || \mathbf{G}_{I_n} || \mathbf{G}_{L_n}$. By flat system we mean the equivalent DES if we ignored the interface structure. For the remainder of this paper, the index j has range $\{1, \ldots, n\}$.

As the j^{th} *interface*, $\mathbf{G}_{I_j}$, is only concerned with communication between the two subsystems, it is defined over the events that are common to both levels of the hierarchy. The events in Σ_{R_j}, called *request events*, represent commands sent from the high-level subsystem to the j^{th} low-level subsystem. The events in Σ_{A_j} are *answer events* and represent the low-level subsystem's responses to the request events.

To these two event types present in the HISC formulation of [6, 7, 9, 8], we add a new event type called *low data events* (Σ_{LD}). These events provide a means for a low-level to send information (data) through the interface, independent of the standard command-answer structure offered by the request and answer events. These new events are the key to keeping the state of the interfaces consistent with the state of their corresponding low-levels. Otherwise, the state of an interface can only be updated after the high-level sends a command down and the low-level responds with an answer event. This is insufficient to allow interfaces to represent polling (and thus buffer) behavior. Request, answer, and low data events are collectively known as the set of *interface events*, defined as $\Sigma_I := \dot{\cup}_{k \in \{1,\ldots,n\}} [\Sigma_{R_k} \dot{\cup} \Sigma_{A_k} \dot{\cup} \Sigma_{LD_k}]$.

In order to enforce the serialization of requests and answers, we restrict the interfaces to the subclass of LD interfaces defined below. Our definition is based upon the state based command-pair interface definition of Leduc et al. [3, 11], which is equivalent to the definitions in [7, 9, 8]. Fig. 2.2 shows an example of a LD interface. It could correspond to a machine at the low-level with an effective internal buffer of two.

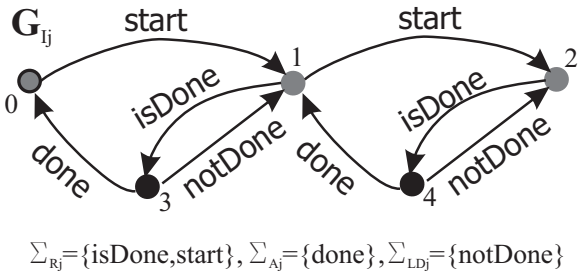


Figure 2.2: Example LD Interface

Definition 2 *The j^{th} interface DES* $\mathbf{G_{I_j}} = (X_j, \Sigma_{I_j}, \xi_j, x_{o_j}, X_{m_j})$ *is a* LD interface *if the following properties are satisfied:*

1. $x_{o_j} \in X_{m_j}$
2. $(\forall x \in X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x,\sigma)! \Rightarrow [\sigma \in \Sigma_{R_j}] \ \vee \ [\sigma \in \Sigma_{LD_j} \wedge \xi_j(x,\sigma) \in X_{m_j}]$
3. $(\forall x \in X_j - X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x,\sigma)! \Rightarrow [\sigma \in \Sigma_{A_j} \wedge \xi_j(x,\sigma) \in X_{m_j}] \vee [\sigma \in \Sigma_{LD_j}]$

In the command-pair interface definition, *Point 2* would have only allowed request events to go to an unmarked state, requiring that they must be followed by an answer event. We allow request events to also go directly to a marked state and thus not requiring a succeeding answer event. Allowing this type of behavior can lead to smaller, more compact interfaces. This is possible because our proofs require that request events be only possible at marked states (allows us to force a low-level to a marked state even if the desired request event is disabled) and every answer event be preceded by a request event (so we can apply *Point 5* of our LD interface consistency definition), not that every request event be followed by an answer event.

The other main change to the interface definition is the addition of the low-level data events. *Point 2* says low data events can occur at marked states, but they may only lead to another marked state. *Point 3* states that low data events may occur at unmarked states, and take us to either marked or unmarked states. These conditions are important because they ensure that the only way to get to a state where answer events are possible (an unmarked state), is via at least one request event.

To simplify notation in our exposition, we bring in the following event sets, natural projections, and languages.

$$\Sigma_{I_j} := \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j} \qquad P_{I_j} : \Sigma^* \rightarrow \Sigma_{I_j}^*$$

$$\Sigma_{IL_j} := \Sigma_{L_j} \cup \Sigma_{I_j}, \qquad P_{IL_j} : \Sigma^* \to \Sigma^*_{IL_j}$$

$$\Sigma_{IH} := \Sigma_H \cup \bigcup_{k\in\{1,\ldots,n\}} \Sigma_{I_k}, \qquad P_{IH} : \Sigma^* \to \Sigma^*_{IH}$$

$$\mathcal{H} := P_{IH}^{-1}(L(\mathbf{G}_H)), \qquad \mathcal{H}_m := P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^*$$

$$\mathcal{L}_j := P_{IL_j}^{-1}(L(\mathbf{G}_{L_j})), \qquad \mathcal{L}_{m_j} := P_{IL_j}^{-1}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^*$$

$$\mathcal{I}_j := P_{I_j}^{-1}(L(\mathbf{G}_{I_j})), \qquad \mathcal{I}_{m_j} := P_{I_j}^{-1}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^*$$

$$\mathcal{I} := \cap_{k\in\{1,\ldots,n\}}\mathcal{I}_k, \qquad \mathcal{I}_m := \cap_{k\in\{1,\ldots,n\}}\mathcal{I}_{m_k}$$

$$\Sigma_{LD} := \bigcup_{k\in\{1,\ldots,n\}} \Sigma_{LD_k} \qquad \Sigma_{IHc} := \Sigma_{IH} - \Sigma_{LD}$$

This allows us to present the proposition below that collects together several similar propositions. As it will be common in the proofs in that follow to show that membership in languages such as $\mathcal{H}$ is dependent only on events in specific subsets (for $\mathcal{H}$, events in subset Σ_{IH}), this proposition will be very useful.

Proposition 1 *For DES $\mathbf{G}_H$ defined over event set Σ_{IH}, DES $\mathbf{G}_{L_j}$ defined over Σ_{IL_j}, and DES $\mathbf{G}_{I_j}$ defined over event set Σ_{I_j} $(j = 1, \ldots, n)$, the following properties hold:*

(a) $(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{H}) \wedge (P_{IH}(s) = P_{IH}(s'))] \Rightarrow s' \in \mathcal{H}$

(b) $(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{H}_m) \wedge (P_{IH}(s) = P_{IH}(s'))] \Rightarrow s' \in \mathcal{H}_m$

(c) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{L}_k) \wedge (P_{IL_k}(s) = P_{IL_k}(s'))] \Rightarrow s' \in \mathcal{L}_k$

(d) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{L}_{m_k}) \wedge (P_{IL_k}(s) = P_{IL_k}(s'))] \Rightarrow s' \in \mathcal{L}_{m_k}$

(e) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{I}_k) \wedge (P_{I_k}(s) = P_{I_k}(s'))] \Rightarrow s' \in \mathcal{I}_k$

(f) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{I}_{m_k}) \wedge (P_{I_k}(s) = P_{I_k}(s'))] \Rightarrow s' \in \mathcal{I}_{m_k}$

Proof:

Same as proof of *Proposition 20* in [6].

■

We now present the properties that the system must satisfy to ensure that it interacts with the interfaces correctly.

Definition 3 *The n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is* LD interface consistent *with respect to the alphabet partition given by (2.1), if for all $j \in \{1, \ldots, n\}$, the following conditions are satisfied:*

Multi-level Properties

1. *The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_{L_j}$ is Σ_{IL_j}.*

2. *$\mathbf{G}_{I_j}$ is a LD interface.*

High-Level Property

3. $(\forall s \in \mathcal{H} \cap \mathcal{I})\ Elig_{\mathcal{I}_j}(s)\ \cap\ (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}) \subseteq Elig_{\mathcal{H}}(s)$

Low-Level Properties

4. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\ Elig_{\mathcal{I}_j}(s)\ \cap\ \Sigma_{R_j} \subseteq Elig_{\mathcal{L}_j}(s)$

5. $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$
$Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma_{L_j}^*) \cap \Sigma_{A_j} = Elig_{\mathcal{I}_j}(s) \cap \Sigma_{A_j}$ *where*

$$Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma_{L_j}^*) := \bigcup_{l \in \Sigma_{L_j}^*} Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(sl)$$

6. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$
$s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma_{L_j}^*)\ sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

The only change from the interface consistency definition of [8], is in *Point 3*. Previously, it only referred to answer events. This addition ensures that if a low-level (which includes an interface) needs a low data event to occur to reach a marked state, the high-level subsystem must allow it. However, if the high-level requires a low data event to occur, there is no such guarantee. For a discussion of the remaining points, please see [6, 8].

Now that we have presented the LD interface consistent definition, we present the *LD interface strict marking* condition, which is a restriction of the LD interface consistent definition. As we will see later, this restriction is useful as it implies *property 6* of the LD interface consistent definition, but is less expensive to evaluate.

Definition 4 *The n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$ is* LD interface strict marking *with respect to the alphabet partition given by (2.1), if:*

$$(\forall j \in \{1, \ldots, n\})\,(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\; s \in \mathcal{I}_{m_j} \Rightarrow\; s \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$$

The above statement could be summarized by saying that if we can bring the *interface* to a marked state, we are guaranteed to also have brought the low-level subsystem to a marked state. The above statement is equivalent to *property 6* of the LD interface consistent definition above, with string l restricted to the empty string.

We will now prove that we can use the LD interface strict marking condition to replace *Property 6* of the LD interface consistent definition.

Proposition 2 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, satisfies* properties 1-5 *of the LD interface consistency definition and is LD interface strict marking with respect to the alphabet partition given by (2.1), then the system is LD interface consistent.*

Proof:

Assume that the system satisfies *properties 1-5* of the LD interface consistency definition and is LD interface strict marking. **(1)**

We now show this implies the system is LD interface consistent.

From **(1)**, we immediately have the system satisfying the first five properties of the LD interface consistency definition.

All that remains is to show that the system satisfies *property 6* of the definition. This means showing: $(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\; s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma^*_{L_j})\; sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

Let $j \in \{1, \ldots, n\}$, $s \in \mathcal{L}_j \cap \mathcal{I}_j$, and assume $s \in \mathcal{I}_{m_j}$

We will now show this implies: $(\exists l \in \Sigma^*_{L_j})\; sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

From **(1)**, we have that the system is interface strict marking and we can thus conclude that:

$s \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

We thus take $l = \epsilon$ and we have $sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$, as required.

We thus have the system satisfying *properties 1-6,* and is thus LD interface consistent. ■

2.1 Local Conditions for Global Nonblocking of the System

We now provide the conditions that the subsystems and their interface(s) must satisfy in addition to the interface consistency properties, if the flat system $\mathbf{G}$ is to be nonblocking.

Definition 5 *The n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is said to be* LD level-wise nonblocking *if the following conditions are satisfied:*

(I) LD nonblocking at the high-level*:*

$(\forall s \in \mathcal{H} \cap \mathcal{I})(\exists s' \in (\Sigma - \Sigma_{LD})^*)\, ss' \in \mathcal{H}_m \cap \mathcal{I}_m$

(II) nonblocking at the low-level*:*

$(\forall j \in \{1, \ldots, n\})\, \overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$

Like the level-wise nonblocking definition of [8], Definition 5 requires that the high-level and each low-level be individually nonblocking. However, we add the further restriction that each string in the high-level's closed behavior must be extendable to a marked string without using low data events. The reason for this is that although the high-level can react to the occurrence of low data events, it has no guarantee that a low-level will ever allow these events to occur. As a result, it must ensure that it can always return to a marked state without them. A useful rule of thumb is to make sure that for each reachable unmarked state in each DES at the high-level (including the interfaces), there is at least one non low data event transition defined leading to a different state. This doesn't guarantee the system will be LD level-wise nonblocking, but will avoid many problems.

We have now presented all the changes to the HISC definitions of [8], required to model the new types of behavior discussed in Chapter 1. It can be easily shown that if a n^{th} degree interface system satisfies the level-wise nonblocking and interface consistency definitions of [8], and contains no low data events (i.e. $\Sigma_{LD} = \emptyset$), then it follows that it is also LD level-wise nonblocking and LD interface consistent. It's also easy to see that a system is LD level-wise controllable (see Chapter 3) if and only if it is level-wise controllable as defined in [8]. The new definitions were difficult to develop since we required that they be backwards compatible, make as few changes as possible to the previous HISC definitions, allow for compact interfaces, and allow for significant reuse of existing proofs and software, while still allowing us to be able to model the desired new behaviors.

2.2 Per Low-level Propositions

We will first give a set of propositions that apply to each low-level in turn to show that we can separately bring the high-level subsystem and that low-level together to a marked state. In the next section, we will combine these results to show that we can bring the entire system to a marked state.

Our first proposition says that a string s accepted by the high-level subsystem and the j^{th} low-level can always be extended to a string accepted by the high-level subsystem, and marked by the j^{th} low-level. In other words, the j^{th} low-level is not dependent on $\Sigma - \Sigma_{IL_j}$ events to reach a marked state.

Proposition 3 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j)(\exists l \in \Sigma_{IL_j}^*)\ sl \in \mathcal{H} \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$$

Proof:

See page 52.

■

Our next proposition asserts that a string s accepted by the j^{th} low-level and marked by the high-level subsystem and the j^{th} interface, implies that s can be extended by a low-level string l such that sl is marked by the high-level subsystem and the j^{th} low-level. In other words, if you can get the high-level subsystem and the j^{th} interface to a marked state, you can always bring the j^{th} low-level subsystem to a marked state by a string containing events the high-level subsystem and the j^{th} low-level are indifferent to.

Proposition 4 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H}_m \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})\ (\exists l \in \Sigma_{L_j}^*)\ sl \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$$

Proof:

Assume system is LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$ and $s \in \mathcal{H}_m \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$. **(2)**

We will now show that we can construct a string $l \in \Sigma_{L_j}^*$ such that $sl \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

From **(2)**, we have $s \in \mathcal{L}_j \cap \mathcal{I}_{m_j}$ and thus $s \in \mathcal{I}_{m_j}$ and $s \in \mathcal{L}_j \cap \mathcal{I}_j$.

From **(1)**, we can apply *Point 6* of the LD interface consistency definition and conclude:

$$(\exists l \in \Sigma_{L_j}^*)\, sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j} \qquad \textbf{(3)}$$

As $l \in \Sigma_{L_j}^*$, we have $P_{IH}(s) = P_{IH}(sl)$. From **(2)**, we have $s \in \mathcal{H}_m$. We can now apply *Proposition 1*, *point b*, and conclude: $sl \in \mathcal{H}_m$

Combining with **(3)**, we have $l \in \Sigma_{L_j}^*$ and $sl \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$, as required.

■

The next proposition says that if a string s, accepted by the j^{th} low-level, is extended by a string h containing no low data and Σ_{A_j} events, and is accepted by the j^{th} interface, then h will be accepted by the j^{th} low-level as well. It was added to make *Proposition 6* simpler.

Proposition 5 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)(\forall h \in (\Sigma_{IHc} - \Sigma_{A_j})^*)\, sh \in \mathcal{I}_j \Rightarrow sh \in \mathcal{L}_j$$

Proof:

See page 54.

■

Our next three propositions can be grouped together as each one builds on the one before it, each time handling progressively more general situations. In the first of the three propositions, we are given a string s accepted by the high-level subsystem and the j^{th} low-level, and a string h of the form $(\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j})$. This means that h ends with exactly one request event and one *answer event* in the given order, and that h does not contain an event in Σ_{A_j} before the last Σ_{R_j} event. The proposition asserts that if string h extends string s such that sh is acceptable to the high-level subsystem and j^{th} interface, then a string u can be constructed such that u has a high-level image equal to h, and that su is acceptable to the high-level subsystem and j^{th} low-level, and marked by the j^{th} interface. In other words, we can use string h as a basis to construct string u by adding low-level events so that the j^{th} low-level subsystem will accept the request event(s) and answer event contained in h. As these events are common to both levels, they must agree on their occurrence.

Proposition 6 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j)(\forall h \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j})$
$sh \in \mathcal{H} \cap \mathcal{I}_j \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\ (su \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}) \wedge (P_{IH}(u) = h)$

Proof:

See page 56.

■

The next proposition of the group is different from *Proposition 6* as it allows h to contain one or more answer events (i.e. multiple command-pairs). It uses *Proposition 6* in an inductive proof to handle an arbitrary number of answer events in string h.

Proposition 7 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})(\forall h \in \Sigma^*_{IHc}.\Sigma_{A_j})$
$sh \in \mathcal{H} \cap \mathcal{I}_j \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\ (P_{IH}(u) = h) \wedge (su \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})$

Proof:

See page 58.

■

Our last proposition of the group is more general than *Proposition 7* as it handles the case that string h doesn't contain *answer events* or doesn't end in an answer event. It makes use of *Proposition 7* to handle the other cases.

Proposition 8 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})(\forall h \in \Sigma^*_{IHc})$
$sh \in \mathcal{H}_m \cap \mathcal{I}_{m_j} \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\,(su \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}) \wedge (P_{IH}(u) = h)$

Proof:

See page 61.

■

2.3 Global Propositions and Theorem

We will now show that we can combine the per low-level results of the previous section so that we can bring the entire system to a marked state.

Our next proposition says that for every string accepted by our system, we can always bring all the low-levels to a marked state via a string containing only low-level and interface events. This is useful as it ensures that after string sl, all interfaces are in a marked states. This means that any future extensions that contain answer events also contain a preceding request event, making the application of *Point 5* of our LD interface consistent definition straight forward.

Proposition 9 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)])\,(\exists l \in \Sigma_{IL}^*)\ (sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])$$

Proof:

See page 63.

■

Our next proposition says that for every string accepted by our system, we can always find a string containing only high-level and interface events but no low data events, that will bring the high-level to a marked state. However, string h may not be accepted by one or more of the low-levels.

Proposition 10 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)])\,(\exists h \in \Sigma_{IHc}^*)\ \ sh \in \mathcal{H}_m \cap \mathcal{I}_m$$

Proof:

Assume system is LD level-wise nonblocking and LD interface consistent. **(1)**

Let $s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)]$ **(2)**

We will now show this implies: $(\exists h \in \Sigma_{IHc}^*)\ \ sh \in \mathcal{H}_m \cap \mathcal{I}_m$

We first note that **(2)** implies $s \in \mathcal{H} \cap \mathcal{I}$.

We next apply *Point I* of the LD level-wise nonblocking definition and conclude:

$(\exists h' \in (\Sigma - \Sigma_{LD})^*)\, sh' \in \mathcal{H}_m \cap \mathcal{I}_m$ **(4)**

We next note that:

$$\begin{aligned} P_{IH}(sh') &= P_{IH}(s)P_{IH}(h') \\ &= P_{IH}(s)P_{IH}(P_{IH}(h')) \text{ as the natural projection is idempotent.} \\ &= P_{IH}(sP_{IH}(h')) \end{aligned} \quad (\mathbf{5})$$

We can now apply *Proposition 1*, *point b*, and conclude:

$sP_{IH}(h') \in \mathcal{H}_m$ **(6)**

As $\Sigma_{I_j} \subseteq \Sigma_{IH}$ for $j = 1, \ldots, n$, we can conclude by **(5)** that $P_{I_j}(sh') = P_{I_j}(sP_{IH}(h'))$.

We can now apply *Proposition 1*, *point f*, and conclude:

$sP_{IH}(h') \in \cap_{j\in\{1,\ldots,n\}} \mathcal{I}_{m_j}$

Combining with **(6)**, we thus have: $sP_{IH}(h') \in \mathcal{H}_m \cap \mathcal{I}_m$

We next note that by **(4)**, we have $h' \in (\Sigma - \Sigma_{LD})^*$.

$\Rightarrow P_{IH}(h') \in (\Sigma_{IH} - \Sigma_{LD})^* = \Sigma^*_{IHc}$

We then take $h = P_{IH}(h')$ and we have as required: $h \in \Sigma^*_{IHc}$ and $sh \in \mathcal{H}_m \cap \mathcal{I}_m$

■

The next proposition says that given an appropriate strings s and h, we can modify string h by adding appropriate low-level events (which are ignored by the high-level) into a string u, such that the flat system is in a marked state. It is key that h does not contain any low data events as there is no guarantee that we could get the low-levels to accept them.

Proposition 11 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])(\forall h \in \Sigma^*_{IHc})$

$sh \in \mathcal{H}_m \cap \mathcal{I}_m \Rightarrow (\exists u \in \Sigma^*)\, (su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]) \wedge (P_{IH}(u) = h)$

Proof:

See page 65. ■

This brings us to our main nonblocking result. The theorem below states that verifying that the system is LD level-wise nonblocking and LD interface consistent is sufficient to show that it is nonblocking. As verifying the LD level-wise nonblocking and LD interface consistent conditions only require a single level at a time, we can evaluate each level independently. We thus do not need to construct the entire system model, potentially significantly reducing the computational resources required.

Theorem 1 *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$$L(G) = \overline{L_m(G)}$$

where $G = G_H||G_{L_1}|| \ldots ||G_{L_n}||G_{I_1}|| \ldots ||G_{I_n}$

Proof:

Assume system is LD level-wise nonblocking and LD interface consistent. **(1)**

As $\overline{L_m(G)} \subseteq L(G)$ is automatic, it suffices to show $L(G) \subseteq \overline{L_m(G)}$

Let $s \in L(G) = \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)]$ (2)

We will now show this implies: $s \in \overline{L_m(G)}$

It is sufficient to show: $(\exists u \in \Sigma^*)\ su \in L_m(G) = \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

Our first step is to show that we can construct a low-level string that will be accepted by the high-level, and will bring all n low-levels to a marked state. We can achieve this immediately by applying *Proposition 9* and conclude:

$(\exists l \in \Sigma^*_{IL})\ sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$ **(3)**

Our next step will be to show that we can construct a string $u' \in \Sigma^*$ such that $slu' \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$.

To achieve this, we will apply *Proposition 11*. Before we can apply the proposition, we must first construct a suitable $h \in \Sigma^*_{IHc}$.

We first note that **(3)** implies that $sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)])$. We can now apply *Proposition 10*, taking sl to be string s in said proposition, and conclude:

$$(\exists h \in \Sigma^*_{IHc})\ \ (slh \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}]) \qquad \textbf{(4)}$$

Combining with **(3)**, we can now apply *Proposition 11*, taking sl to be string s in said proposition, and conclude:

$$(\exists u' \in \Sigma^*)\,(slu' \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$$

We take string $u = lu'$ and we have $su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})] = L_m(G)$, as required.

■

Chapter 3

HISC With Low Data Events: Controllability

For controllability, we need to split the subsystems into their plant and supervisor components. To do this, we define the *high-level plant* to be $\mathbf{G}_H^p$, and the *high-level supervisor* to be $\mathbf{S}_H$ (both defined over event set Σ_{IH}). Similarly, the j^{th} *low-level plant* and *supervisor* are $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ (defined over Σ_{IL_j}). The high-level subsystem and the j^{th} low-level subsystem are then $\mathbf{G}_H := \mathbf{G}_H^p||\mathbf{S}_H$ and $\mathbf{G}_{L_j} := \mathbf{G}_{L_j}^p||\mathbf{S}_{L_j}$, respectively.

We can now define our *flat supervisor* and *plant* as well as some useful languages as follows:

$$\begin{aligned}
\mathbf{Plant} &:= \mathbf{G}_H^p||\mathbf{G}_{L_1}^p||\ldots||\mathbf{G}_{L_n}^p \\
\mathbf{Sup} &:= \mathbf{S}_H||\mathbf{S}_{L_1}||\ldots||\mathbf{S}_{L_n}||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n} \\
\mathcal{H}^p &:= P_{IH}^{-1}L(\mathbf{G}_H^p), \quad \mathcal{S}_H := P_{IH}^{-1}L(\mathbf{S}_H), \subseteq \Sigma^* \\
\mathcal{L}_j^p &:= P_{IL_j}^{-1}L(\mathbf{G}_{L_j}^p), \quad \mathcal{S}_{L_j} := P_{IL_j}^{-1}L(\mathbf{S}_{L_j}), \subseteq \Sigma^*
\end{aligned}$$

This allows us to present the proposition below that collects together several similar propositions. As it will be common in the proofs that follow to show that membership in languages such as $\mathcal{H}^p$ are dependent only on events in specific subsets (for $\mathcal{H}^p$, events in subset Σ_{IH}), this proposition will be very useful.

Proposition 12 *For DES $\mathbf{G}_H^p$ and $\mathbf{S}_H$ defined over event set Σ_{IH}, DES $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ defined over Σ_{IL_j} ($j = 1, \ldots, n$), the following properties hold:*

(a) $(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{H}^p) \wedge (P_{IH}(s) = P_{IH}(s'))] \Rightarrow s' \in \mathcal{H}^p$

(b) $(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{S}_H) \wedge (P_{IH}(s) = P_{IH}(s'))] \Rightarrow s' \in \mathcal{S}_H$

(c) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{L}_k^p) \wedge (P_{IL_k}(s) = P_{IL_k}(s'))] \Rightarrow s' \in \mathcal{L}_k^p$

(d) $(\forall k \in \{1, \ldots, n\})(\forall s, s' \in \Sigma^*)\ [(s \in \mathcal{S}_{L_k}) \wedge (P_{IL_k}(s) = P_{IL_k}(s'))] \Rightarrow s' \in \mathcal{S}_{L_k}$

Proof:

Same as proof of *Proposition 27* in [6].

■

3.1 Local Conditions for Global Controllability of the System

We now provide the controllability requirements that each level must satisfy.

Definition 6 *The* n^{th} *degree* $(n \geq 1)$ *interface system composed of DES* $\mathbf{G}_H^p, \mathbf{S}_H, \mathbf{G}_{L_1}^p, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}_{L_n}^p, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$, *is* LD level-wise controllable *with respect to the alphabet partition given by (2.1), if for all* $j \in \{1, \ldots, n\}$ *the following conditions hold:*

(I) The alphabet of $\mathbf{G}_H^p$ *and* $\mathbf{S}_H$ *is* Σ_{IH}*, the alphabet of* $\mathbf{G}_{L_j}^p$ *and* $\mathbf{S}_{L_j}$ *is* Σ_{IL_j}*, and the alphabet of* $\mathbf{G}_{I_j}$ *is* Σ_{I_j}

(II) $(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)\ \ Elig_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$

(III) $(\forall s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H)\ \ Elig_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}_H}(s)$

The above definition is identical to the level-wise controllability definition of [8], except that the system is defined over an event set that can also contain low data events. For a discussion of the individual points, please see [6, 8].

3.2 Controllability Propositions and Theorem

Our first proposition says that the j^{th} low-level supervisor, in conjunction with the j^{th} interface, is controllable for the flat plant, and that this follows from the system being LD level-wise controllable.

Proposition 13 *If n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}^p_H, \mathbf{S}_H, \mathbf{G}^p_{L_1}, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}^p_{L_n}$, $\mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ is LD level-wise controllable with respect to the alphabet partition given by (2.1), then*

$$(\forall j \in \{1, \ldots, n\})\,(\forall s \in L(\mathbf{Plant}) \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)\; Elig_{L(\mathbf{Plant})}(s) \cap \Sigma_u \;\subseteq\; Elig_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$$

Proof:

Assume that the n^{th} degree ($n \geq 1$) parallel interface system is LD level-wise controllable. **(1)**

Let $j \in \{1, \ldots, n\}$, $s \in L(\mathbf{Plant}) \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j$, and $\sigma \in \text{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u$. **(2)**

We will now show that this implies $\sigma \in \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$.

We first note that $s, s\sigma \in \mathcal{H}^p \cap [\cap_{k \in \{1,\ldots,n\}} \mathcal{L}^p_k] = L(\mathbf{Plant})$ by **(2)**. **(3)**

$\Rightarrow s, s\sigma \in \mathcal{L}^p_j$

$\Rightarrow \sigma \in \text{Elig}_{\mathcal{L}^p_j}(s) \cap \Sigma_u$

We can now conclude by *Point 2* of the LD level-wise controllability definition:

$\sigma \in \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$.

■

Our next proposition says that the high-level supervisor is controllable for the flat plant, when the plant is already under the control of the system's interfaces.

Proposition 14 *If n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}^p_H, \mathbf{S}_H, \mathbf{G}^p_{L_1}, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}^p_{L_n}$, $\mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ is LD level-wise controllable with respect to the alphabet partition given by (2.1), then*

$$(\forall s \in L(\mathbf{Plant}) \cap \mathcal{S}_H \cap \mathcal{I}) \quad Elig_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u \;\subseteq\; Elig_{\mathcal{S}_H}(s)$$

Proof:

Assume that the n^{th} degree ($n \geq 1$) parallel interface system is LD level-wise controllable. **(1)**

Let $s \in L(\mathbf{Plant}) \cap \mathcal{S}_H \cap \mathcal{I}$, and $\sigma \in \text{Elig}_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u$ **(2)**

We will now show that this implies: $\sigma \in \text{Elig}_{\mathcal{S}_H}(s)$

We first note that by **(2)**, we have:

$s, s\sigma \in \mathcal{H}^p \cap [\cap_{k \in \{1,\ldots,n\}} (\mathcal{L}^p_k \cap \mathcal{I}_k)] = L(\mathbf{Plant}) \cap \mathcal{I}$

$\Rightarrow s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H$ and $\sigma \in \text{Elig}_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u$

We can thus conclude by *Point 3* of the LD level-wise controllability definition (by **(1)**) that:

$\sigma \in \text{Elig}_{\mathcal{S}_H}(s)$ ■

This brings us to our main controllability result. The theorem below states that verifying that the system is LD level-wise controllable is sufficient to show that the flat supervisor is controllable for the flat plant. As verifying the LD level-wise controllable condition only requires a single level at a time, we can evaluate each level independently. We again do not need to construct the entire system model.

Theorem 2 *If n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}^p_H, \mathbf{S}_H, \mathbf{G}^p_{L_1}, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}^p_{L_n}, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ is LD level-wise controllable with respect to the alphabet partition given by (2.1), then*

$$(\forall s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup}))\ \mathit{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq \mathit{Elig}_{L(\mathbf{Sup})}(s)$$

Proof:

Assume that the interface system is LD level-wise controllable. **(1)**

Let $s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup})$, and $\sigma \in \text{Elig}_{L(\mathbf{Plant})}(s) \cap \Sigma_u$ **(2)**

We will now show this implies: $\sigma \in \text{Elig}_{L(\mathbf{Sup})}(s)$.

It's sufficient to show $s\sigma \in L(\mathbf{Sup}) = \mathcal{S}_H \cap [\cap_{j \in \{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)]$

We first note that by **(2)**, we have $s, s\sigma \in \mathcal{H}^p \cap [\cap_{j \in \{1,\ldots,n\}} \mathcal{L}^p_j] = L(\mathbf{Plant})$ and $s \in \mathcal{S}_H \cap [\cap_{j \in \{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)]$ **(3)**

We next note that, by **(1)**, **(2)**, and **(3)**, we can apply *Proposition 13* and conclude:

$s\sigma \in \cap_{j \in \{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)$ **(4)**

All that remains is to show that $s\sigma \in \mathcal{S}_H$

From **(3)**, we have $s \in L(\mathbf{Plant}) \cap \mathcal{S}_H \cap \mathcal{I}$

From **(3)** and**(4)**, we have $s\sigma \in L(\mathbf{Plant}) \cap \mathcal{I}$

$\Rightarrow \sigma \in \text{Elig}_{L(\mathbf{Plant}) \cap \mathcal{I}}(s) \cap \Sigma_u$

We can now apply *Proposition 14*, and conclude: $\sigma \in \text{Elig}_{\mathcal{S}_H}(s)$

$\Rightarrow s\sigma \in \mathcal{S}_H$

Combining with **(4)**, we can now conclude $s\sigma \in \mathcal{S}_H \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{S}_{L_j} \cap \mathcal{I}_j)]$, as required.

■

Chapter 4

Buffers and Verification

In this chapter, we discuss how to represent buffers using interfaces as well as investigate how to verify the new HISC low data properties.

4.1 Representing Buffers as Interfaces

Now that we have defined the new low data HISC conditions, we will discuss how to use interfaces to represent low-levels that behave as buffers. It proved to be tricky to figure out how to do this in a way that produced compact interfaces, yet did not violate *Point 5* of the LD interface consistency definition. *Point 5* says that immediately after a request event occurs, there must exist a path containing only low-level events to each answer event that can follow said request event.

If we are only interested in starting tasks and determining when the task is complete, we can use the approach shown in Fig. 2.2 which represents a two slot buffer. Request event *start* is used to add a task to the low-level's buffer. As we will start the task and then poll to determine completion, we don't need a matching answer event; thus event *start* takes us directly to a marked state (grey circle). We can then either add another task or use request event *isDone* to check to see if the task is complete.

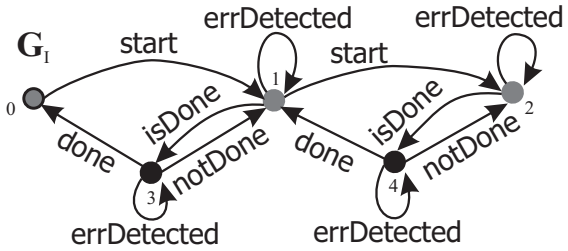
Done must be an answer event so that we can be assured that it will eventually occur, and so that the high-level can require that it occur in order to reach a marked state. If we made *done* a low data event, we would have problems with *Point I* of the LD level-wise nonblocking definition.

Response events that represent behavior that is not required to complete a task (i.e. error events, notDone etc.) should be made low data events as they typically don't need to occur to reach a marked state. Accordingly, we make *notDone* a low data event. This is key to passing *Point 5* of the LD

interface consistency definition. If we made *notDone* an answer event and the low-level finished its task while the interface was in state 1 or 2, then *notDone* would no longer be possible at state 3. This would cause *Point 5* of the LD interface consistency definition to fail. However, if *notDone* is a low data event, it is not subject to *Point 5* and need never occur as the high-level can always return to a marked state without it.

The approach shown in Fig. 2.2 works well if we only wish to report that the task is complete. If we wish to also report that an error occurred, for instance, things are more complicated. If we added a low data event, *error*, from state 3 to 0, this would cause event *done* to fail *Point 5* of the LD interface consistency definition if the low-level determined an error occurred while the interface was at state 1. The reason is that event *done* would no longer be possible at state 3.

We can use the approach shown in Fig. 4.1 to handle this situation. Here we have added a new low data event, *errDetected*, and selflooped it at states 1–4. As the occurrence of *errDetected* does not preempt the occurrence of answer event *done*, all is well. The high-level can detect the occurrence of *errDetected* and change state so that it will take the appropriate action once event *done* finally occurs. There is one caveat: event *errDetected* must be the first point at which the low-level knows that an error has occurred (i.e. up until *errDetected* occurs, it must have been still possible that the task could have completed without error). If this is not the case, then it is possible that once the interface reaches state 3 or 4, all paths to answer event *done* contain the low data event *errDetected*, violating *Point 5* of the LD interface consistency definition. As can be seen from Fig. 4.1, the new low data HISC conditions allow buffers to be modelled in a very compact manner, minimizing the increase in complexity of the high-level.



$\sum_R$={isDone,start}, $\sum_A$={done}, $\sum_{LD}$={notDone,errDetected}

Figure 4.1: Example LD Interface with Error Event

4.2 Verifying Properties

The LD interface definition can be verified using the command-pair interface algorithm from [3] with straight forward modifications. The LD level-wise controllability definition as well as *Point II* of the LD level-wise nonblocking definition can be verified using existing supervisory control algorithms after suitable definitions have been made. For *Point I* of the LD level-wise nonblocking definition, we can use a standard nonblocking algorithm modified to ignore low data events when it checks coreachability.

For *Point 3* of the LD interface consistency definition, we can use standard controllability algorithms such as TCT's **condat** function [14]. We simply define **ThePlant** = $\mathbf{G}_{I_j}$, **TheSpec** = $\mathbf{G}_H$, $\Sigma_u = \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$, and $\Sigma_c = \Sigma - \Sigma_u$. The remaining conditions of the LD interface consistency definition can be verified using the interface consistency algorithm in [3].

Chapter 5

Overview of the AIP

To demonstrate the utility of our method, we apply it to a large manufacturing system, the Atelier Inter-établissement de Productique (AIP). The AIP was originally investigated by Charbonnier et al. in [1, 2], by Leduc et al. in [6, 8] using the HISC method, and finally by Ma et al. in [10] using state tree structures and binary decision diagrams.

The AIP, shown in Fig 5.1, is an automated manufacturing system consisting of a central loop (CL) and four external loops (EL), three assembly stations(AS), an input/output (I/O) station, and four inter-loop transfer units (TU). The I/O station is where the pallets enter and leave the system. Pallets entering the system can be of type 1 or of type 2.

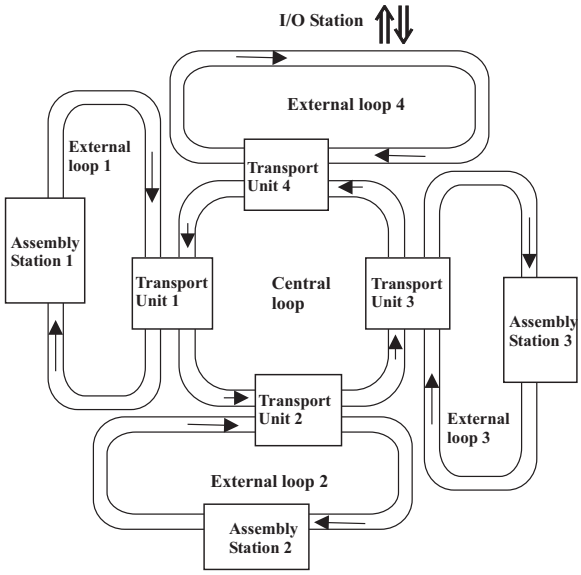


Figure 5.1: The Atelier Inter-établissement de Productique

5.1 Assembly Stations

The assembly stations are shown in Figure 5.2. Each consists of a robot to perform assembly tasks, an extractor to transfer the pallet from the conveyor loop to the robot, sensors to determine the location of the extractor, and a raising platform to present the pallet to the robot. The station also contains a pallet sensor to detect a pallet at the pallet gate, the pallet stop, and a sensor to detect when a pallet has left the station. Finally, the assembly station contains a read/write (R/W) device to read and write to the pallet's electronic label. The pallet label contains information about the pallet type, error status, and assembly status (which tasks have been performed).

Whereas the assembly stations contain the same basic components, they differ with respect to functionality. Station 1 is capable of performing two separate tasks denoted task1A and task1B, while station 2 can perform tasks task2A and task2B. Station 3 can perform all four of these tasks as well as functioning as a repair station allowing an operator to repair a damaged pallet. The assembly stations also differ with respect to reliability. Stations 1 and 2 can break down and must be repaired, while station 3 is of higher quality and is assumed never to break down. Station 3 is used as a substitute for the other stations when they are down.

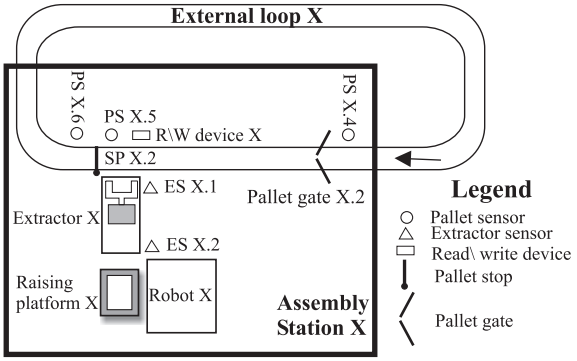


Figure 5.2: Assembly Station of External Loop $X = 1, 2, 3$

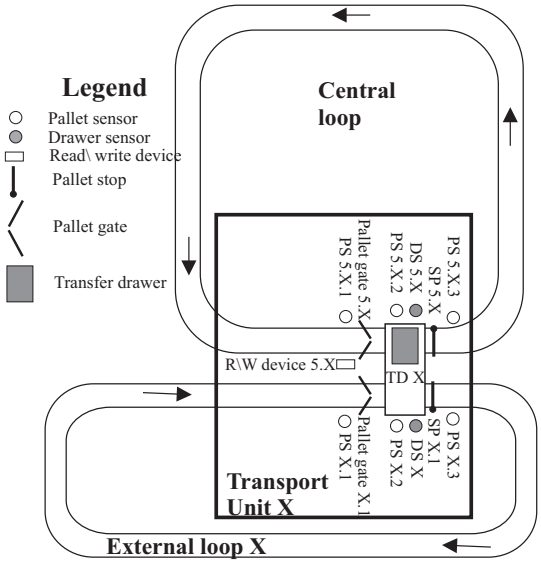


Figure 5.3: Transport Unit for External Loop $X = 1, 2, 3, 4$

5.2 Transport Units

The structure of the four identical transport units is shown in Figure 5.3. The transport units are used to transfer pallets between the central loop, and the external loops. Each one consists of a transport drawer which physically conveys the pallet between the two loops, plus sensors to determine the drawer's location. At each loop, the unit contains a pallet gate and a pallet stop, to control access to the unit from the given loop. The unit also contains multiple pallet sensors to detect when a pallet is at a gate, drawer, or has left the unit. Also, each unit contains a R/W device located before the central loop gate.

5.3 System Structure for AIP

In the original HISC design from [6, 8], the system consisted of a high-level and seven low-levels; one for each transfer unit and assembly station. Also, only one pallet at a time was allowed on external loops 1 and 2. For the full control specifications, see [6, 8].

In our new extended version, we wanted to allow two pallets at a time in external loops 1 and 2, and move the details of these loops (including the assembly stations) out of the high-level. To achieve this, we merged low-levels AS1 and TU1 into a new low-level called EL1. Similarly, AS2 and TU2 became EL2. We now have only five low-levels, as shown in Figure 5.4. As many of the DES in our example are

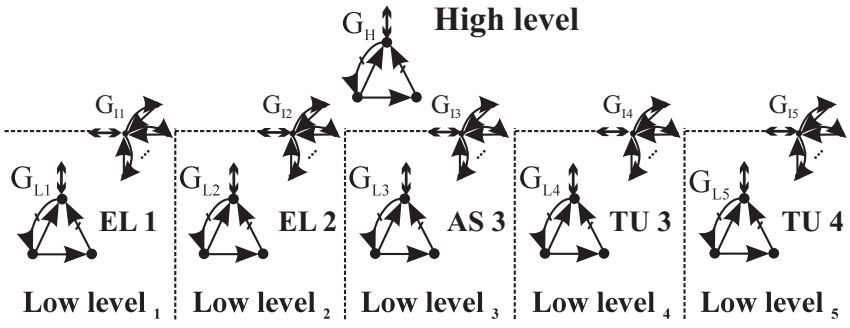


Figure 5.4: Structure of Parallel System.

unchanged from [6, 8], we will not discuss them here. We will instead focus on the DES that are new or have been changed. In the following diagrams, uncontrollable events are shown in italics; all other events are controllable. Initial states can be recognized by a thick outline, and marked states are filled.

The alphabet partition that we will use is $\Sigma := \Sigma_H \,\dot\cup\, \dot\bigcup_{j=1,\ldots,5}[\Sigma_{L_j} \dot\cup \Sigma_{R_j} \dot\cup \Sigma_{A_j} \dot\cup \Sigma_{LD_j}]$, with $\Sigma_{LD_j} = \emptyset$ for $j = 3, 4, 5$. As Σ_{L_j}, Σ_{R_j} and Σ_{A_j} are unchanged from [6, 8] for $j = 3, 4, 5$, we will not discuss them here but instead focus on the event sets that have changed. For use with the remaining event sets, we

define indices $(w, r, k) = (1, TU1, AS1), (2, TU2, AS2)$.

$$
\begin{aligned}
\Sigma_H &= \{IsPalletCL.TU1, IsPalletCL.TU2, IsPalletCL.TU3, IsPalletCL.TU4, IsPalletEL.TU3, \\
&\quad IsPalletEL.TU4, NoPalletCL.TU1, NoPalletCL.TU3, NoPalletCL.TU4, NoPalletCL.TU2, \\
&\quad NoPalletEL.TU3, NoPalletEL.TU4, PalletArvGEL\_2.AS3, QPalletAtCL.TU1, QPalletAtCL.TU2, \\
&\quad QPalletAtCL.TU3, QPalletAtCL.TU4, QPalletAtEL.TU3, QPalletAtEL.TU4\} \\
\Sigma_{L_1} &= \Sigma_{L_{AS1}} \cup \Sigma_{L_{TU1}} \\
\Sigma_{L_2} &= \Sigma_{L_{AS2}} \cup \Sigma_{L_{TU2}} \\
\Sigma_{L_{AS1}} &= \{DoneRead.AS1, DoneWrite.AS1, ExtrArvConv.AS1, ExtrArvPlatf.AS1, GClosesEL\_2.AS1, \\
&\quad GOpensEL\_2.AS1, IsType1.AS1, IsType2.AS1, MvExtrToConv.AS1, MvExtrToPlatf.AS1, \\
&\quad PalletArvGEL\_2.AS1, PalletLvAS.AS1, PalletLvGEL\_2.AS1, PArvAtExtractor.AS1, \\
&\quad PStopClosesEL\_2.AS1, PStopOpensEL\_2.AS1, QType.AS1, ReadLabel.AS1, RelPallet.AS1, \\
&\quad WrtCplT1A\_1\ B.AS1, WrtErr1A.AS1, WrtErr1B.AS1, ProcTyp1.AS1, ProcTyp2.AS1, \\
&\quad RTasksCpl.AS1, RobDwn.AS1, AssmbErrA.AS1, AssmbErrB.AS1, AError1A.AS1, \\
&\quad AError1B.AS1, RobRprCpl.AS1, RtaskCpl1A.AS1, RtaskCpl1B.AS1, StrRobRepair.AS1, \\
&\quad StrRtask1A.AS1, StrRtask1B.AS1, StrRtimer.AS1, DoRpr.AS1, ProcPallet.AS1, ASDwn.AS1, \\
&\quad ProcCpl.AS1, ProcErr.AS1\} \\
\Sigma_{L_{AS2}} &= \{DoneRead.AS2, DoneWrite.AS2, ExtrArvConv.AS2, ExtrArvPlatf.AS2, GClosesEL\_2.AS2, \\
&\quad GOpensEL\_2.AS2, IsType1.AS2, IsType2.AS2, MvExtrToConv.AS2, MvExtrToPlatf.AS2, \\
&\quad PalletArvGEL\_2.AS2, PalletLvAS.AS2, PalletLvGEL\_2.AS2, PArvAtExtractor.AS2, \\
&\quad PStopClosesEL\_2.AS2, PStopOpensEL\_2.AS2, QType.AS2, ReadLabel.AS2, RelPallet.AS2, \\
&\quad WrtCplT2A\_2B.AS2, WrtErr2A.AS2, WrtErr2B.AS2, ProcTyp1.AS2, ProcTyp2.AS2, \\
&\quad RTasksCpl.AS2, RobDwn.AS2, AssmbErrA.AS2, AssmbErrB.AS2, AError2A.AS2, \\
&\quad AError2B.AS2, RobRprCpl.AS2, RtaskCpl2A.AS2, RtaskCpl2B.AS2, StrRobRepair.AS2, \\
&\quad StrRtask2A.AS2, StrRtask2B.AS2, StrRtimer.AS2, DoRpr.AS1, ProcPallet.AS1, ASDwn.AS1, \\
&\quad ProcCpl.AS1, ProcErr.AS1\} \\
\Sigma_{L_r} &= \{DoneRead.r, DrwArvCL.r, DrwArvEL.r, DrwAtCL.r, DrwAtEL.r, GClosesCL.r, GClosesEL\_1.r, \\
&\quad GOpensCL.r, GOpensEL\_1.r, MvDrwToCL.r, MvDrwToEL.r, PalletArvDrwCL.r, \\
&\quad PalletArvDrwEL.r, PalletLvGCL.r, PalletLvGEL\_1.r, PalletLvTUAtCL.r, PalletLvTUAtEL\_1.r, \\
&\quad PStopClosesCL.r, PStopClosesEL\_1.r, PStopOpensCL.r, PStopOpensEL\_1.r, QDrwLoc.r, ReadLabel.r, \\
&\quad QError.r, NoError.r, IsErr1A.r, IsErr1B.r, IsErr2A.r, IsErr2B.r, NoErr.r, QErr1A.r, QErr1B.r, \\
&\quad QErr2A.r, QErr2B.r, QOpNeeded.r, OpNeeded.r, NotOpNeeded.r, IsCpl.r, IsType1.r, \\
&\quad IsType2.r, NotCpl.r, QCplT1A\_1B.r, QCplT2A\_2B.r, QType.r, NoTrnsfEL.r, PalletArvGEL\_1.r \\
&\quad StartTrnsfToCL.r, DelayTrnsfCplToCL.r\}
\end{aligned}
$$

$$\Sigma_{R_w} = \{\ TrnsfToEL.r,\ TrnsfELToCL.r,\ LibPallet.r,\ SkipTrnsfCL.r,\ ASUpYet.k\}$$

$$\Sigma_{A_w} = \{\ TrnsfCplToEL.r,\ TrnsfCplToCL.r,\ PalletRlsd.r,\ RobUp.k\}$$

$$\Sigma_{LD_w} = \{Rtimeout.k, NoTrnsfCL.r, NotASUp.k, \}$$

Chapter 6

AIP High-Level

The high-level subsystem, keeps track of the breakdown status of assembly stations 1 and 2, and controls the external operation of all transport units, while tracking the pallets' progress around the manufacturing system. The plant components of the high-level primarily provide a mechanism to determine information about the system, such as if a pallet is present at a given transport unit. Much of the high-level's behaviour is encapsulated in its supervisors. The high-level $\mathbf{G}_H$ consists of the synchronous product of the 10 DES listed in Figure 6.1. It is further subdivided into a plant component $\mathbf{G}_H^p$, and a supervisor component $\mathbf{S}_H$, as indicated in the diagram.

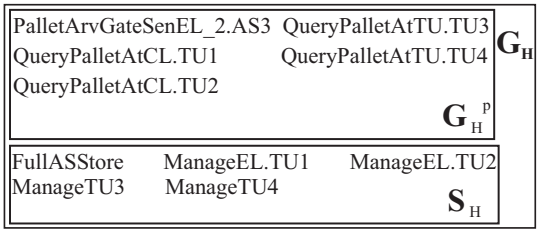


Figure 6.1: AIP High-level

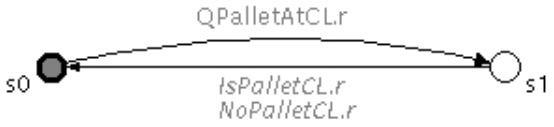


Figure 6.2: **QueryPalletAtCL.r**, with $r = \text{TU1}, \text{TU2}$

6.1 Plant Components

Plant component **QueryPalletAtCL**, shown in Figure 6.2, provides a means to determine if a pallet is waiting to enter the indicated transport unit at the central loop.

6.2 Supervisor Components

Supervisor **ManageTU3**, shown in Figure 6.3, controls the transfer of pallets between the central loop and external loop 3, and permits pallets on the central loop to pass through TU3 (to be liberated) without being transferred to its attached external loop. DES **ManageTU3** also passes the breakdown status of assembly stations 1 and 2 to TU3. If the transport unit indicates the pallet is not to be transferred to EL3, the pallet is liberated.

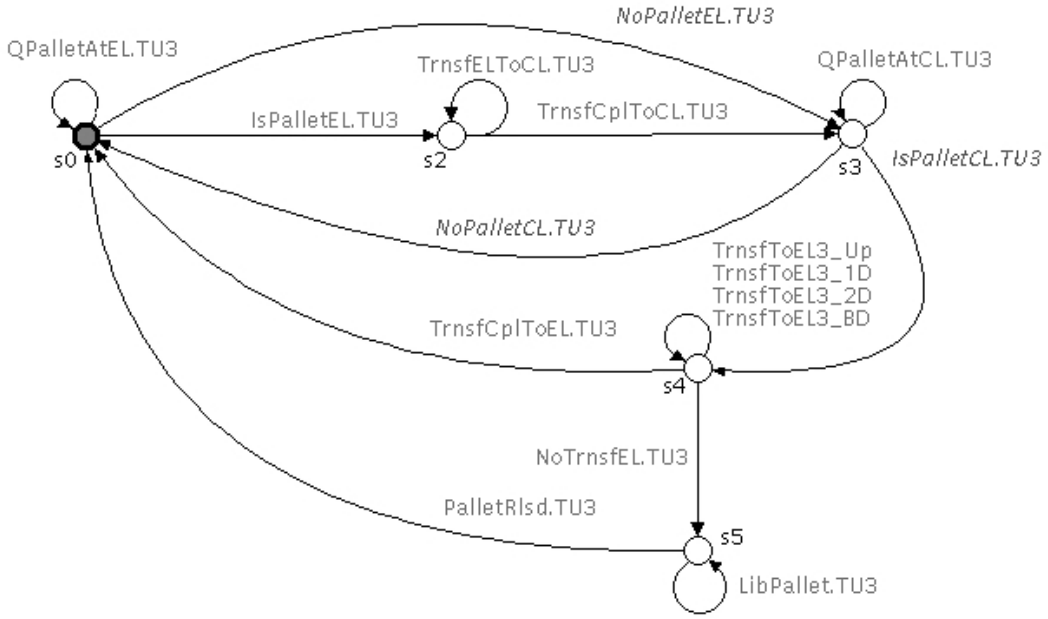


Figure 6.3: **ManageTU3**

The next new supervisor is **FullASStore**, shown in Figure 6.4. It tracks events *Rtimeout.k* and *RobUp.k* (k = AS1, AS2) to determine if assembly stations 1 or 2 are broken down. The supervisor encodes this information (i.e. which *TrnsfToEL3* event is enabled) to be passed on to TU3.

The last supervisor is **ManageEL**, shown in Figure 6.5. It controls the transfer of pallets between the central loop and the indicated external loops, and permits pallets on the central loop to pass through a transport unit (to be liberated) without being transferred to the attached external loop. Pallets are liberated if the attached external loop is at maximum capacity, the associated assembly station is down, or the transport unit determines that the pallet is not to be transferred.

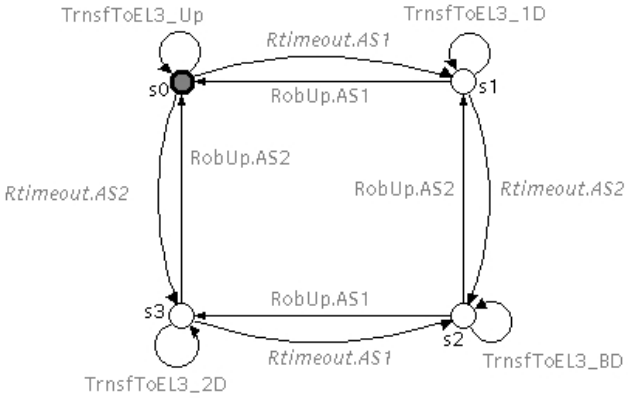


Figure 6.4: **FullASStore**

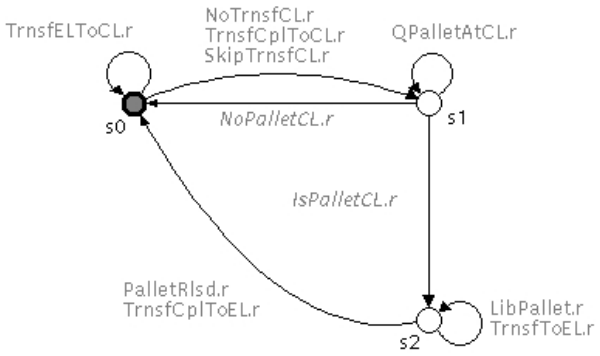


Figure 6.5: **ManageEL.r**, with $r = \mathrm{TU1}, \mathrm{TU2}$

Chapter 7

AIP Low-levels EL1 and EL2

We now describe the low-level subsystems that represent external loops 1 and 2. For EL1, this includes the behavior of assembly station 1 and transport unit 1. Similarly, EL2 includes the behavior of assembly station 2 and transport unit 2. As they are identical, we will describe them collectively as *component* m, where $m = \text{EL1}, \text{EL2}$. Component m (low-level $w = 1, 2$) contains the 49 DES listed in Figure 7.1. The diagram gives the definition of component m's subsystem $\mathbf{G}_{L_w}$, plant component $\mathbf{G}^p_{L_w}$, and supervisor component $\mathbf{S}_{L_w}$. Again, they are the synchronous product of the indicated automata.

7.1 EL Interface

Component m provides the functionality specified in its interface, shown in Figure 7.4. In essence, the EL interface represents a two slot buffer (states s0-s5, s9, s10) where pallets are moved from the central loop to the external loop via request-answer event pair *TrnsfToEL-TransfCplToEL*, processed internally by the assembly station, and then returned to the central loop via request-answer event pair *TrnsfELToCL-TrnsfCplToCL*. Request event *SkipTrnsfCL*, selflooped at states s0, s6, and s11, is used by the high-level to bypass attempts to execute event *TrnsfELToCL* as it would cause blocking at that point. At state s5, the EL is full so request-answer event pair *LibPallet-palletRlsd* allows pallets on the central loop to pass through the TU and continue moving around the CL.

At states s2 and s5, we are polling to see if a pallet on the external loop has been processed by the assembly station and is now waiting to be transferred back to the central loop. If not, low data event *noTrnsfCL* occurs and we keep waiting. Otherwise, the pallet is transferred to the CL.

States s6-s8 and s11-s14 handle the case when the assembly station is down. Low data event,

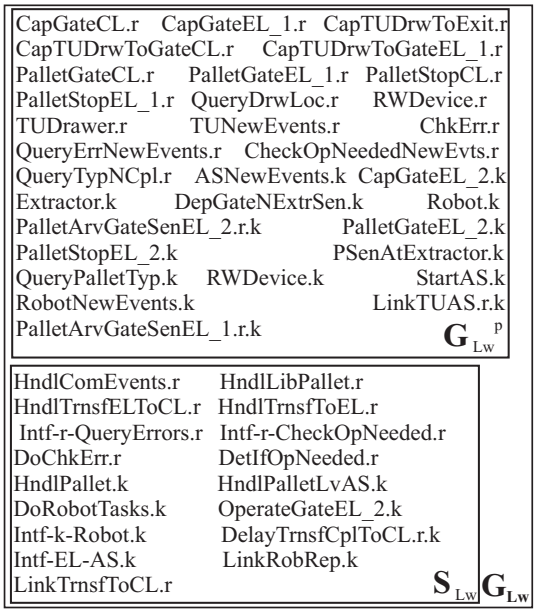


Figure 7.1: AIP Low-level m, with $(r, k) = (\text{TU1}, \text{AS1}), (\text{TU2}, \text{AS2})$

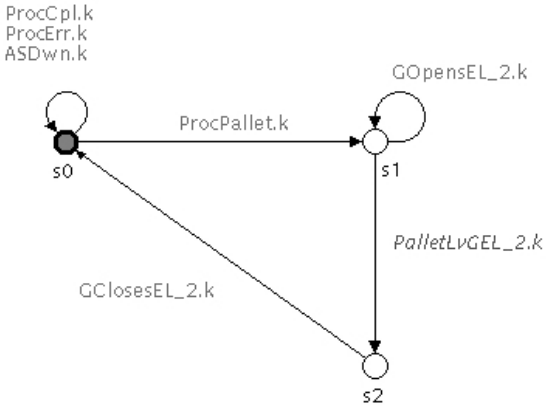


Figure 7.2: **OperateGateEL_2.k**, with $k = \text{AS1}, \text{AS2}$

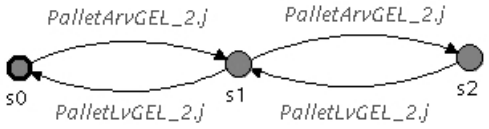


Figure 7.3: **CapGateEL_2.j**, with $j = \text{AS1}, \text{AS2}$

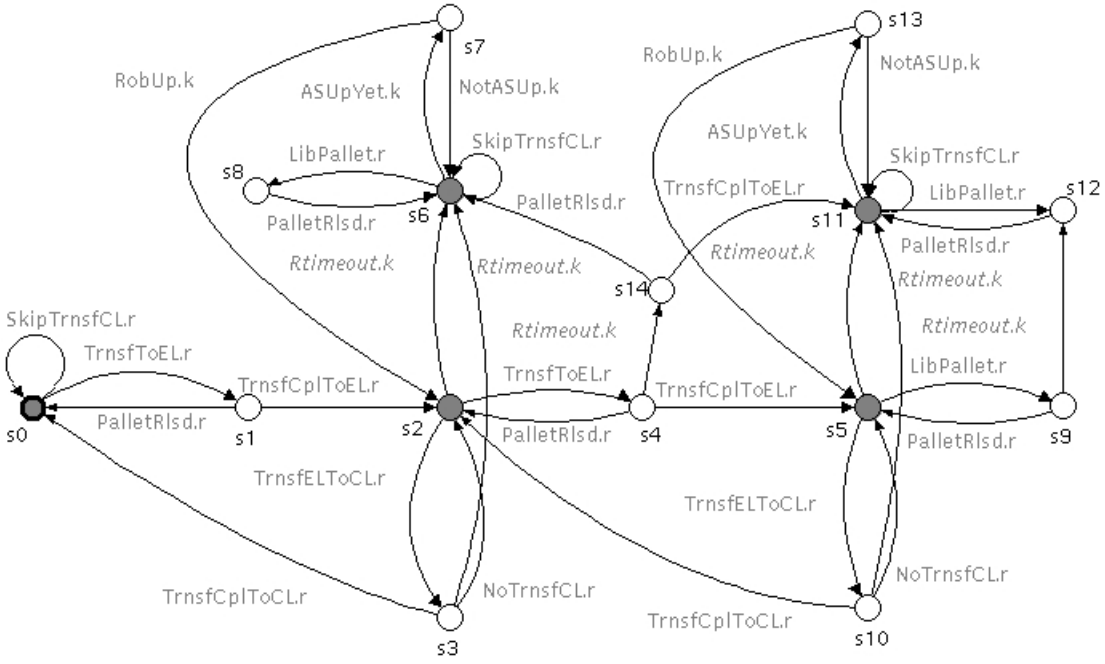


Figure 7.4: EL Interface, with $(r, k) = (\text{TU1}, \text{AS1}), (\text{TU2}, \text{AS2})$

Rtimeout, signals that the AS is down, and causes the interface to change to a new behavior where transfers to and from the external loop are suspended until the AS is repaired. Request-answer event pair *LibPallet-palletRlsd* allows pallets on the central loop to pass through the TU and continue moving around the CL. We use request event *ASUpYet* to poll if the AS is up yet, where answer event *RobUp* and low data event *NotASUp* provide the responses.

7.2 Assembly Stations

Component m contains the behavior of AS1 or AS2. An assembly station accepts the pallet at its gate, and presents the pallet to the robot for assembly. It then releases the pallet, and reports on the success of the assembly operation. If the robot breaks down, this is reported and then the assembly station waits for a repair command to return the robot to operation.

7.2.1 Plant Components

The first new plant component is **CapGateEL_2**, shown in Figure 7.3. DES **CapGateEL_2** models how many pallets can fit into gate $w.2$ at once. More importantly, it creates a dependency between pallets arriving and pallets leaving the gate.

We now discuss DES **StartAS**, shown in Figure 7.5. The DES models the fact that a pallet can not be processed, until a pallet arrives at the assembly station gate.

We next discuss DES **LinkTUAS**, **PalletArvGateSenEL_1**, and **PalletArvGateSenEL_2**, shown in Figures 7.6, 7.7, and 7.8, respectively. These DES model dependencies between the transport unit and the assembly station. DES **LinkTUAS** and **PalletArvGateSenEL_2** are concerned with pallets arriving at the external loop. DES **LinkTUAS** established a dependency between pallets arriving at the external loop and the assembly station starting to process the pallet, while **PalletArvGateSenEL_2** creates a dependency between pallets leaving the transport unit and arriving at the gate before the assembly station. DES **PalletArvGateSenEL_1** is concerned with pallets leaving the EL. It creates a dependency between pallets leaving the assembly station and arriving at the transport unit gate.

7.2.2 Supervisor Components

The first new supervisor components we discuss are **Intf-k-Robot.k** and **Intf-EL-AS.k**, shown in Figures 7.9, and 7.10. Supervisor **Intf-k-Robot.k** defines the tasks that a robot can perform while supervisor **Intf-EL-AS.k** defines the tasks that an assembly station can perform.

Figure 7.5: **StartAS.k**, with $k = \text{AS1}, \text{AS2}$

Figure 7.6: **LinkTUAS.r.k**, with $(r, k) = (\text{TU1}, \text{AS1}), (\text{TU2}, \text{AS2})$

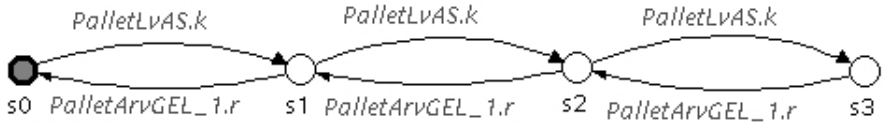


Figure 7.7: **PalletArvGateSenEL_1.r.k**, with $(r, k) = (\text{TU1}, \text{AS1}), (\text{TU2}, \text{AS2})$

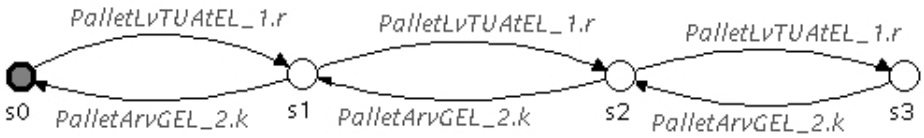


Figure 7.8: **PalletArvGateSenEL_2.r.k**, with $(r, k) = (\text{TU1}, \text{AS1}), (\text{TU2}, \text{AS2})$

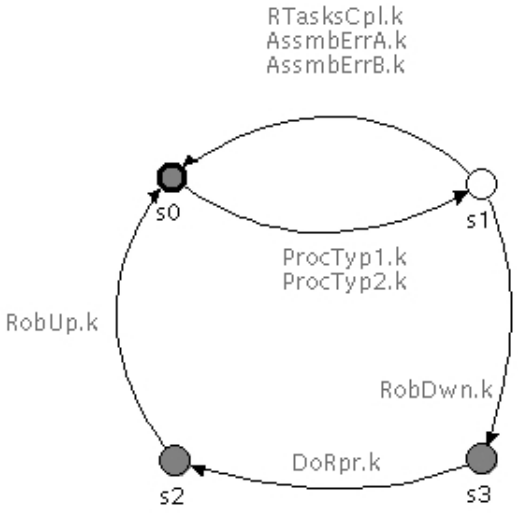


Figure 7.9: **Intf-k-Robot.k**, with $k = \text{AS1}, \text{AS2}$

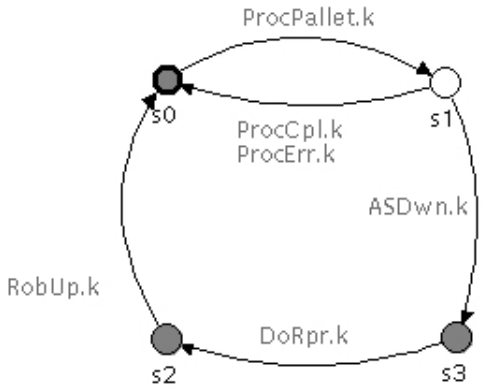


Figure 7.10: **Intf-EL-AS.k**, with $k = \text{AS1}, \text{AS2}$

We next discuss supervisors **OperateGateEL_2** and **DoRobotTasks**, shown in Figures 7.2, 7.11, and 7.12. Once a pallet has arrived at gate $w.2$, supervisor **OperateGateEL_2** opens the gate and allows the pallet to enter. This supervisor, in conjunction with **HndlPalletLvAS,** (see description in [6, 8]) guarantees that there is at most one pallet in the assembly station at a given time. Supervisor **DoRobotTasks** controls the operation of the robot. It makes sure that the assembly tasks are performed in the correct order for a given type of pallet, it reports on the success of the assembly operation, and it handles repairs when the robot breaks down.

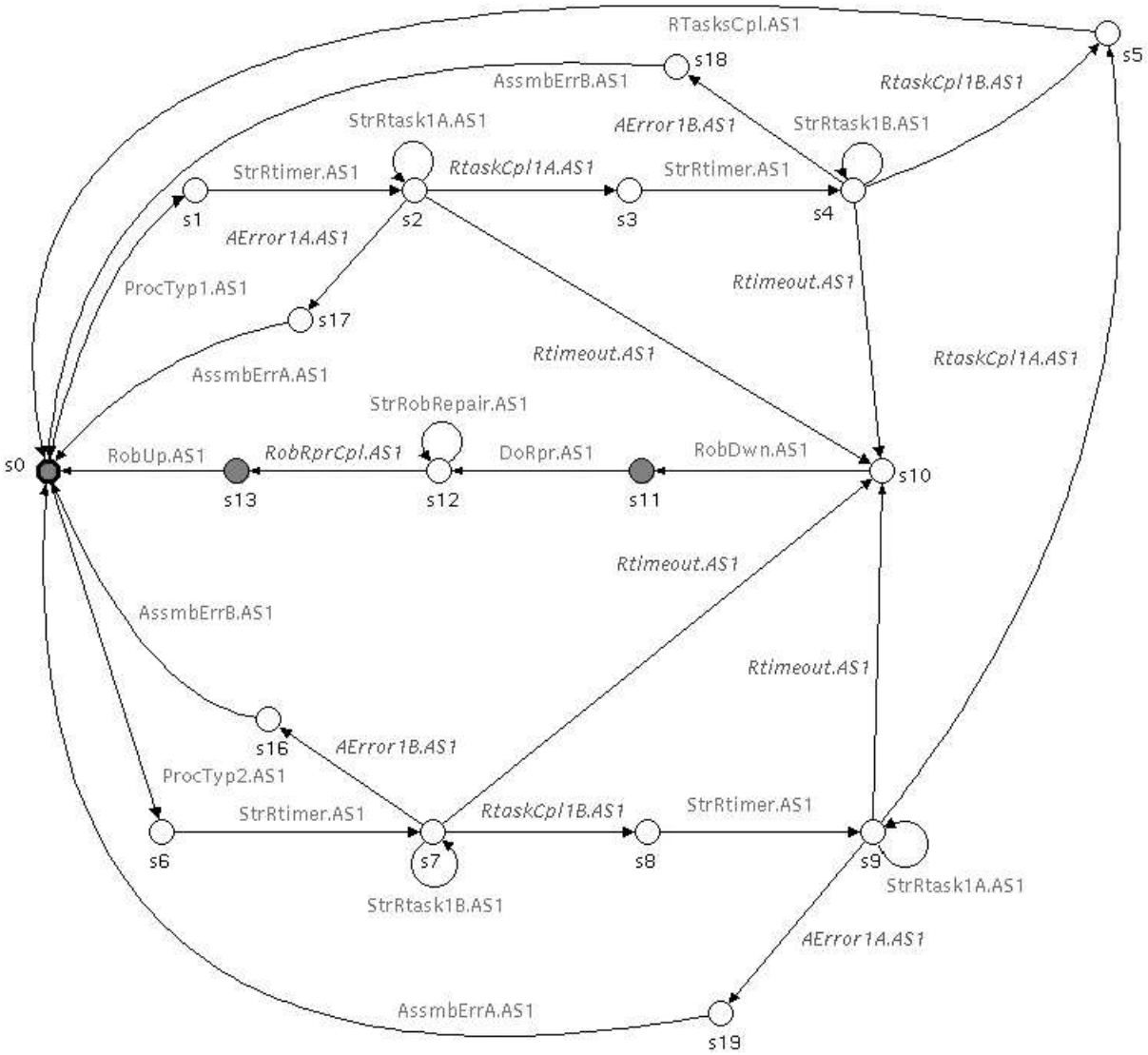


Figure 7.11: **DoRobotTasks.AS1**

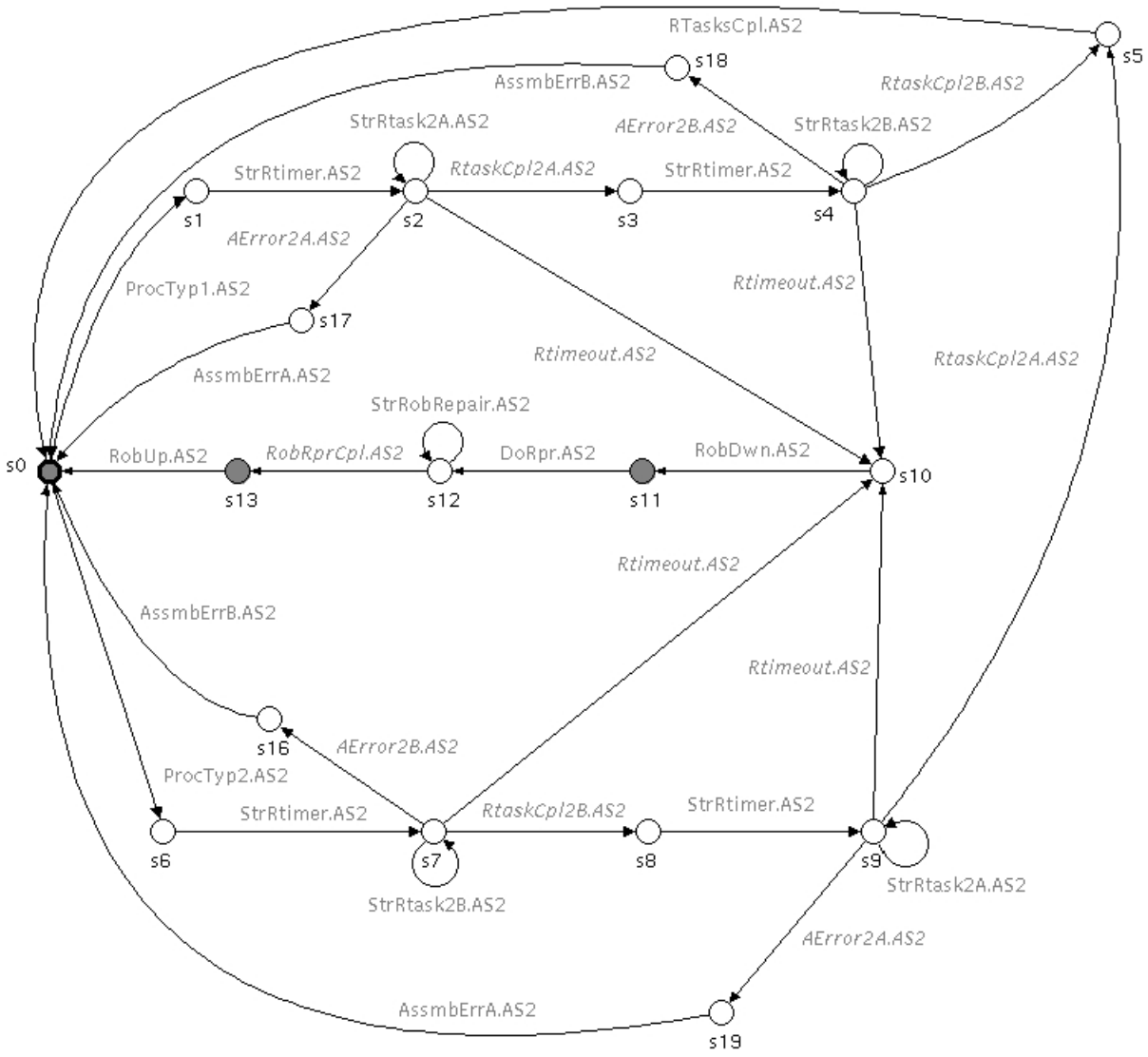


Figure 7.12: **DoRobotTasks.AS2**

We now discuss supervisor **LinkRobRep**, shown in Figure 7.13. The purpose of this supervisor is to give meaning to the low data event *NotASUp*. In the EL interface, event *NotASUp* provides a negative response to request event *ASUpYet*. DES **LinkRobRep** makes sure that *NotASUp* can occur only while the assembly station is down. Once event *RobRprCpl* has occurred, the station is back up and event *RobUp* should occur instead.

7.3 Transport Units

Component m contains the behavior of TU1 or TU2. The transport units are used to transfer pallets between the central loop, and the external loops (i.e. TU1 transfers pallets between CL and EL1). A transport unit has two entry points for pallets, the central loop gate, and the external loop gate. If a pallet is at the EL gate, the TU transfers the pallet to the central loop. If a pallet is at the CL gate, the TU can be requested to liberate the pallet (allow it to pass through and continue on the CL), or to transfer the pallet to the EL. When requested to transfer a pallet to the EL, the TU will only transfer the pallet if the pallet is undamaged and if the next assembly task required by the pallet is performed by the external loop's assembly station.

7.3.1 Plant Components

The first new plant component is **TUNewEvents**, shown in Figure 7.14. This DES simply introduces new events that are used by the transport unit. We refer to these events as *virtual events* as they don't physically exist as part of the plant. They are typically introduced to help link control behavior between supervisors and are analogous to sending messages between the supervisors to aid in synchronization. These events typically occur quickly after they are enabled, and may be implemented as the output of a synchronous state machine in digital logic, or perhaps as a shared memory variable on a computer.

We now discuss DES **CapGateCL** and **CapGateEL_1**, shown in Figures 7.15 and 7.17. Plant **CapGateCL** models how many pallets can fit into gate $5.X$ at once, where $X = 1, 2$. More importantly, it creates a dependency between events *TrnsfToEL* and *LibPallet*, and pallets leaving the gate. As events *TrnsfToEL* and *LibPallet* are dependent on a pallet arriving at gate $5.X$, this creates a dependency between pallets arriving and pallets leaving the gate. Similarly, DES **CapGateEL_1** creates a dependency between pallets arriving and pallets leaving gate $X.1$.

Figure 7.13: **LinkRobRep.k**, with k = AS1, AS2

Figure 7.14: **TUNewEvents.r**, with r = TU1, TU2

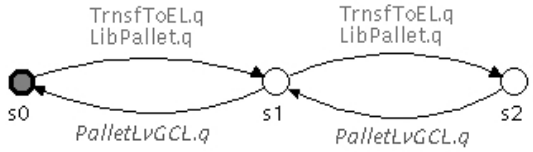


Figure 7.15: **CapGateCL.q**, with q = TU1, TU2

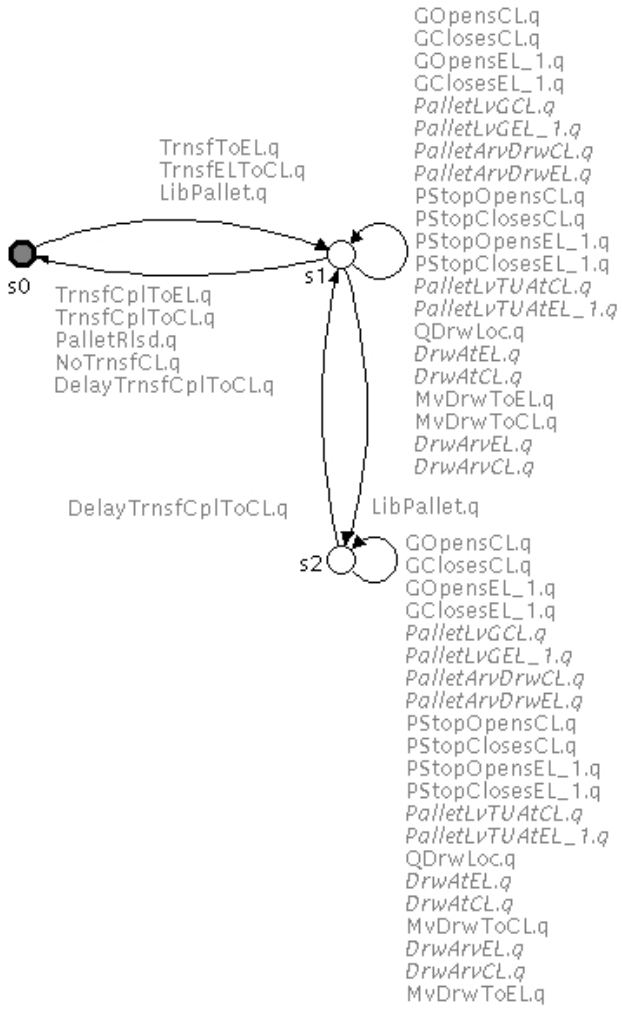


Figure 7.16: **HndlComEvents.q**, with q = TU1, TU2

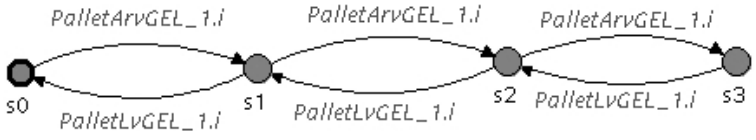


Figure 7.17: **CapGateEL_1.i**, with i = TU1, TU2

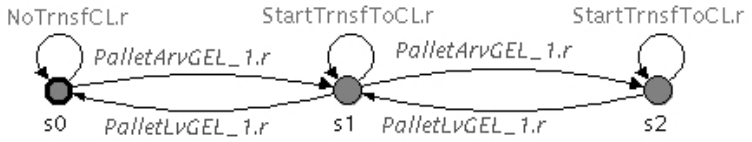


Figure 7.18: **LinkTrnsfToCL.r**, with r = TU1, TU2

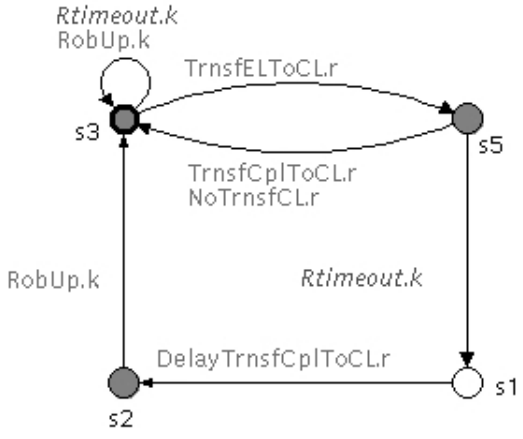


Figure 7.19: **DelayTrnsfCpltToCL.r.k**, with (r, k) = (TU1, AS1), (TU2, AS2)

7.3.2 Supervisor Components

The first new supervisor we discuss is **LinkTrnsfToCL**, shown in Figure 7.18. This supervisor defines the meaning of low data event *NoTrnfsCL* and event *StartTrnsfToCL*. It makes sure *NoTrnfsCL* is only possible when there are no pallets waiting to be transferred back to the central loop. DES **LinkTrnsfToCL** also allows event *StartTrnsfToCL* to occur once for each time a pallet arrives at gate $X.1$, where $X = 1, 2$. These events are used by supervisor **HndlTrnsfELToCL** (discussed below) to determine whether to start a pallet transfer or not (ie if there is a pallet there to transfer or not).

We next discuss DES **HndlComEvents**, **HndlLibPallet**, and **HndlTrnsfELToCL**, shown in Figures 7.16, 7.20, and 7.21. Supervisor **HndlComEvents** allows DES **HndlLibPallet, HndlTrnsfELToCL,** and **HndlTrnsfToEL** (see description in [6, 8]) to be designed independent of each other, but not cause each other to deadlock even though they use common events. Next, supervisor **HndlLibPallet** handles liberating pallets. It allows a pallet at gate $5.X$ on the central loop to pass through the transport unit without being transferred to the EL. It also works with supervisor **HndlTrnsfToEL**. If **HndlTrnsfToEL** determines that a pallet should not be transferred to the EL, it allows event NoTrnsfEL to occur which signals DES **HndlLibPallet** to liberate the pallet. Finally, supervisor **HndlTrnsfELToCL,** handles transporting pallets from EL X to the central loop.

The last new supervisor is **DelayTrnsfCpltToCL**, shown in Figure 7.19. For this design, we tried to keep the state size of interface EL as small as we could. As a result, when low data event *Rtimeout* occurred at state s3 (similarly at state s10) of the EL interface, we allowed the transfer of the pallet to the central loop to be interrupted until the assembly station was repaired. To handle this interuption, we introduced event *DelayTrnsfCplToCL* and the corresponding additional behavior into DES **HndlComEvents**, **HndlLibPallet**, and **HndlTrnsfELToCL**. This was largely to prevent blocking due to shared events. The addition of state s2 in **HndlComEvents** and state s2 and s3 in **HndlLibPallet** was to prevent the low-level from blocking the occurrence of request event *LibPallet* and thus causing *Point 4* of the LD interface consistency definition to fail. The addition of state s12 in **HndlTrnsfELToCL** gives the supervisor a place to wait and release the common events (needed by **HndlLibPallet**) while the assembly station is repaired. Supervisor **DelayTrnsfCpltToCL** was added to aid in this process. In particular, it was intended to make sure that event *DelayTrnsfCplToCL* was only possible if a *Rtimeout* event occurred during a transfer attempt to the central loop. Otherwise, DES **HndlComEvents**, **HndlLibPallet**, and **HndlTrnsfELToCL** could end up in the wrong state and cause blocking. It also ensures that event *DelayTrnsfCplToCL* occurs before event *RobUp* to ensure

that the above supervisors are in the correct state when the assembly station comes back up. As one might expect, we found that simplifying the interface caused an increase in complexity at the low-level.

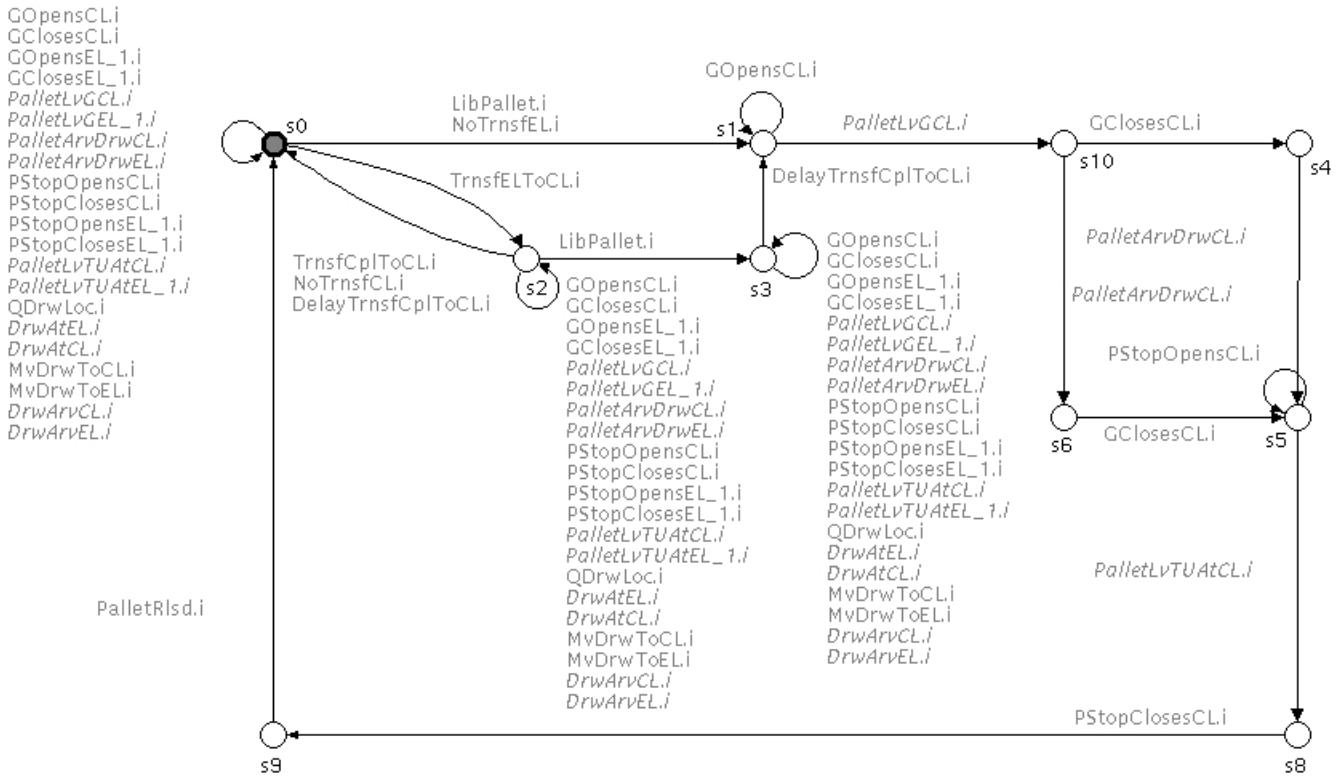


Figure 7.20: **HndlLibPallet.i**, with $i = \text{TU1}, \text{TU2}$

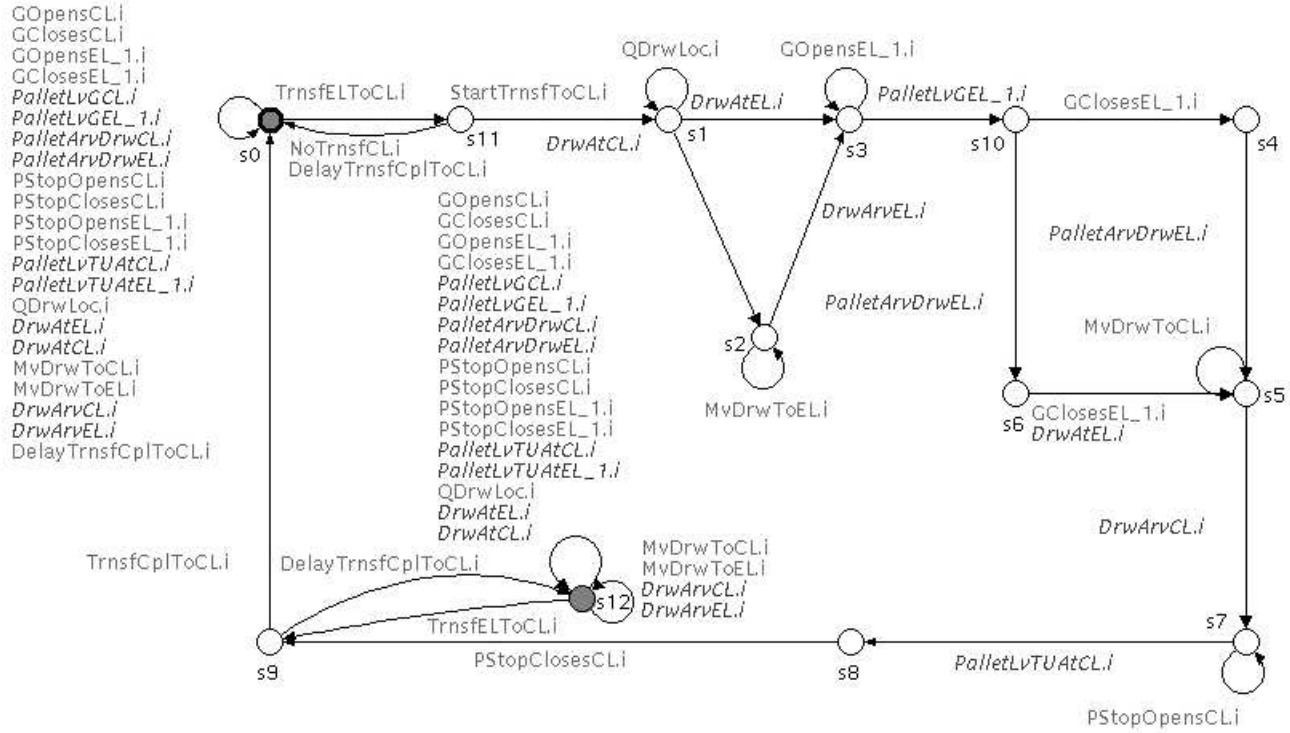


Figure 7.21: **HndlTrnsfELToCL.i**, with $i = \text{TU1}, \text{TU2}$

Chapter 8

Remaining AIP Low-levels

We now briefly discuss the remaining three low-levels, assembly station 3, and transport units 3 and 4. As these are unchanged from [6, 8], we will not discuss them in much depth.

8.1 Low-Level AS3

Component AS3 provides the functionality specified in its interface, shown in Figure 8.1. This subsystem describes the behaviour of assembly station 3, which is very similar to stations 1 and 2. The main differences are that station 3 can repair damaged pallets, is assumed not to breakdown, and can substitute for either AS1 or AS2 when they are down. Component AS3 (low-level 3) contains the 27 DES listed in Figure 8.2. The diagram gives the definition of Component AS3's subsystem $\mathbf{G}_{L_3}$, plant component $\mathbf{G}^p_{L_3}$, and supervisor component $\mathbf{S}_{L_3}$. Refer to [6, 8] for a description of these DES.

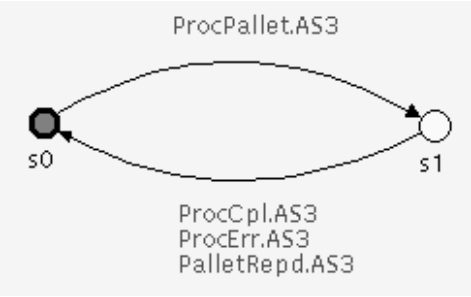


Figure 8.1: Interface to Low-level AS3

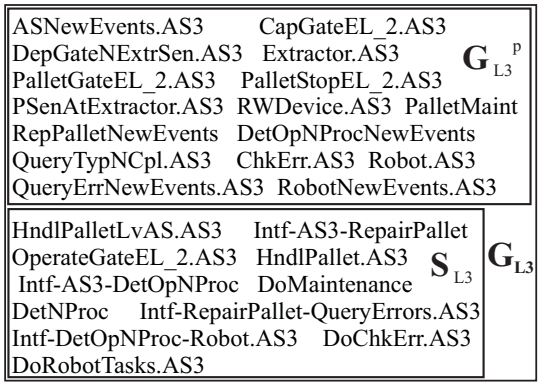


Figure 8.2: AIP Low-level AS3

8.2 Low-Level TU3

Low-level TU3 provides the functionality specified in its interface, shown in Figure 8.3. This component describes the behaviour of transport unit 3, which is very similar to TU1 and TU2. TU3 differs in how it decides if a pallet should be transferred from the central loop to external loop 3. First, all damaged pallets are to be transferred to EL3 for maintenance. Second, if an assembly station is down and it performs the next pending task for the pallet, then the pallet is to be transferred. As TU3 must know the breakdown status of assembly stations 1 and 2, this information is passed in explicitly as differently labelled request events. Component TU3 (low-level 4) contains the 29 DES listed in Figure 8.4. The diagram gives the definition of Component TU3's subsystem $\mathbf{G}_{L_4}$, plant component $\mathbf{G}^p_{L_4}$, and supervisor component $\mathbf{S}_{L_4}$. Refer to [6, 8] for a description of these DES.

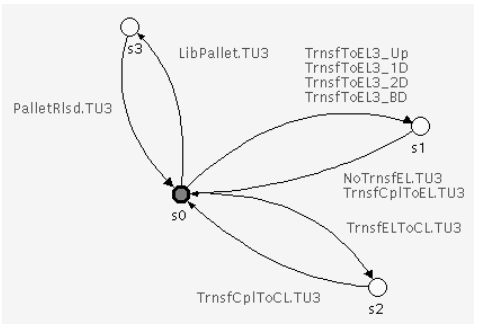


Figure 8.3: Interface to Low-level TU3

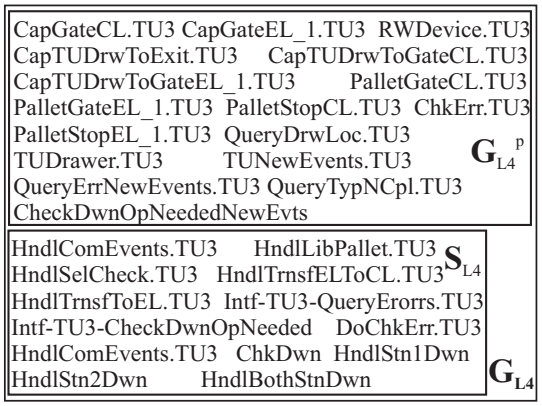


Figure 8.4: AIP Low-level TU3

8.3 Low-Level TU4

Low-level TU4 provides the functionality specified in its interface, shown in Figure 8.5, and contains the 19 DES listed in Figure 8.6. The diagram gives the definition of Component TU4's (low-level 5) subsystem $\mathbf{G}_{L_5}$, plant component $\mathbf{G}^p_{L_5}$, and supervisor component $\mathbf{S}_{L_5}$. Refer to [6, 8] for a description of these DES.

Component TU4 describes the behaviour of transport unit 4, which is very similar to TU1 and TU2. TU4 differs in how it decides if a pallet should be transferred from the central loop to its external loop (EL4), which contains the I/O station. As pallets are required to leave the system in a particular order (i.e. type 1, type 2, type 1, ...), TU4 keeps track of the type of the last pallet to be transferred to EL4 and will only transfer the current pallet if it is of the type required by the sequence, and if all required

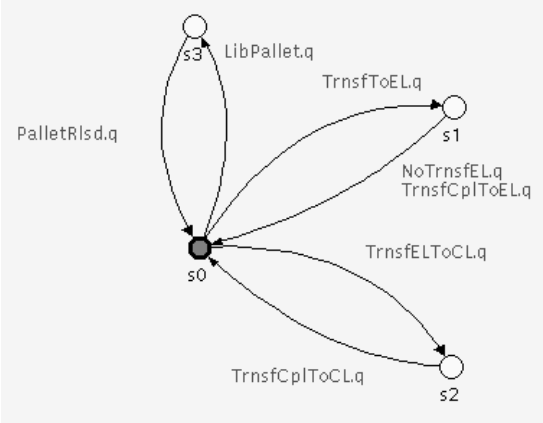


Figure 8.5: Interface to Low-level TU4, with q = TU4

CapGateCL.TU4 CapGateEL_1.TU4 QCorrectType
CapTUDrwToExit.TU4 CapTUDrwToGateCL.TU4
CapTUDrwToGateEL_1.TU4 PalletGateCL.TU4
PalletGateEL_1.TU4 PalletStopCL.TU4 $\mathbf{G}_{L5}^{p}$
PalletStopEL_1.TU4 RWDevice.TU4 TUDrawer.TU4
QueryDrwLoc.TU4 QTasksCpl TUNewEvents.TU4

HndlComEvents.TU4 HndlLibPallet.TU4 $\mathbf{S}_{L5}$
HndlTrnsfELToCL.TU4 HndlTrnsfToEL.TU4

$\mathbf{G}_{L5}$

Figure 8.6: AIP Low-level TU4

assembly tasks have been successfully performed on the pallet.

Chapter 9

AIP Example Results and Discussion

In this chapter, we presents the results of applying the low data HISC method to the AIP example as well as a discussion on how the example could have been improved.

9.1 AIP Results

After the design was completed, we applied our software to the system and determined that it was LD level-wise nonblocking and controllable, and LD interface consistent. We can thus apply *Theorems 1* and *2* and conclude that the flat system is nonblocking and that the flat supervisor is controllable for the flat plant. The computation took 104 seconds, and required 117MB. Detailed results are shown in Table 9.1, including the statespace of each level. This example has an estimated closed-loop statespace of 2.97×10^{21}. This estimate was calculated by determining the closed-loop statespace of the high-level and each low-level, and then multiplying these together to create a worst case state estimate. It's quite likely that the actual system will be considerably smaller.

Table 9.1: Low Data AIP Results

	States	
Subsystem	**|| with $\mathbf{G}_{I_j}$**	**Size of $\mathbf{G}_{I_j}$**
$\mathbf{G}_H$	518,400	7,200
EL1, **EL2**	30,185	15
AS3	203	2
TU3	204	4
TU4	152	4

On the same computer, we also verified that the AIP example from [6, 8] was level-wise nonblocking and controllable, and interface consistent. The computation took 356.76 seconds, and required 760MB. Table 9.2 shows the detailed results. As we can see, because the new low Data HISC method allowed us to shift behavior to the low-levels, we were able to decrease the statespace of the high-level by a factor of 6.4. This resulted in a 3.4 times computation speedup, and a 6.5 times reduction in memory. We believe that applying this approach to much larger examples will produce even more dramatic savings.

Table 9.2: Original AIP Example Results

	States	
Subsystem	$\|\|$ **with** $\mathbf{G}_{I_j}$	**Size of** $\mathbf{G}_{I_j}$
$\mathbf{G}_H$	3,306,240	8,192
AS1, **AS2**	120	4
AS3	203	2
TU1, **TU2**	98	4
TU3	204	4
TU4	152	4

9.2 Future Improvements

As we mentioned in Section 7.3.2, when we designed the external loop low-level and interface, our goal was to keep the EL interface as small as possible. As a result, when low data event *Rtimeout* occurred at state s3 (similarly at state s10) of the EL interface (see Figure 7.4), we allowed the transfer of the pallet to the central loop to be interrupted until the assembly station was repaired. To handle this interuption, we introduced event *DelayTrnsfCplToCL* and the corresponding additional behavior into DES **HndlComEvents**, **HndlLibPallet**, and **HndlTrnsfELToCL**. This complicated the behaviour of the low-level, making it much harder to design.

In retrospect, it might have been better to have added a new state near s3 (similarly for state s10), and have the *Rtimeout* event go to this state and wait for the result of the attempt of the transfer to the central loop. If we got a *NoTrnsCL* event, this would take us to s6. If we got a *TrnsfCplToCL* event, we would then have to go to a new structure similar to the one based at s6, but the eventual *RobUp* event would take us back to the initial state. This would allow us to remove the *DelayTrnsfCpltToCL* event and its corresponding additional behavior. This would simplify the low-level, making it much easier to design. It would also be a more natural, elegant way to model the system. Of course, the tradeoff is that the interface would be more complicated, thus likely increasing the size of the high-level.

Chapter 10

Conclusions

Hierarchical interface-based supervisory control (HISC) offers an effective method to model systems with a natural client-server architecture. As each requirement can be verified using only one subsystem, the entire plant model never needs to be constructed or traversed, offering potentially significant savings in computation.

In this paper, we extend the range of the behavior of low-levels that interfaces can model by adding a new type of event, low data events, and by relaxing some restrictions in the HISC definitions. These new changes are backwards compatible, allow for compact interfaces, and permit significant reuse of existing proofs, and software.

These changes allow us to give low-levels more freedom in sending information to the high-level, allow us to model polling behavior, and most importantly, allow us to model low-levels that behave as buffers in a natural, intuitive manner. This not only lets us greatly enrich the behaviors we can model, but enables us to move more behavior from the high-level to low-levels. Reducing the complexity of the high-level increases the size of the systems we can apply HISC to since the statesize of the high-level grows as we add more low-levels, becoming a limiting factor.

This paper also offers a tutorial on how to model buffers using interfaces. As determining how to do this in an intuitive, compact manner was nontrivial, this provides an invaluable resource for using the new approach.

Finally, our application of our method to a large manufacturing system (estimated worst case statespace on order of 10^{21}) based on the AIP example shows how we can apply the method to a real system and reduce the state size of the high-level by a factor of 6.4. This resulted in a 3.4 computation speedup, and a 6.5 times reduction in memory just from moving behavior out of the high-level.

Appendix A

Proofs of Selected Propositions

In order to make this work more readable, the proofs of some propositions were not given as the propositions were introduced. They will now be presented below.

A.1 Proof of Proposition 3

Proof for *Proposition 3* on page 12: *If the* n^{th} *degree* ($n \geq 1$) *interface system composed of DES* $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, *is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j)(\exists l \in \Sigma^*_{IL_j})\ sl \in \mathcal{H} \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

Proof:

Assume system is LD level-wise nonblocking and LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$ and $s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j$ **(2)**

Will now show this implies: $(\exists l \in \Sigma^*_{IL_j})\ sl \in \mathcal{H} \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

To do this, we will construct a suitable string l. We start by applying *Point II* of the LD level-wise nonblocking definition to conclude:

$(\exists s' \in \Sigma^*)\ ss' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$ **(3)**

As we require a string in $\Sigma^*_{IL_j}$, we take $l' = P_{IL_j}(s')$. We will now show $sl' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

From the definition of the natural projection, we have $P_{IL_j}(P_{IL_j}(s')) = P_{IL_j}(s')$

$\Rightarrow P_{IL_j}(ss') = P_{IL_j}(s)P_{IL_j}(s') = P_{IL_j}(s)P_{IL_j}(P_{IL_j}(s')) = P_{IL_j}(sP_{IL_j}(s')) = P_{IL_j}(sl')$, as the natural

projection is catenative.

We now note that as $\Sigma_{I_j} \subseteq \Sigma_{IL_j}$, it follows $P_{I_j}(s') = P_{I_j}(P_{IL_j}(s'))$.

$\Rightarrow P_{I_j}(ss') = P_{I_j}(s)P_{I_j}(s') = P_{I_j}(s)P_{I_j}(P_{IL_j}(s')) = P_{I_j}(sP_{IL_j}(s')) = P_{I_j}(sl')$, as the natural projection is catenative.

We can now use **(3)**, and apply *Proposition 1*, *points d* and *f*, and conclude:

$sl' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$ **(4)**

We now note that if $sl' \in \mathcal{H}$, we can take $l = l'$ and we have the desired result. We can thus with no loss in generality, assume: $sl' \notin \mathcal{H}$ **(5)**

We will now show this implies that string sl' is not accepted by $\mathcal{H}$ due to an event in Σ_{R_j}. We will then use this fact to construct a suitable string l.

We first observe that $s \in \mathcal{H}$, $sl' \notin \mathcal{H}$, and $l' \in \Sigma^*_{IL_j}$ implies:

$(\exists l'' \in \Sigma^*_{IL_j})(\exists \sigma \in \Sigma_{IL_j})\ (l''\sigma \leq l') \wedge (sl'' \in \mathcal{H}) \wedge (sl''\sigma \notin \mathcal{H})$ **(6)**

We note the following points, which will be used later in the proof:

- $sl'' \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j$ by the facts $sl'' \in \mathcal{H}$, $l'' < l'$, $sl' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$, and the fact $\mathcal{L}_j$ and $\mathcal{I}_j$ are closed languages. **(7)**
- $sl''\sigma \in \mathcal{I}_j$ by **(4)**, **(6)**, and fact $\mathcal{I}_j$ is closed. **(8)**

We now show that **(6)** implies $\sigma \in \Sigma_{R_j}$. We know:

- $\sigma \notin \Sigma_{L_j}$ as $\sigma \in \Sigma_{L_j}$ would imply $P_{IH}(sl''\sigma) = P_{IH}(sl'')$. Since $sl'' \in \mathcal{H}$, *Proposition 1*, *point a*, would then imply $sl''\sigma \in \mathcal{H}$ which would contradict **(6)**.
- $\sigma \notin (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j})$ as $\sigma \in (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j})$, $sl''\sigma \in \mathcal{I}_j$ (by **(8)**) and *point 3* of the LD interface consistency definition would imply $sl''\sigma \in \mathcal{H}$, which would contradict **(6)**.

As $\sigma \in \Sigma_{IL_j} = \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j} \dot{\cup} \Sigma_{L_j}$, we can thus conclude $\sigma \in \Sigma_{R_j}$ by process of elimination.

We now show that $\sigma \in \Sigma_{R_j}$ implies $sl'' \in \mathcal{I}_{m_j}$. This will allow us to use *Point 6* of the LD interface consistency definition to extend sl'' to a string marked by the j^{th} low-level.

From *Point 2* of the LD interface consistency definition, we have that DES G_{I_j} is a LD interface.

As $\sigma \in \Sigma_{R_j}$ and $sl''\sigma \in \mathcal{I}_j$ (from **(8)**), we can conclude: $P_{I_j}(sl''\sigma) = P_{I_j}(sl'')\sigma \in L(G_{I_j})$

We can thus conclude by *Points 2* and *3* of the LD interface definition that $P_{I_j}(sl'') \in L_m(G_{I_j})$ since Σ_{R_j} events are only possible at marked states.

$\Rightarrow sl'' \in \mathcal{I}_{m_j}$

From **(7)**, we have $sl'' \in \mathcal{L}_j \cap \mathcal{I}_j$ so we can now apply *Point 6* of the LD interface consistency definition and conclude:

$(\exists l''' \in \Sigma^*_{L_j})\, sl''l''' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$

As $l'' \in \Sigma^*_{IL_j}$ from **(6)**, we have $l''l''' \in \Sigma^*_{IL_j}$

We take $l = l''l'''$ and immediately have $sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$ and $l \in \Sigma^*_{IL_j}$. All that remains is to show $sl \in \mathcal{H}$

From **(6)**, we have $sl'' \in \mathcal{H}$. As $l''' \in \Sigma^*_L$, we have $P_{IH}(sl'') = P_{IH}(sl''l''') = P_{IH}(sl)$.

We can thus apply *Proposition 1*, *point a*, and conclude $sl \in \mathcal{H}$, as required..

■

A.2 Proof of Proposition 5

Proof for *Proposition 5* on page 13: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)(\forall h \in (\Sigma_{IHc} - \Sigma_{A_j})^*)\, sh \in \mathcal{I}_j \Rightarrow sh \in \mathcal{L}_j$

Proof:

Assume system is LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$, $s \in \mathcal{L}_j \cap \mathcal{I}_j$, $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*$ and assume $sh \in \mathcal{I}_j$ **(2)**

We will now show this implies that $sh \in \mathcal{L}_j$.

We have two cases: 1) $h \in (\Sigma_{IHc} - \Sigma_{I_j})^*$ and 2) $h \notin (\Sigma_{IHc} - \Sigma_{I_j})^*$

Case 1) $h \in (\Sigma_{IHc} - \Sigma_{I_j})^*$

This implies that $P_{IL_j}(sh) = P_{IL_j}(s)$. As $s \in \mathcal{L}_j$ (by **(2)**), we can now apply *Proposition 1*, *point c*,

and conclude as required: $sh \in \mathcal{L}_j$

Case 2) $h \notin (\Sigma_{IHc} - \Sigma_{I_j})^*$

By **(2)**, this implies that $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{A_j})^*$.

We first let v be the number of Σ_{R_j} events in string h ($v \geq 1$).

This implies: $(\exists h_1, \ldots, h_{v+1} \in (\Sigma_{IHc} - \Sigma_{I_j})^*)(\exists \rho_1, \ldots, \rho_v \in \Sigma_{R_j})\, h_1\rho_1 \ldots h_v\rho_v h_{v+1} = h$ **(3)**

From **(2)**, we immediately have: $sh_1\rho_1 \ldots h_v\rho_v h_{v+1} \in \mathcal{I}_j$ **(4)**

Using an inductive proof, we will now show that $sh_1\rho_1 \ldots h_v\rho_v \in \mathcal{L}_j$.

Claim to be proven

For $k \in \{1, \ldots, v\}$, $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k\rho_k \in \mathcal{L}_j$

As there is no $h_0\rho_0$, we define for simplicity $sh_0\rho_0 = s$.

We will first prove the base case, and then the general case of $k \in \{1, \ldots, v\}$. We can then conclude by induction that the claim has been proven.

Base Case:

$sh_0\rho_0 = s \in \mathcal{L}_j$ is automatic from **(2)**.

Base case complete.

Inductive Step:

Let $k \in \{1, \ldots, v\}$. Assume $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1} \in \mathcal{L}_j$. **(5)**

We will now show this implies: $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k\rho_k \in \mathcal{L}_j$

As $h_k \in (\Sigma_{IHc} - \Sigma_{I_j})^*$, we have $P_{IL_j}(h_k) = \epsilon$. We thus have $P_{IL_j}(sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k) = P_{IL_j}(sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1})$.

As $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1} \in \mathcal{L}_j$ (by **(5)**), we can now apply *Proposition 1*, *point c*, and conclude: $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k \in \mathcal{L}_j$ **(6)**

We next note that since language $\mathcal{I}_j$ is prefix closed, **(4)** implies $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k \in \mathcal{I}_j$ and

$sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k\rho_k \in \mathcal{I}_j$

Combining with **(6)**, we can apply *point 4* of the LD interface consistency definition (by **(1)**), and conclude: $sh_0\rho_0 h_1\rho_1 \ldots h_{k-1}\rho_{k-1}h_k\rho_k \in \mathcal{L}_j$

Inductive step complete.

We have now proven the *base case* and the *inductive step*. We now conclude that the *Claim* is true, by induction.

We thus take $k = v$ and we can conclude: $sh_0\rho_0 h_1\rho_1 \ldots h_v\rho_v = sh_1\rho_1 \ldots h_v\rho_v \in \mathcal{L}_j$

As $h_{v+1} \in (\Sigma_{IHc} - \Sigma_{I_j})^*$, we have $P_{IL_j}(h_{v+1}) = \epsilon$. We thus have $P_{IL_j}(sh_1\rho_1 \ldots h_v\rho_v h_{v+1}) = P_{IL_j}(sh_1\rho_1 \ldots h_v\rho_v)$.

We can now apply *Proposition 1*, *point c*, and conclude: $sh_1\rho_1 \ldots h_v\rho_v h_{v+1} = sh \in \mathcal{L}_j$

Case 2 complete.

By both *cases 1* and *2*, we have $sh \in \mathcal{L}_j$, as required.

■

A.3 Proof of Proposition 6

Proof for *Proposition 6* on page 14: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j)(\forall h \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j})$
$sh \in \mathcal{H} \cap \mathcal{I}_j \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\ (su \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}) \wedge (P_{IH}(u) = h)$

Proof:

Assume system is LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$, $s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j$, $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j}$ and assume $sh \in \mathcal{H} \cap \mathcal{I}_j$. **(2)**

We will now show this implies we can construct a string u with the desired properties.

We first note that $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j}$ implies:

$(\exists h' \in (\Sigma_{IHc} - \Sigma_{A_j})^*)(\exists \rho \in \Sigma_{R_j})(\exists h'' \in (\Sigma_{IHc} - \Sigma_{I_j})^*)(\exists \alpha \in \Sigma_{A_j})\ h'\rho h''\alpha = h$ **(3)**

We will now show that we can construct a string $l \in \Sigma_{L_j}^*$ such that $sh'\rho lh''\alpha \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$. We will also show that $P_{IH}(h'\rho lh''\alpha) = h$.

Our approach will be to show that $sh'\rho \in \mathcal{L}_j \cap \mathcal{I}_j$ and $\alpha \in \text{Elig}_{\mathcal{I}_j}(sh'\rho)$. We will then use *Point 5* of the LD interface consistent definition to construct a suitable string l.

We next note that $sh \in \mathcal{H} \cap \mathcal{I}_j$ (by **(2)**), as well as $h' \leq h$ and $h'\rho \leq h$ (by **(3)**). As $\mathcal{H}$ and $\mathcal{I}_j$ are closed languages, we can now conclude: $sh' \in \mathcal{H} \cap \mathcal{I}_j$ and $sh'\rho \in \mathcal{H} \cap \mathcal{I}_j$ **(4)**

We next note that $h'\rho \in (\Sigma_{IHc} - \Sigma_{A_j})^*$. We also have $s \in \mathcal{L}_j \cap \mathcal{I}_j$ by **(2)**. We can now apply *Proposition 5* by taking $h'\rho$ to be string h in said proposition, and conclude:

$sh'\rho \in \mathcal{L}_j$

$\Rightarrow sh'\rho \in \mathcal{L}_j \cap \mathcal{I}_j$, by **(4)**.

As $h'' \in (\Sigma_{IHc} - \Sigma_{I_j})^*$ by **(3)**, we can conclude $P_{I_j}(sh'\rho h''\alpha) = P_{I_j}(sh'\rho)P_{I_j}(h'')P_{I_j}(\alpha) = P_{I_j}(sh'\rho\alpha)$

From **(2)** and **(3)**, we have $sh'\rho h''\alpha \in \mathcal{I}_j$. We can now apply *Proposition 1*, *point e*, and conclude: $sh'\rho\alpha \in \mathcal{I}_j$

$\Rightarrow \alpha \in \text{Elig}_{\mathcal{I}_j}(sh'\rho)$.

We can now apply *Point 5* of the LD interface consistency definition (by **(1)**) and conclude:

$(\exists l \in \Sigma_{L_j}^*)\ \alpha \in \text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(sh'\rho l)$. **(5)**

$\Rightarrow sh'\rho l\alpha \in \mathcal{L}_j \cap \mathcal{I}_j$

As $h'' \in (\Sigma_{IHc} - \Sigma_{I_j})^*$ by **(3)**, we can conclude $P_{IL_j}(sh'\rho l\alpha) = P_{IL_j}(sh'\rho lh''\alpha)$ and $P_{I_j}(sh'\rho l\alpha) = P_{I_j}(sh'\rho lh''\alpha)$. We can now apply *Proposition 1*, *points c* and *e*, and conclude:

$sh'\rho lh''\alpha \in \mathcal{L}_j \cap \mathcal{I}_j$ **(6)**

We next note that by **(1)**, DES G_{I_j} is a LD interface.

As $\alpha \in \Sigma_{A_j}$ (by **(3)**), we can now conclude: $P_{I_j}(sh'\rho lh'')\alpha \in L(G_{I_j})$

$\Rightarrow P_{I_j}(sh'\rho lh''\alpha) \in L_m(G_{I_j})$ by *Points 2* and *3* of the LD interface definition.

$\Rightarrow sh'\rho lh''\alpha \in \mathcal{I}_{m_j}$ **(7)**

From **(2)** and **(3)**, we have $sh'\rho h''\alpha \in \mathcal{H}$. As $l \in \Sigma_{L_j}^*$ (by **(5)**), we can conclude:

$P_{IH}(sh'\rho h''\alpha) = P_{IH}(sh'\rho lh''\alpha)$.

We can now apply *Proposition 1*, *point a*, and conclude: $sh'\rho lh''\alpha \in \mathcal{H}$

Combining with **(6)** and **(7)**, we have $sh'\rho lh''\alpha \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$.

From **(3)** and **(5)**, we have $P_{IH}(h'\rho lh''\alpha) = P_{IH}(h'\rho h''\alpha) = h$

We next note that as $l \in \Sigma_{L_j}^*$ by **(5)**, we can conclude by **(3)** that $h'\rho lh''\alpha \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$.

We take $u = h'\rho lh''\alpha$, and the proof is complete.

■

A.4 Proof of Proposition 7

Proof for *Proposition 7* on page 14: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})(\forall h \in \Sigma_{IHc}^*.\Sigma_{A_j})$

$sh \in \mathcal{H} \cap \mathcal{I}_j \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\ (P_{IH}(u) = h) \wedge (su \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})$

Proof:

Assume system is LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$, $s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$, $h \in \Sigma_{IHc}^*.\Sigma_{A_j}$, and assume $sh \in \mathcal{H} \cap \mathcal{I}_j$ **(2)**

We will now show this implies we can construct a string u with the desired properties.

Our approach will be to break string h into substrings containing pairs of Σ_{R_j} events and Σ_{A_j} events. We will then construct u iteratively, using these substrings.

We first let v be the number of Σ_{A_j} events in string h ($v \geq 1$ as $h \in \Sigma_{IHc}^*.\Sigma_{A_j}$).

This implies: $(\exists h_1, h_2, \ldots, h_v \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{A_j})\, h_1 h_2 \ldots h_v = h$ **(3)**

We have thus broken h into v strings each containing one Σ_{A_j} event at the end of the string.

From **(2)**, we immediately have: $sh_1 h_2 \ldots h_v \in \mathcal{H} \cap \mathcal{I}_j$ **(4)**

Using an inductive proof, we will now show:

$(\exists u_0, u_1, \ldots, u_v \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\, su_0 u_1 \ldots u_v \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j} \wedge P_{IH}(u_0 u_1 \ldots u_v) = h_0 h_1 \ldots h_v$, where we define $h_0 := \epsilon$.

Claim to be proven

For $k \in \{0, 1, \ldots, v\}$, there exists $u_0, u_1, \ldots, u_k \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ such that the following are true: **(5)**

(a) $P_{IH}(u_0 u_1 \ldots u_k) = h_0 h_1 \ldots h_k$

(b) $s u_0 u_1 \ldots u_k \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$

We will first prove the initial case ($k = 0$), and then the general case of $k \in \{1, \ldots, v\}$. We can then conclude by induction that the claim has been proven.

Initial Case: $k = 0$

We take $u_0 = \epsilon$ and we immediately have $P_{IH}(u_0) = \epsilon = h_0$, and thus *Property a* of **(5)** is satisfied.

We have $s u_0 \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$ as $s u_0 = s$ and $s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$ (by **(2)**), and thus *Property b* of **(5)** is satisfied.

Initial case complete.

Inductive Step:

Let $k \in \{1, \ldots, v\}$. Assume there exists $u_0, u_1, \ldots, u_{k-1} \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ and that they satisfy *Properties a* and *b* of **(5)** when $k - 1$ is substituted for k. **(6)**

We will show that this implies there exists $u_k \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ that satisfies *Properties a* and *b* of **(5)**.

Our approach will be to apply *Proposition 6*. To do this, our first step is to show that
$s u_0 u_1 \ldots u_{k-1} h_k \in \mathcal{H} \cap \mathcal{I}_j$

We first note that $s h_0 h_1 \ldots h_k \in \mathcal{H} \cap \mathcal{I}_j$ by **(4)**, and the facts that $h_0 = \epsilon$ and $\mathcal{H}$ and $\mathcal{I}_j$ are closed languages. **(7)**

From **(6)**, we have $P_{IH}(u_0 u_1 \ldots u_{k-1}) = h_0 h_1 \ldots h_{k-1}$. From **(3)** and fact $h_0 = \epsilon$, we have $P_{IH}(h_0 h_1 \ldots h_k) = h_0 h_1 \ldots h_k$. We can thus conclude:

$$\begin{aligned} P_{IH}(s u_0 u_1 \ldots u_{k-1} h_k) &= P_{IH}(s) P_{IH}(u_0 u_1 \ldots u_{k-1}) P_{IH}(h_k) \\ &= P_{IH}(s) h_0 h_1 \ldots h_{k-1} P_{IH}(h_k) \\ &= P_{IH}(s h_0 h_1 \ldots h_k) \end{aligned} \quad (8)$$

With **(7)**, we can now apply *Proposition 1*, *point a*, and conclude:

$$su_0u_1 \dots u_{k-1}h_k \in \mathcal{H} \quad \textbf{(9)}$$

As $\Sigma_{I_j} \subseteq \Sigma_{IH}$, we can conclude by **(8)** that:

$$P_{I_j}(su_0u_1 \dots u_{k-1}h_k) = P_{I_j}(sh_0h_1 \dots h_k)$$

With **(7)**, we can now apply *Proposition 1*, *point e*, and conclude: $su_0u_1 \dots u_{k-1}h_k \in \mathcal{I}_j$

Combining with **(9)**, we have $su_0u_1 \dots u_{k-1}h_k \in \mathcal{H} \cap \mathcal{I}_j$ **(10)**

We will now show that $h_k \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j}$. It is sufficient to show that $P_{I_j}(h_k) \in \Sigma^*_{R_j}.\Sigma_{R_j}.\Sigma_{A_j}$.

We first note that $P_{I_j}(su_0u_1 \dots u_{k-1}) \in L_m(G_{I_j})$ as $su_0u_1 \dots u_{k-1} \in \mathcal{I}_{m_j}$, by **(6)**. This tells us that string $P_{I_j}(su_0u_1 \dots u_{k-1})$ takes G_{I_j} to a marked state. **(11)**

Similarly by **(10)**, we have $P_{I_j}(su_0u_1 \dots u_{k-1}h_k) \in L(G_{I_j})$. **(12)**

We now note that $h_k \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{A_j}$ (by **(3)**) implies that $P_{I_j}(h_k) \in \Sigma^*_{R_j}.\Sigma_{A_j}$. **(13)**

We next note that by **(1)**, DES G_{I_j} is a LD interface.

From *points 2-3* of the LD interface definition, we see that events in Σ_{A_j} are only possible at unmarked states, and only events in Σ_{R_j} can take us from a marked state to an unmarked state. From **(11)**, we know that string $P_{I_j}(su_0u_1 \dots u_{k-1})$ took G_{I_j} to a marked state. As $P_{I_j}(h_k)$ ends with an answer event and does not contain any low data events, then $P_{I_j}(h_k)$ must contain a Σ_{R_j} directly before the answer event.

$\Rightarrow P_{I_j}(h_k) \in \Sigma^*_{R_j}.\Sigma_{R_j}.\Sigma_{A_j}$, by **(13)**.

$\Rightarrow h_k \in (\Sigma_{IHc} - \Sigma_{A_j})^*.\Sigma_{R_j}.(\Sigma_{IHc} - \Sigma_{I_j})^*.\Sigma_{A_j}$

We may now apply *Proposition 6* by taking $su_0u_1 \dots u_{k-1}$ to be string s, and h_k to be string h in said proposition. We thus conclude:

$$(\exists u' \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\,(su_0u_1 \dots u_{k-1}u' \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_m) \ \wedge \ (P_{IH}(u') = h_k)$$

We can also conclude $P_{IH}(u_0u_1 \dots u_{k-1}u') = h_0h_1 \dots h_k$ by **(6)** and the fact P_{IH} is catenative.

We now take $u_k = u'$ and we have $u_k \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ and we have it satisfying *Properties a* and *b* of **(5)**.

Inductive step complete.

We have now proven the *initial case* and the *inductive step*. We now conclude that the *Claim* is true, by induction.

We thus take $k = v$ and have $u_0, u_1, \ldots, u_v \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$, $su_0u_1 \ldots u_v \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$, and $P_{IH}(u_0u_1 \ldots u_v) = h_0h_1 \ldots h_v = h$

We thus take $u = u_0u_1 \ldots u_v$, and the proof is complete.

■

A.5 Proof of Proposition 8

Proof for *Proposition 8* on page 14: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall j \in \{1, \ldots, n\})(\forall s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j})(\forall h \in \Sigma^*_{IHc})$

$sh \in \mathcal{H}_m \cap \mathcal{I}_{m_j} \Rightarrow (\exists u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\,(su \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}) \wedge (P_{IH}(u) = h)$

Proof:

Assume system is LD interface consistent. **(1)**

Let $j \in \{1, \ldots, n\}$, $s \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$, $h \in \Sigma^*_{IHc}$, and assume $sh \in \mathcal{H}_m \cap \mathcal{I}_{m_j}$ **(2)**

We will now show this implies we can construct a string u with the desired properties.

We have two cases to examine: **I)** $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*$ **II)** $h \notin (\Sigma_{IHc} - \Sigma_{A_j})^*$

Case I) $h \in (\Sigma_{IHc} - \Sigma_{A_j})^*$

From **(2)**, we have $s \in \mathcal{L}_j \cap \mathcal{I}_j$ and $sh \in \mathcal{I}_j$. We can thus apply *Proposition 5*, and conclude:

$sh \in \mathcal{L}_j$

Combining with **(2)**, we thus have $sh \in \mathcal{H}_m \cap \mathcal{L}_j \cap \mathcal{I}_{m_j}$

We can now apply *Proposition 4*, taking sh to be string s in said proposition, and conclude:

$(\exists l \in \Sigma^*_{L_j})\ shl \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_m$

We take $u = hl$ and we immediately have $u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$, $su \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$, and $P_{IH}(u) = h$.

Case I complete.

Case II) $h \notin (\Sigma_{IHc} - \Sigma_{A_j})^*$ (i.e. string h contains one or more events from Σ_{A_j})

This implies $P_{I_j}(h) \in \Sigma^*_{I_j}.\Sigma_{A_j}.\Sigma^*_{I_j}$ **(3)**

Our approach will be first to apply *Proposition 7* to enable us to construct a string u' such that $su' \in \mathcal{L} \cap \mathcal{H} \cap \mathcal{I}_j$. We will next use string u' to construct a string u, and then use *Proposition 4* to show that $su \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

Our first step will to be to construct a suitable string $h' \leq h$, $h' \in \Sigma^*_{IHc}.\Sigma_{A_j}$, so that we can apply *Proposition 7*.

From **(3)**, it follows that:

$$(\exists h' \in \Sigma^*_{IHc}.\Sigma_{A_j})(h'' \in (\Sigma_{IHc} - \Sigma_{A_j})^*)\, h'h'' = h \quad \textbf{(4)}$$

We can now conclude $sh' \in \mathcal{H} \cap \mathcal{I}_j$ as $sh \in \mathcal{H}_m \cap \mathcal{I}_{m_j}$ (by **(2)**) and the fact that $\mathcal{H}$ and $\mathcal{I}_j$ are closed languages.

We can now apply *Proposition 7* by taking h' to be string h in said proposition. We can then conclude:

$$(\exists u' \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*)\,(P_{IH}(u') = h') \wedge (su' \in \mathcal{H} \cap \mathcal{L}_j \cap \mathcal{I}_j) \quad \textbf{(5)}$$

We note for future use that $P_{IH}(u') = h'$ and the fact that $\Sigma_{I_j} \subseteq \Sigma_{IH}$ implies that $P_{I_j}(u') = P_{I_j}(h')$. **(6)**

We will now show that $su'h'' \in \mathcal{L}_j \cap \mathcal{H}_m \cap \mathcal{I}_{m_j}$ so that we can apply *Proposition 4* .

From **(5)**, we have $su' \in \mathcal{L}_j \cap \mathcal{I}_j$. From **(4)**, we have $h'' \in (\Sigma_{IHc} - \Sigma_{A_j})^*$. In order to apply *Proposition 5*, we only still need to show $su'h'' \in \mathcal{I}_j$.

From **(2)** and **(4)**, we have $sh'h'' \in \mathcal{I}_j$. From **(6)**, we have $P_{I_j}(u') = P_{I_j}(h')$.

$$\Rightarrow P_{I_j}(su'h'') = P_{I_j}(s)P_{I_j}(u')P_{I_j}(h'') = P_{I_j}(s)P_{I_j}(h')P_{I_j}(h'') = P_{I_j}(sh'h'') \quad \textbf{(7)}$$

We can now apply *Proposition 1*, *point e*, and conclude: $su'h'' \in \mathcal{I}_j$

We can now apply *Proposition 5* taking su' and h'' to be strings s and h in said proposition, respectively. We can now conclude: $su'h'' \in \mathcal{L}_j$ **(8)**

From **(2)**, we have $sh \in \mathcal{H}_m \cap \mathcal{I}_{m_j}$. From **(4)**, we can conclude:

$$sh'h'' \in \mathcal{H}_m \cap \mathcal{I}_{m_j} \quad \textbf{(9)}$$

As $P_{IH}(u') = h'$ (from **(5)**), we have:

$P_{IH}(sh'h'') = P_{IH}(s)P_{IH}(h')P_{IH}(h'') = P_{IH}(s)P_{IH}(u')P_{IH}(h'') = P_{IH}(su'h'')$.

We can now apply *Proposition 1*, *point b*, and conclude $su'h'' \in \mathcal{H}_m$ **(10)**

From **(7)**, we can now apply *Proposition 1*, *point f*, and conclude $su'h'' \in \mathcal{I}_{m_j}$

Combining with **(8)** and**(10)**, we can conclude:

$su'h'' \in \mathcal{L}_j \cap \mathcal{H}_m \cap \mathcal{I}_{m_j}$

We can now apply *Proposition 4* by taking $su'h''$ to be string s in that proposition. We thus conclude:

$(\exists l \in \Sigma^*_{L_j})\, su'h''l \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$ **(11)**

We thus take $u = u'h''l$ and we have $u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ (by **(4)**, **(5)**, and **(11)**), $P_{IH}(u) = P_{IH}(u'h''l) = h'h''\epsilon = h$ and $su \in \mathcal{H}_m \cap \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

Case II is now complete.

By **Cases I** and **II**, we now have constructed a string $u \in (\Sigma_{IHc} \cup \Sigma_{L_j})^*$ such that $P_{IH}(u) = h$ and $su \in \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_{m_j}$, as required.

■

A.6 Proof of Proposition 9

Proof for *Proposition 9* on page 15: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (2.1), then*

$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)])\,(\exists l \in \Sigma^*_{IL})\ (sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])$

Proof:

Assume system is LD level-wise nonblocking and LD interface consistent. **(1)**

Let $s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)]$ **(2)**

We will now show this implies: $(\exists l \in \Sigma^*_{IL})\ sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

To do this, we will use an inductive proof. We define $\Sigma^*_{IL_0} = \Sigma^*$, $\mathcal{L}_{m_0} = \mathcal{I}_{m_0} = \Sigma^*$ and $\mathcal{L}_{n+1} = \mathcal{I}_{n+1} =$

Σ^*. Here we are using the fact that intersection with Σ^* is the identity operator as all other languages are subsets of Σ^*. This is to handle the boundary cases of $k = 0$ and $k = n$ in order to avoid intersection with $\emptyset$.

Claim to be proven:

For $k \in \{0, 1, \ldots, n\}$, there exist strings $l_i \in \Sigma^*_{IL_i}$, $i = 0, 1, \ldots, k$, such that:

$$sl_0l_1 \ldots l_k \in \mathcal{H} \cap [\cap_{v\in\{0,1,\ldots,k\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k+1,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w)] \quad \textbf{(3)}$$

We will first prove the initial case, $k = 0$, and then the general case of $k \in \{1, \ldots, n\}$. We can then conclude by induction that the claim has been proven.

Initial Case: $k = 0$

We take $l_0 = \epsilon \in \Sigma^*_{IL_0}$. We immediately have $sl_0 = s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_j \cap \mathcal{I}_j)]$ by **(2)**.

We have automatically $sl_0 \in (\mathcal{L}_{m_0} \cap \mathcal{I}_{m_0} \cap \mathcal{L}_{n+1} \cap \mathcal{I}_{n+1}) = \Sigma^*$.

Initial case complete.

Inductive Step:

Let $k \in \{1, \ldots, n\}$. Assume there exist strings $l_i \in \Sigma^*_{IL_i}$, $i = 0, 1, \ldots, k-1$, and that they satisfy **(3)** when $k-1$ is substituted for k.

$$\Rightarrow sl_0l_1 \ldots l_{k-1} \in \mathcal{H} \cap [\cap_{v\in\{0,1,\ldots,k-1\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w) \quad \textbf{(4)}$$

We will show this implies that we can construct string $l_k \in \Sigma^*_{IL_k}$, such that $sl_0l_1 \ldots l_k \in \mathcal{H} \cap [\cap_{v\in\{0,1,\ldots,k\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k+1,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w)]$

Our approach will be to apply *Proposition 3* with respect to $j = k$.

We first note that **(4)** implies $sl_0l_1 \ldots l_{k-1} \in \mathcal{H} \cap [\cap_{w\in\{1,\ldots,n\}}(\mathcal{L}_w \cap \mathcal{I}_w)]$

We can now apply *Proposition 3* by taking $j = k$ and $sl_0l_1 \ldots l_{k-1}$ to be string s in that proposition. We thus conclude:

$$(\exists l_k \in \Sigma^*_{IL_k})\, sl_0l_1 \ldots l_{k-1}l_k \in \mathcal{H} \cap \mathcal{L}_{m_k} \cap \mathcal{I}_{m_k} \quad \textbf{(5)}$$

We will now show that this implies:

$$sl_0l_1 \ldots l_{k-1}l_k \in [\cap_{v\in\{0,1,\ldots,k-1\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k+1,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w)]$$

From **(4)**, we have $sl_0l_1 \ldots l_{k-1} \in [\cap_{v\in\{0,1,\ldots,k-1\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k+1,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w)$

We next note that as $l_k \in \Sigma^*_{IL_k}$, we have for $j = 1, \ldots, k-1, k+1, \ldots, n$ that $P_{IL_j}(l_k) = \epsilon$

$$\Rightarrow P_{IL_j}(sl_0l_1 \ldots l_{k-1}l_k) = P_{IL_j}(sl_0l_1 \ldots l_{k-1}) \quad \textbf{(6)}$$

We can thus apply *Proposition 1*, *points c-d*, and conclude:

$$sl_0l_1 \ldots l_{k-1}l_k \in [\cap_{v\in\{1,\ldots,k-1\}}\mathcal{L}_{m_v}] \cap [\cap_{w\in\{k+1,\ldots,n\}}\mathcal{L}_w] \quad \textbf{(7)}$$

As $\Sigma_{I_j} \subseteq \Sigma_{IL_j}$ $(j = 1, \ldots, k-1, k+1, \ldots, n)$, it follows from **(6)** that $P_{I_j}(sl_0l_1 \ldots l_{k-1}l_k) = P_{I_j}(sl_0l_1 \ldots l_{k-1})$.

We can thus apply *Proposition 1*, *points e-f*, and conclude:

$$sl_0l_1 \ldots l_{k-1}l_k \in [\cap_{v\in\{1,\ldots,k-1\}}\mathcal{I}_{m_v}] \cap [\cap_{w\in\{k+1,\ldots,n\}}\mathcal{I}_w] \quad \textbf{(8)}$$

We next note that we have automatically $sl_0l_1 \ldots l_{k-1}l_k \in (\mathcal{L}_{m_0} \cap \mathcal{I}_{m_0} \cap \mathcal{L}_{n+1} \cap \mathcal{I}_{n+1}) = \Sigma^*$.

Combining with **(5)**, **(7)**, and **(8)**, we thus have as required:

$$sl_0l_1 \ldots l_k \in \mathcal{H} \cap [\cap_{v\in\{0,1,\ldots,k\}}(\mathcal{L}_{m_v} \cap \mathcal{I}_{m_v})] \cap [\cap_{w\in\{k+1,\ldots,n+1\}}(\mathcal{L}_w \cap \mathcal{I}_w)]$$

Inductive step complete.

We have now proven the *initial case* and the *inductive step*. We now conclude that the *claim* is true, by induction.

Taking $k = n$ and using fact that $l_0 = \epsilon$, we thus can conclude there exists strings $l_i \in \Sigma^*_{IL_i}$, $i = 1, \ldots, n$, such that: $sl_1 \ldots l_n \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

We thus take $l = l_1 \ldots l_n$ and we have $l \in \Sigma^*_{IL}$ and $sl \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$, as required.

■

A.7 Proof of Proposition 11

Proof for *Proposition 11* on page 16: *If the n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is LD interface consistent with respect to the alphabet partition given by*

(2.1), then

$(\forall s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})])(\forall h \in \Sigma^*_{IHc})$

$sh \in \mathcal{H}_m \cap \mathcal{I}_m \Rightarrow (\exists u \in \Sigma^*)\,(su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]) \wedge (P_{IH}(u) = h)$

Proof:

Assume system is LD interface consistent. **(1)**

Let $s \in \mathcal{H} \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$, $h \in \Sigma^*_{IHc}$, and assume $sh \in \mathcal{H}_m \cap \mathcal{I}_m$. **(2)**

We will now show this implies: $(\exists u \in \Sigma^*)\,(su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]) \wedge (P_{IH}(u) = h)$

To do this, we will use string h and apply *Proposition 8* iteratively, to construct n strings labelled $u_i \in (\Sigma_{IHc} \cup \Sigma_{L_i})^*$, $i = 1, \ldots, n$, with the properties $su_i \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}] \cap \mathcal{L}_{m_i}$ (i.e. string u_i takes the high-level and the i^{th} low-level subsystem to a marked state) and $P_{IH}(u_i) = h$ (string h is the high-level image of each string u_i). We will then use these n strings to construct the desired string u.

Iterative step:

For each $i \in \{1, \ldots, n\}$, construct u_i as follows:

To apply *Proposition 8* for index $j = i$, we need to show: $s \in \mathcal{H} \cap \mathcal{L}_i \cap \mathcal{I}_{m_i}$, and $sh \in \mathcal{H}_m \cap \mathcal{I}_{m_i}$. This follows immediately from **(2)** since $\mathcal{L}_{m_i} \subseteq \mathcal{L}_i$ and $\mathcal{I}_m = \cap_{k\in\{1,\ldots,n\}}\mathcal{I}_{m_k}$.

We can thus apply *Proposition 8* and conclude:

$(\exists u_i \in (\Sigma_{IHc} \cup \Sigma_{L_i})^*)\,(su_i \in \mathcal{H}_m \cap \mathcal{L}_{m_i} \cap \mathcal{I}_{m_i}) \wedge (P_{IH}(u_i) = h)$ **(3)**

We now need to show: $su_i \in \cap_{j\in\{1,\ldots,i-1,i+1,\ldots,n\}}\mathcal{I}_{m_j}$

From **(2)** we have $sh \in \cap_{j\in\{1,\ldots,i-1,i+1,\ldots,n\}}\mathcal{I}_{m_j}$ and from **(3)** we have $P_{IH}(u_i) = h$.

It thus follows that $P_{IH}(su_i) = P_{IH}(s)P_{IH}(u_i) = P_{IH}(s)h = P_{IH}(s)P_{IH}(h) = P_{IH}(sh)$, as P_{IH} is catenative and $h \in \Sigma^*_{IH}$.

Since for $j = 1, \ldots, i-1, i+1, \ldots, n$ we have $\Sigma_{I_j} \subseteq \Sigma_{IH}$, it follows that $P_{I_j}(su_i) = P_{I_j}(sh)$.

We can thus conclude by *Proposition 1, point f*, that: $su_i \in \cap_{j\in\{1,\ldots,i-1,i+1,\ldots,n\}}\mathcal{I}_{m_j}$.

Combining with **(3)**, we have:

$su_i \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}] \cap \mathcal{L}_{m_i}$, as required.

Iterative step complete.

Now that we have completed the *iterative step*, we have shown the following:

$(\forall i \in \{1,\ldots,n\})(\exists u_i \in (\Sigma_{IHc} \cup \Sigma_{L_i})^*)\,(su_i \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}] \cap \mathcal{L}_{m_i}) \wedge (P_{IH}(u_i) = h)$ **(3)**

We will now use this information to construct a string $u \in \Sigma^*$ with the property:

$(su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]) \wedge (P_{IH}(u) = h)$

We first need to define the nature projections $P_{IHL_i} : \Sigma^* \to (\Sigma_{IH} \cup \Sigma_{L_i})^*$, $i = 1,\ldots,n$.

We take u to be any string in set $\cap_{i\in\{1,\ldots,n\}} P^{-1}_{IHL_i}(u_i)$ **(4)**

We know that the set is non-empty for the following reasons:

- For each $i \in \{1,\ldots,n\}$, we have $u_i \in (\Sigma_{IHc} \cup \Sigma_{L_i})^*$.
- The only events strings u_i and u_k $(i,k \in \{1,\ldots,n\}, i \neq k)$ have in common are $\sigma \in \Sigma_{IH}$.
- All strings u_i agree on common events as $P_{IH}(u_i) = h$

From **(4)**, we have $(\forall i \in \{1,\ldots,n\})P_{IHL_i}(u) = u_i$. As $\Sigma_{IH} \subseteq (\Sigma_{IH} \cup \Sigma_{L_i})$ and $h \in \Sigma^*_{IH}$ (by **(2)**), we can conclude:

$P_{IH}(u) = P_{IH}(u_i) = h = P_{IH}(h).$ **(5)**

$\Rightarrow P_{IH}(su) = P_{IH}(sh)$ **(6)**

From **(2)**, we have: $sh \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}]$. We can now apply *Proposition 1*, *point b*, and conclude:

$su \in \mathcal{H}_m$ **(7)**

As $\Sigma_{I_i} \subseteq \Sigma_{IH}$ for $i = \{1,\ldots,n\}$, we can conclude by **(5)** that $P_{I_i}(su) = P_{I_i}(sh)$. We can now apply *Proposition 1*, *point f*, and conclude: $su \in \mathcal{I}_{m_i}$

Combining with **(7)**, we can conclude:

$su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}\mathcal{I}_{m_j}]$ **(8)**

All that remains is to show $su \in \cap_{j\in\{1,\ldots,n\}}\mathcal{L}_{m_j}$

From **(3)**, we have $su_i \in \mathcal{L}_{m_i}$ for $i = \{1,\ldots,n\}$. From **(4)**, we have $P_{IHL_i}(u) = u_i = P_{IHL_i}(u_i)$ as $u_i \in (\Sigma_{IH} \cup \Sigma_{L_i})^*$. As $\Sigma_{IL_i} \subseteq (\Sigma_{IH} \cup \Sigma_{L_i})$, we can conclude $P_{IL_i}(u) = P_{IL_i}(u_i)$. This implies $P_{IL_i}(su) = P_{IL_i}(su_i)$. We can now apply *Proposition 1*, *point d*, and conclude:

$su \in \mathcal{L}_{m_i}$

Combining with **(8)**, we have: $su \in \mathcal{H}_m \cap [\cap_{j\in\{1,\ldots,n\}}(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})]$

From **(5)**, we have $P_{IH}(u) = h$, as required.

■

Bibliography

[1] B.A. Brandin and F.E. Charbonnier. The supervisory control of the automated manufacturing system of the AIP. In *Proc. Rensselaer's 1994 Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 319–324, Troy, Oct 1994.

[2] F. Charbonnier. Commande par supervision des systèmes à événements discrets: application à un site expérimental l'Atelier Inter-établissement de Productique. Technical report, Laboratoire d'Automatique de Grenoble, Grenoble, France, 1994.

[3] Pengcheng Dai. Synthesis method for hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2006. [ONLINE] http://www.cas.mcmaster.ca/~leduc/#studtheses.

[4] Hugo Flordal and Robi Malik. Modular nonblocking verification using conflict equivalence. In *Proc. of WODES 2006*, pages 100–106, Ann Arbor, USA, Jul. 2006.

[5] Richard Hill and Dawn Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Proc. of WODES 2006*, pages 399–406, Ann Arbor, USA, Jul. 2006.

[6] Ryan Leduc. *Hierarchical Interface-based Supervisory Control.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2002. [ONLINE] Available: http://www.cas.mcmaster.ca/~leduc.

[7] Ryan J. Leduc, Bertil A. Brandin, Mark Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control, part I: Serial case. *IEEE Trans. Automatic Control*, 50(9):1322–1335, Sept. 2005.

[8] Ryan J. Leduc, Mark Lawford, and Pengcheng Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Trans. on Control Systems Technology*, 14(4):654–668, July 2006.

[9] Ryan J. Leduc, Mark Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control, part II: Parallel case. *IEEE Trans. Automatic Control*, 50(9):1336–1348, Sept. 2005.

[10] C. Ma and W. Murray Wonham. *Nonblocking Supervisory Control of State Tree Structures*, volume 317 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Berlin, Germany, 2005. A1: 20010101.

[11] Ryan J. Leduc and Pengcheng Dai. Synthesis method for hierarchical interface-based supervisory control. In *Proc. of 26th American Control Conference*, pages 4260–4267, New York City, USA, July 2007.

[12] P.N. Pena, J.E.R. Cury, and S. Lafortune. Testing modularity of local supervisors: An approach based on abstractions. In *Proc. of WODES 2006*, pages 107–112, Ann Arbor, USA, Jul. 2006.

[13] P. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete-event processes. *SIAM J. Control Optim*, 25(1):206–230, 1987.

[14] W. M. Wonham. *Supervisory Control of Discrete-Event Systems*. Department of Electrical and Computer Engineering, University of Toronto, July 2006. Monograph and TCT software can be downloaded at http://www.control.toronto.edu/DES/.

[15] W. M. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim*, 25(3):637–659, May 1987.