

Hierarchical Interface-based Supervisory Control of a Flexible Manufacturing System

R.J. Leduc, M. Lawford, and P. Dai

December 22, 2005

Abstract

Flexible manufacturing systems have long been touted as an application area for supervisory control theory. Unfortunately, due to the typical exponential growth of state space with the number of interacting subsystem, concurrent systems such as manufacturing applications have, for the most part, remained beyond the reach of existing supervisory control theory tools. This paper demonstrates how, by imposing a hierarchical, modular, interface-based architecture on the system, significant gains can be made in the size of applications that can be handled by supervisory control theory. We first review Hierarchical Interface-based Supervisory Control (HISC), providing the theory necessary to motivate the creation of well defined, automata-based interfaces between components. This architecture permits the verification of global safety (controllability) and non-blocking properties to be decomposed into a set of local checks, each of which only involves an individual component subsystem and its interface automata. The paper then provides a detailed description of how the theory can be applied to the design and verification of a flexible manufacturing system work cell. The work cell model is based upon the Atelier Inter-établissement de Productique (AIP) flexible manufacturing workcell, a system that has been previously studied in the literature with limited success.

Index Terms

Discrete event systems, hierarchical systems, automata, interfaces, formal methods.

I. INTRODUCTION

In Supervisory Control 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 (a form

of liveness) and (ii) controllable (a form of safety). The main obstacle to performing these tasks is the combinatorial explosion of the product state space. Although many methods have been developed to deal with this problem (modular control [1], [2], [3], [4], [5], decentralized control [6], [7], [8], [9], [10], Vector DES (VDES) [5], [11], [12] and Petri Nets (PN) [13], [14], model aggregation methods - e.g. [15], [16], [17], [18], [19], [20], and multi-level hierarchy [21], [22], [23], [24], [25]), large-scale systems are still problematic, particularly for verification of nonblocking.

In contrast to the majority of approaches which apply mathematical techniques to produce aggregate models of an existing system, our method of restricting component interaction to well defined interfaces provides a design heuristic to guarantee scalability by construction. As we will demonstrate with an example, when the design heuristic is followed in the construction of a large-scale system, our theoretical results can be applied to verify the non-blocking and controllability of the complete system by checking local conditions. In order to achieve our ultimate goal of scalability, we restrict the permissible system architectures and sacrifice global maximal permissiveness to obtain a (generally) suboptimal solution, but one that is more tractable.

In [26], [27], [28], we developed *Hierarchical Interface-based Supervisory Control (HISC)*, a method that 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. Each subsystem is modeled as a deterministic finite state automaton that is capable of generating different sequences of events. The interfaces are also modeled as automata, with the results that the past history of communication between components influences the current and future possible communications. Each interface between a pair of components effectively establishes a protocol for interaction of the connected subsystems. As a result of this structured interaction, it is possible to derive a set of local consistency properties that can be used to verify if a discrete-event system is globally nonblocking and controllable. Each of these consistency properties can be verified using a single subsystem and its interface(s); thus the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources.

In this paper we show how HISC can be applied to the verification of a controller for a model of a relatively complex flexible manufacturing system. The model is based upon the Atelier Inter-établissement de Productique (AIP), an automated manufacturing system consisting of a

central loop and four external loops, three assembly stations, an input/output station, and four inter-loop transfer units. The estimated worst case state space size of the plant model is 2×10^{43} . As a result, controller synthesis techniques that required the construction of the entire plant, such as most of those referenced above, could not be utilized to design a verifiably correct controller. Therefore after modeling the open loop system, local controllers were designed based upon heuristics and experience and then verified to result in globally nonblocking, controllable closed loop behavior using the HISC method.

Despite the limitations of existing controller synthesis techniques, when the architecture required by HISC is imposed upon the system, supervisory control theory still provides a useful framework for the design and verification of a complex system. In this paper, our goal is to provide a detailed example that serves effectively as a tutorial on how HISC might be applied to similar system and motivate further research on interface based supervisor synthesis methods.

Section II provides an overview of the HISC results of [29], [30], [28]. It defines interfaces, describes how they are used to restrict the information flow of the system, and then provides a set of local consistency properties that can be used to verify if the system is globally nonblocking and controllable. Section III provides an overview of the AIP system and describes the desired closed loop behavior of the system under a particular set of assumptions. We describe the modular decomposition (in the sense of [31]) of the system and provide the details of the component interfaces and supervisor design in Section IV. We close with Section V discussing the results of applying the method to the AIP system and Section VI drawing general conclusions and motivating future work while discussing the methods limitations.

We note here that the interface conditions that we present in this work are a modified version of the conditions used in [29], [30], [28]. The new set of conditions are more concise and clear, particularly with respect to what checks need to be performed on a given component. In the Appendix, we show that our definitions are equivalent to the ones given in [29], [30], [28].

II. OVERVIEW OF HISC

We begin with a brief review of supervisory control theory and then provide the HISC theory needed for the AIP example.

A. Discrete-Event Systems Preliminaries

Supervisory control theory [32], [33], [5] 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 [5]. 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 $\text{Pwr}(\Sigma)$ denote the power set of Σ . For language L , the eligibility operator $\text{Elig}_L : \Sigma^* \rightarrow \text{Pwr}(\Sigma)$ is given by $\text{Elig}_L(s) := \{\sigma \in \Sigma \mid s\sigma \in L\}$ for $s \in \Sigma^*$. We will write $\Sigma - \Sigma_o$ to be the set subtraction of Σ_o from Σ .

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 $\delta : Y \times \Sigma \rightarrow Y$ 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 $\delta : Y \times \Sigma^* \rightarrow Y$ 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 $L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(y_o, s)!\}$. The marked behavior of \mathbf{G} , is defined as $L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(y_o, s) \in Y_m\}$. The reachable state subset of DES \mathbf{G} , denoted Y_r , is: $Y_r := \{y \in Y \mid (\exists s \in \Sigma^*) \delta(y_o, s) = y\}$. A DES \mathbf{G} is reachable if $Y_r = Y$. We will always assume \mathbf{G} is reachable.

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

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

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 $P_i^{-1} : \text{Pwr}(\Sigma_i^*) \rightarrow \text{Pwr}(\Sigma^*)$ is the inverse image function of P_i (see e.g. [5]).

The synchronous product of DES $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o1}, Y_{m1})$ and $\mathbf{G}_2 = (Y_2, \Sigma_2, \delta_2, y_{o2}, Y_{m2})$, denoted $\mathbf{G}_1 \parallel \mathbf{G}_2$, is defined to be a reachable DES \mathbf{G} with the properties:¹

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1) \parallel L_m(\mathbf{G}_2), \quad L(\mathbf{G}) = L(\mathbf{G}_1) \parallel L(\mathbf{G}_2), \quad \text{and event set } \Sigma = \Sigma_1 \cup \Sigma_2$$

For DES, the two main properties we want to check are *nonblocking* and *controllability*. A DES \mathbf{G} is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, i.e. any string can always be continued to a string resulting in a marked state.

To control the plant, we define a *supervisor*. As this paper focuses on verification and not synthesis, we will use the terms supervisor and specification interchangeably as we require that all specifications be controllable. A supervisor is represented as an automaton $\mathbf{S} = (X, \Sigma_S, \xi, x_o, X_m)$.

The synchronous product operator is used to specify the closed loop behavior of the system. The behaviour of a plant \mathbf{G} under the control of a supervisor \mathbf{S} is thus $\mathbf{System} := \mathbf{G} \parallel \mathbf{S}$.

We will denote the disjoint union of sets by $\dot{\cup}$ and adopt the standard partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events*. The formal definition for controllability follows.

Definition 1: Let $\Sigma := \Sigma_1 \cup \Sigma_S, P_1 : \Sigma^* \rightarrow \Sigma_1^*$ and $P_S : \Sigma^* \rightarrow \Sigma_S^*$. 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, $(\forall s \in \mathcal{L}_{\mathbf{G}_1} \cap \mathcal{L}_{\mathbf{S}}) \text{Elig}_{\mathcal{L}_{\mathbf{G}_1}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{L}_{\mathbf{S}}}(s)$.

Thus a supervisor is controllable for a plant if no uncontrollable plant actions can take the plant outside of the supervisor's specified behavior.

B. Hierarchical Interface Based Supervisory Control

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. Figure 1 shows conceptually the structure and information flow of the system in the special case when there is only a single low-level system. Communication between the high-level system and the low-level system occurs in a serial fashion. A request from the high-level is followed by an answer from the low-level before the next request is issued to the low-level subsystem. This style of interaction is enforced by an interface that mediates communication between the

¹We are overloading the \parallel operator here by using it for both languages and DES, but this should not cause confusion as the choice of arguments will make the meaning clear.

two subsystems. All system components, including the interface, are modeled as automata as shown in Fig. 2 where our *flat system* would be $\mathbf{G} := \mathbf{G}_H || \mathbf{G}_I || \mathbf{G}_L$. By flat system we mean the equivalent DES if we ignored the interface structure.

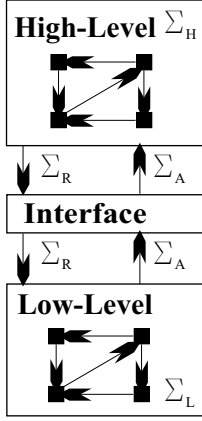


Fig. 1. Interface Block Diagram.

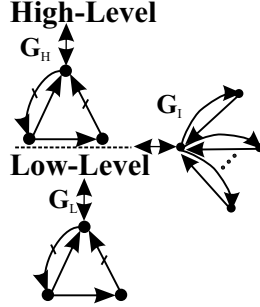


Fig. 2. Two Tiered Structure of the System.

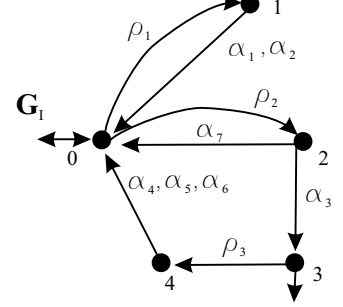


Fig. 3. Example Interface.

In order to restrict information flow and decouple the subsystems, the event set Σ is split into four disjoint alphabets: Σ_H , Σ_L , Σ_R , and Σ_A . The events in Σ_H are *high-level events* and the events in Σ_L *low-level events* as these events appear only in the high-level and low-level models, \mathbf{G}_H and \mathbf{G}_L respectively. We then have \mathbf{G}_H defined over $\Sigma_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ and \mathbf{G}_L defined over $\Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$.

As the *interface automaton* \mathbf{G}_I is only concerned with communication between the two subsystems, it is defined over the events that are common to both levels of the hierarchy, $\Sigma_R \dot{\cup} \Sigma_A$, which are collectively known as the set of *interface events*, denoted Σ_I . The events in Σ_R , called *request events*, represent commands sent from the high-level subsystem to the low-level subsystem. The events in Σ_A are *answer events* and represent the low-level subsystem's responses to the request events. In order to enforce the serialization of requests and answers, we restrict the interface to the subclass of command-pair interfaces defined below.

Definition 2: A DES $\mathbf{G}_I = (X, \Sigma_R \dot{\cup} \Sigma_A, \xi, x_o, X_m)$ is a *command-pair interface* if:

- (A) $L(\mathbf{G}_I) \subseteq \overline{(\Sigma_R \cdot \Sigma_A)^*}$, and
- (B) $L_m(\mathbf{G}_I) = (\Sigma_R \cdot \Sigma_A)^* \cap L(\mathbf{G}_I)$

Condition (A) says that request events and answer events must alternate (i.e. serialization of requests) while condition (B) states that every answered request results in a marked state. An

example command pair interface with $\Sigma_R := \{\rho_i | i = 1, 2, 3\}$ and $\Sigma_A := \{\alpha_i | i = 1, \dots, 7\}$ is shown in Fig. 3.

We now generalize the above “serial case” where there is a single low-level system, to the *parallel* case where there are n low-level subsystems. In this case we say that the system is an n^{th} degree parallel system. Figure 4 shows conceptually the structure and flow of information. The single high-level subsystem, interacts with $n \geq 1$ independent low-level subsystems, communicating with each low-level subsystem in parallel through a separate interface.

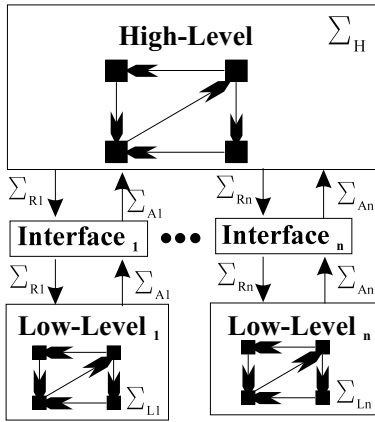


Fig. 4. Parallel Interface Block Diagram.

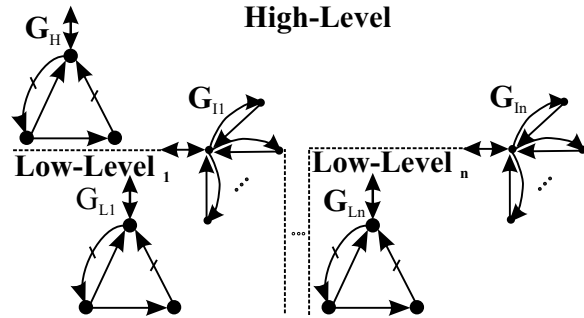


Fig. 5. Two Tiered Structure of Parallel System

As in the serial case, to restrict the flow of information at the interface, we partition the system alphabet into pairwise disjoint alphabets:

$$\Sigma := \Sigma_H \dot{\cup} \bigcup_{j=1, \dots, n} [\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j}] \quad (1)$$

The high-level subsystem is modeled by DES G_H (defined over event set $\Sigma_H \dot{\cup} (\dot{\cup}_{j \in \{1, \dots, n\}} [\Sigma_{R_j} \dot{\cup} \Sigma_{A_j}])$). For $j \in \{1, \dots, n\}$, the j^{th} low-level subsystem is modeled by DES G_{L_j} (defined over event set $\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j}$), and the j^{th} interface by DES G_{I_j} (defined over event set $\Sigma_{R_j} \dot{\cup} \Sigma_{A_j}$). The overall system has the structure shown in Fig. 5. Thus our flat system is $G = G_H || G_{I_1} || G_{L_1} || \dots || G_{I_n} || G_{L_n}$.

To simplify notation in our exposition, we bring in the following event sets, natural projections, and languages. For the remainder of this section, the index j has range $\{1, \dots, n\}$.

$$\begin{aligned} \Sigma_{I_j} &:= \Sigma_{R_j} \cup \Sigma_{A_j}, & P_{I_j} &: \Sigma^* \rightarrow \Sigma_{I_j}^* \\ \Sigma_{IL_j} &:= \Sigma_{L_j} \cup \Sigma_{I_j}, & P_{IL_j} &: \Sigma^* \rightarrow \Sigma_{IL_j}^* \end{aligned}$$

$$\begin{aligned}
\Sigma_{IH} &:= \Sigma_H \cup \bigcup_{k \in \{1, \dots, n\}} \Sigma_{I_k} & P_{IH} &: \Sigma^* \rightarrow \Sigma_{IH}^* \\
\mathcal{H} &:= P_{IH}^{-1}(L(\mathbf{G}_H)), & \mathcal{H}_m &:= P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\
\mathcal{L}_j &:= P_{IL_j}^{-1}(L(\mathbf{G}_{L_j})), & \mathcal{L}_{m_j} &:= P_{IL_j}^{-1}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^* \\
\mathcal{I}_j &:= P_{I_j}^{-1}(L(\mathbf{G}_{I_j})), & \mathcal{I}_{m_j} &:= P_{I_j}^{-1}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^*
\end{aligned}$$

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 ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is *interface consistent* with respect to the alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$, 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 command-pair interface.

High-Level Property

- 3)

$$(\forall s \in \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k) \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k}(s)$$

Low-Level Properties

- 4) $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j) \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}_j}(s)$
- 5) $(\forall s \in \Sigma^* \cdot \Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$

$$\text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s \Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \quad \text{where}$$

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

- 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 first two properties assert that the system has the required basic architecture, with the high-level and low-level subsystems only sharing request and answer events and the interaction between the levels mediated by interfaces. This provides a form of information hiding as it

restricts the high-level subsystem from knowing (and directly affecting) internal details of the low-level subsystems and vice versa.

The high-level property (3) asserts that when \mathbf{G}_H is synchronized with all of the other subsystem interfaces $\mathbf{G}_{I_k}, k \neq j$, it must always accept an answer event if the event is eligible in the interface \mathbf{G}_{I_j} . In other words, the high-level subsystem is forbidden to assume more about when an answer event can occur than what is provided by the interface. Similarly, low-level property (4) asserts that the low-level subsystem (\mathbf{G}_{L_j}) must always accept a request event if the event is eligible in its interface \mathbf{G}_{I_j} . We note that both (3) and (4) can be computed using the standard algorithms for controllability.

Condition (5) states that immediately after a request event (some $\rho \in \Sigma_{R_j}$) has occurred, and before it is followed by any low-level events in Σ_{L_j} , there exist one or more paths via strings in $\Sigma_{L_j}^*$ to each answer event that \mathbf{G}_{I_j} says can follow the request event. Finally, (6) asserts that every string marked by the interface \mathbf{G}_{I_j} and accepted by the low-level subsystem, can be extended by a low-level string to a string marked by \mathbf{G}_{L_j} .

C. 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 system \mathbf{G} is to be nonblocking.

Definition 4: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is said to be *level-wise nonblocking* if the following conditions are satisfied:

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

$$\overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}} = \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k$$

(II) *nonblocking at the low-level:* for all $j \in \{1, \dots, n\}$,

$$\overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$$

The above definition can be paraphrased as saying that for each component subsystem synchronized with its interface(s), every reachable state must have a path to a state that is marked by both the subsystem and its interface(s). We are now ready to present our nonblocking theorem for parallel interface systems.

Theorem 1: If the n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is level-wise nonblocking and interface consistent with respect to the alphabet partition given by (1), then $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, where $\mathbf{G} = \mathbf{G}_H || \mathbf{G}_{I_1} || \mathbf{G}_{L_1} || \dots || \mathbf{G}_{I_n} || \mathbf{G}_{L_n}$.

Proof:

Results follow immediately from *Theorem 3* from [28], and *Theorems 3*, and *4* of the Appendix. ■

D. Local Conditions for Global Controllability of the System

The representation of the system given in Fig. 5 simplifies notation when verifying nonblocking by ignoring the distinction between plants and supervisors. 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:

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

For the controllability requirements at each level, we adopt the standard partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events*. Note that this partition may, in general, be independent of the partition (1).

Definition 5: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \dots, \mathbf{G}_{L_n}^p, \mathbf{S}_H, \mathbf{S}_{L_1}, \dots, \mathbf{S}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is *level-wise controllable* with respect to the alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$ 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) $(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j) \text{ Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$
- (III) $(\forall s \in \mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k] \cap \mathcal{S}_H) \text{ Elig}_{\mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k]}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_H}(s)$

The above definition states that the system is level-wise controllable if, for the given distributed supervisor, the high-level supervisor is controllable for the high-level plant combined with all of the interfaces (by III) and that each low-level supervisor synchronized with the subsystem's interface is controllable for the subsystem's low-level plant (by II). Point (I) is an information hiding statement. By restricting the supervisors to the indicated event sets, we allow them to only view and disable events for their specific component.

In the above definition, we treated interfaces as supervisors in point (II), and plants in point (III). This discrepancy is a result of our hierarchy. As the high-level is concerned with global behavior (behavior that affects more than one low-level), it sees request and answer events as abstract actions performed by the low-level, but is not concerned with the details of how these events are implemented. As a low-level's job is to implement the commands (request events) given it by the high-level, it thus has sufficient details to verify that its interface, working in conjunction with the low-level supervisor, is indeed controllable. As we have verified that the interfaces are controllable in point (II), we do not need to verify them again in point (III). By treating the interfaces as plants at the high-level, we can allow the high-level supervisor to be less restrictive, as the high-level plant alone does not typically have enough information about when interface events are eligible to occur as the interfaces themselves. Removing the interfaces from point (III) would also be too restrictive as this could not only make the high-level supervisor uncontrollable for the high-level plant, but the the supremal controllable sublanguage (for the high-level) would likely be the empty set!

We now present a sufficient condition for controllability of parallel interface systems. At first glance, the controllability definition used below might seem slightly different than the one given in Section 1, but this can be easily reconciled by noting that for *Theorem 2*, $\Sigma_1 = \Sigma_S = \Sigma$.

Theorem 2: If the n^{th} degree ($n \geq 1$) parallel interface system composed of plant components $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \dots, \mathbf{G}_{L_n}^p$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \dots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is level-wise controllable with respect to the alphabet partition given by (1), then

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

Proof:

Results follow immediately from *Theorem 4* from [28], and *Theorem 5* of the Appendix. ■

E. Verifying Properties

To aid in investigating HISC, we have developed software routines to verify that a system satisfies the level-wise nonblocking, interface consistent, and level-wise controllable conditions. These routines were developed by Leduc during his collaboration with Siemens Corporate Research and are a part of an experimental software tool. Currently, an open source implementation of these routines is being developed, but has not yet been released.

Examining the conditions to be verified, one sees that most of them are either very straightforward (i.e. verifying two sets are disjoint), or can be verified using existing supervisory control algorithms after suitable definitions have been made. **Points 3 and 4** of the interface consistency definition can be verified using standard controllability algorithms such as TCT's **condat** function [5]. For example, in the case of **Point 4**, we simply define **ThePlant** = \mathbf{G}_{L_j} , **TheSpec** = \mathbf{G}_{L_j} , $\Sigma_u = \Sigma_{R_j}$, and $\Sigma_c = \Sigma - \Sigma_{R_j}$.

The exceptions are **Points 5 and 6** of the interface consistency definition. They both require new algorithms, which we presented in [30]. Further details of the algorithms, including discussion of counter example generation when the conditions fail, can be found in [30] and [28].

An important topic of current research is how to synthesize local controls that cause an interface inconsistent system to become consistent or produce a controllable nonblocking sublanguage of a specification when these conditions fail. This is currently the topic of the M.A.Sc. thesis [34].

III. 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) as described in [35] and [36]. The AIP, shown in Figure 6, 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, chosen at random.

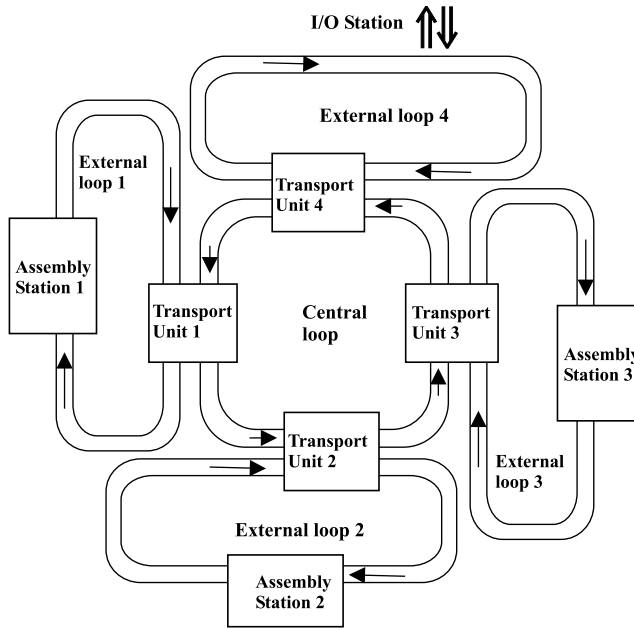


Fig. 6. The Atelier Inter-établissement de Productique

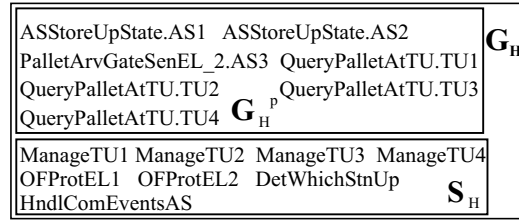


Fig. 7. AIP High-Level

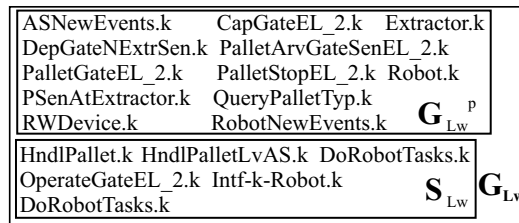


Fig. 8. AIP Low-Level $k = AS1, AS2$

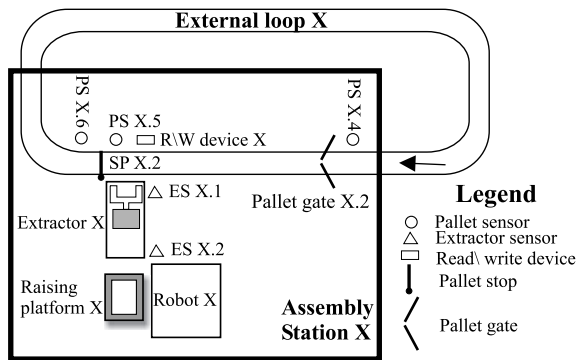


Fig. 9. Assembly Station of External Loop $X = 1, 2, 3$.

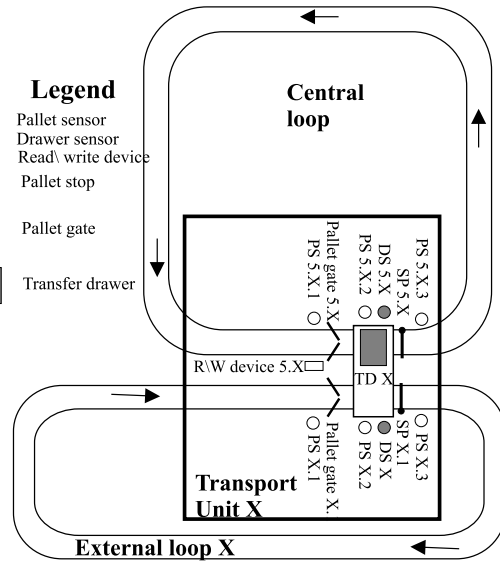


Fig. 10. Transport Unit for External Loop $X = 1, 2, 3, 4$

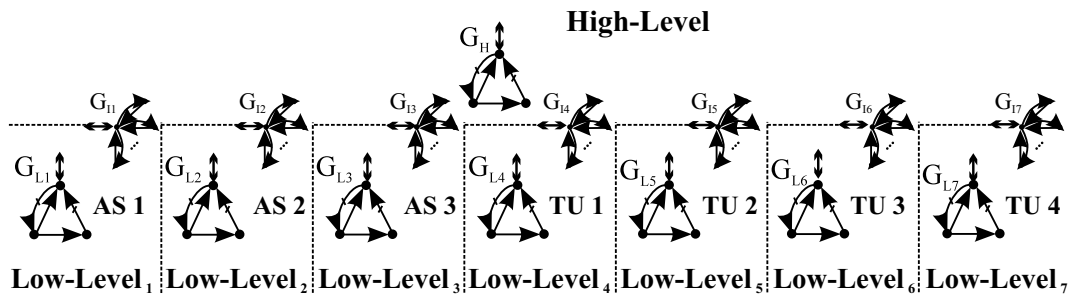


Fig. 11. Structure of Parallel System

A. Assembly Stations

The assembly stations are shown in Figure 9. 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.

B. Transport Units

The structure of the four identical transport units is shown in Figure 10. 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.

C. Control Specifications

For this example, we adopt the control specifications and assumptions used in [35] and [36] and restated as points (1)-(6) below. To this we add *Specification 7* to make the assembly stations more interesting and complicated.

Assumptions: We assume that (i) the system is initially empty, (ii) two types of pallets are randomly introduced to the system, subjected to assembly operations, and then leave, and (iii) pallets enter the system following the order: type 1, type 2, type 1, ...

Specifications:

- 1) **Routing:** Pallets follow a certain route based on their type. A type 1 pallet must go first to AS1, then AS2 before leaving the system. Type 2 pallets go first to AS2, then AS1 before leaving the system. A pallet is not allowed to leave the system until all four assembly tasks have been successfully performed on it.
- 2) **Maximum capacity of external loops 1 and 2:** The maximum allowed number of pallets in each loop at a given time is one.
- 3) **Ordering of pallet exit from system:** The pallets must exit the system in the following order: type 1, type 2, type 1, ...
- 4) **Assembly errors:** When a robot makes an assembly error, the pallet is marked damaged and routed to AS3 for maintenance. After maintenance, the pallet is returned to the original assembly station to undergo the assembly operation again.
- 5) **Assembly station breakdown:** The robots of external loops 1 and 2 are susceptible to breakdowns. When a station is down, pallets are routed to assembly station 3 which is capable of performing all tasks of the other two stations. When the failed station is repaired, all pallets not already in external loop 3 are rerouted to the original station.
- 6) **Maximum capacity of assembly stations:** To avoid collisions, only one pallet is allowed in a given station at a time.
- 7) **Assembly task ordering:** Assembly tasks are performed in a different order for pallets of different types. For pallets of type 1, task1A is performed before task1B, and task2A is performed before task2B. For pallets of type 2, task1B is performed before task1A, and task2B is performed before task2A.

IV. SYSTEM STRUCTURE

To cast the AIP into a parallel interface system, we break the system down into a high-level (models how the low-levels interact with each other), and seven low-levels corresponding to the three assembly stations and four transport units, as shown in Figure 11. We describe each of the subsystems (components) in the following sections. As this example contains 181 DES, we are

not able to describe the design in complete detail, but refer the reader to [28] for a complete description.

The models and supervisors developed for this example are based on the automata presented in [35] and [36]. We have altered them to fit our setting, and extended them to fill in the missing details of several events that were defined, in order to simplify the model and reduce complexity, as “macro events.” By adding in these missing details, we substantially increased the complexity of the example which is immediately apparent by noting that the estimated worst case state space size of the original plant in [35] is 2×10^{24} , while our estimated worst case state space size is 2×10^{43} (see Section V for details).

In this paper, all supervisors were designed for their level as modular supervisors. The supervisors were designed by hand to meet the given specifications, and then verified that they satisfy their share of the interface, controllability, and non-blocking properties. If a component fails to satisfy its share of these properties, it is modified until it does satisfy them.

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 marker states are filled.

A. The High-Level

The high-level subsystem, keeps track of the breakdown status of assembly stations 1 and 2, and enforces the maximum capacity of external loops 1 and 2. This component controls the external operation of all transport units and assembly stations, 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 given assembly station is down. The majority of the high-level’s behaviour is encapsulated in its supervisors. The high-level \mathbf{G}_H consists of the synchronous product of the 15 DES listed in Figure 7. It is further subdivided into a plant component \mathbf{G}_H^p , and a supervisor component \mathbf{S}_H as indicated in the diagram.

As an example of the high-level subsystem’s behavior, we discuss supervisor *ManageTU1*, shown in Figure 12. This supervisor controls the transfer of pallets between the central loop and external loop 1. It permits pallets on the central loop to pass through transport unit 1 (to be liberated) without being transferred to the external loop. Pallets are liberated if EL1 is at maximum capacity, AS1 is down, or TU1 determines that the pallet is not to be transferred.

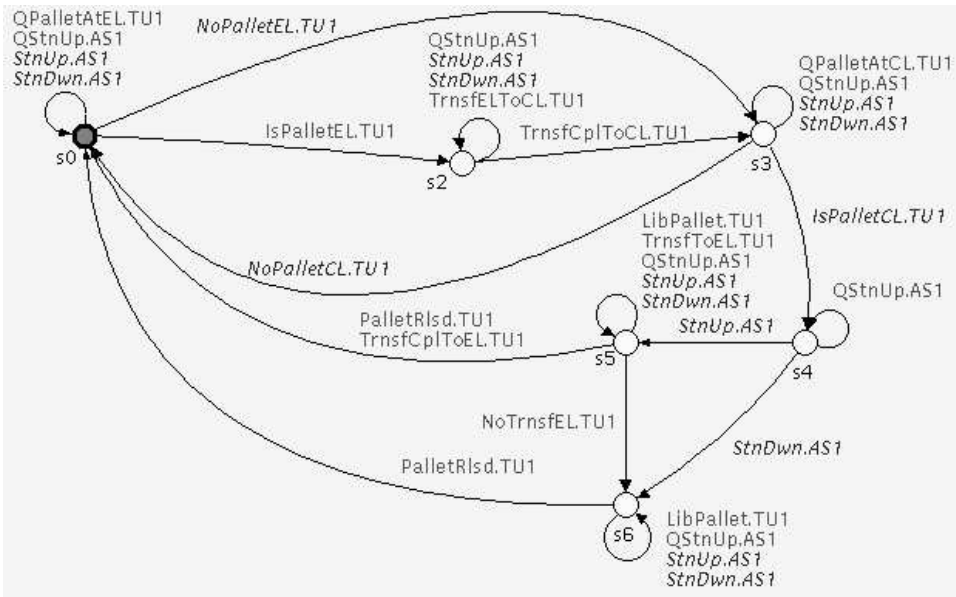


Fig. 12. Supervisor ManageTU1.

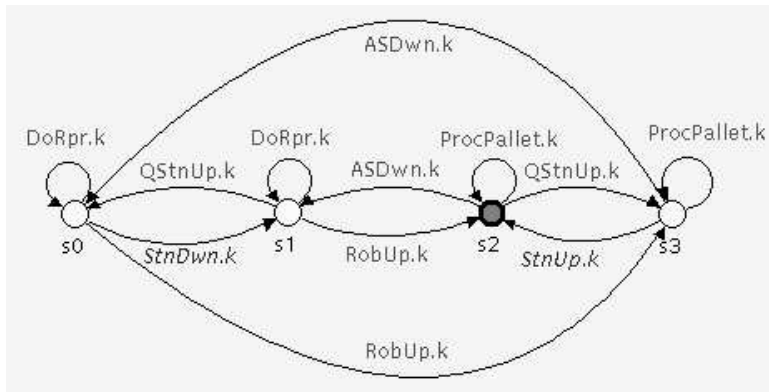


Fig. 13. ASStoreUpState.k

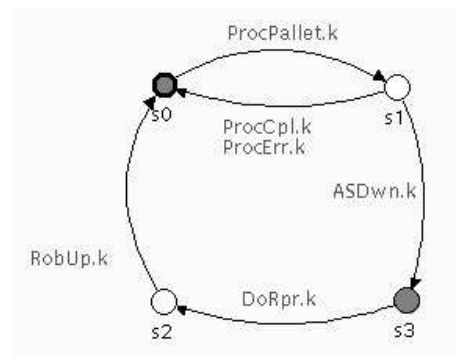


Fig. 14. Interface to low-level $k = AS1, AS2$.

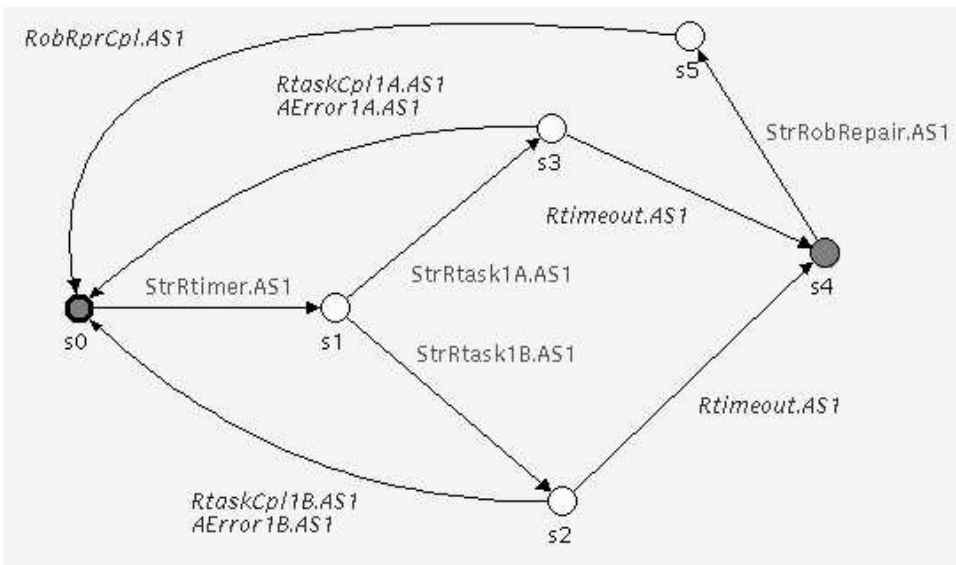


Fig. 15. Robot.AS1

The high level is able to determine if AS1 or AS2 is operational by querying subplants $ASStoreUpState.k$ ($k = AS1, AS2$), shown in Figure 13. These DES track the breakdown status of their respective assembly station, and provide a method for other DES to query the stations' status.

B. Low-Levels AS1 and AS2

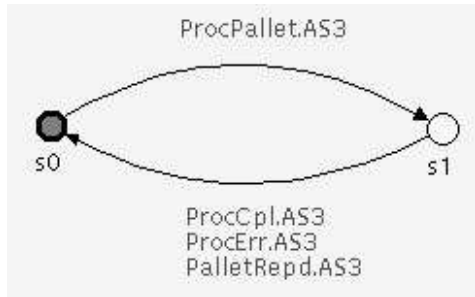
We now describe the low-level subsystems that represent assembly stations 1 and 2. As they are identical, we will describe them collectively as *component* k , where $k = AS1, AS2$. Component k provides the functionality specified in its interface, shown in Figure 14. The assembly station accepts the pallet at its gate, and presents it 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 through the interface, and the pallet is released. Component k then waits for a repair command to return the robot to operation. Figure 14 shows how the breakdown status of component k is reflected in its interface, and how only the appropriate request event is possible at a given marked state. Component k (low-level $w = 1, 2$) contains the 17 DES listed in Figure 8. The diagram gives the definition of component k 's subsystem G_{L_w} , plant component $G_{L_w}^p$, and supervisor component S_{L_w} . Again they are the synchronous product of the indicated automata.

As an example, we consider the robot that performs the assembly operations for AS1. The robot can perform two assembly tasks each, can successfully complete the assembly, generate an assembly error, or break down. Breakdown is detected by the robot failing to complete a task before its timer expires. This behavior is encapsulated in subplant $Robot.AS1$, shown in Figure 15.

$Robot.AS1$ is augmented by supervisor $DoRobotTasks.AS1$, shown in Figure 18. $DoRobotTasks.AS1$ 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, reports on the success of the assembly operation, and handles repairs when the robot breaks down.

C. Low-Level AS3

Component AS3 provides the functionality specified in its interface, shown in Figure 16. 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,



ASNewEvents.AS3	CapGateEL_2.AS3	G_{L3}^p	
DepGateNExtrSen.AS3	Extractor.AS3		
PalletGateEL_2.AS3	PalletStopEL_2.AS3		
PSenAtExtractor.AS3	RWDevice.AS3		PalletMaint
RepPalletNewEvents	DetOpNProcNewEvents		
QueryTypNCpl.AS3	ChkErr.AS3	Robot.AS3	
QueryErrNewEvents.AS3	RobotNewEvents.AS3		
HndlPalletLvAS.AS3	Intf-AS3-RepairPallet	S_{L3}	
OperateGateEL_2.AS3	HndlPallet.AS3		
Intf-AS3-DetOpNProc	DoMaintenance		
DetNProc	Intf-RepairPallet-QueryErrors.AS3		
Intf-DetOpNProc-Robot.AS3	DoChkErr.AS3		
DoRobotTasks.AS3			

Fig. 16. Interface to low level AS3.

Fig. 17. AIP Low Level AS3

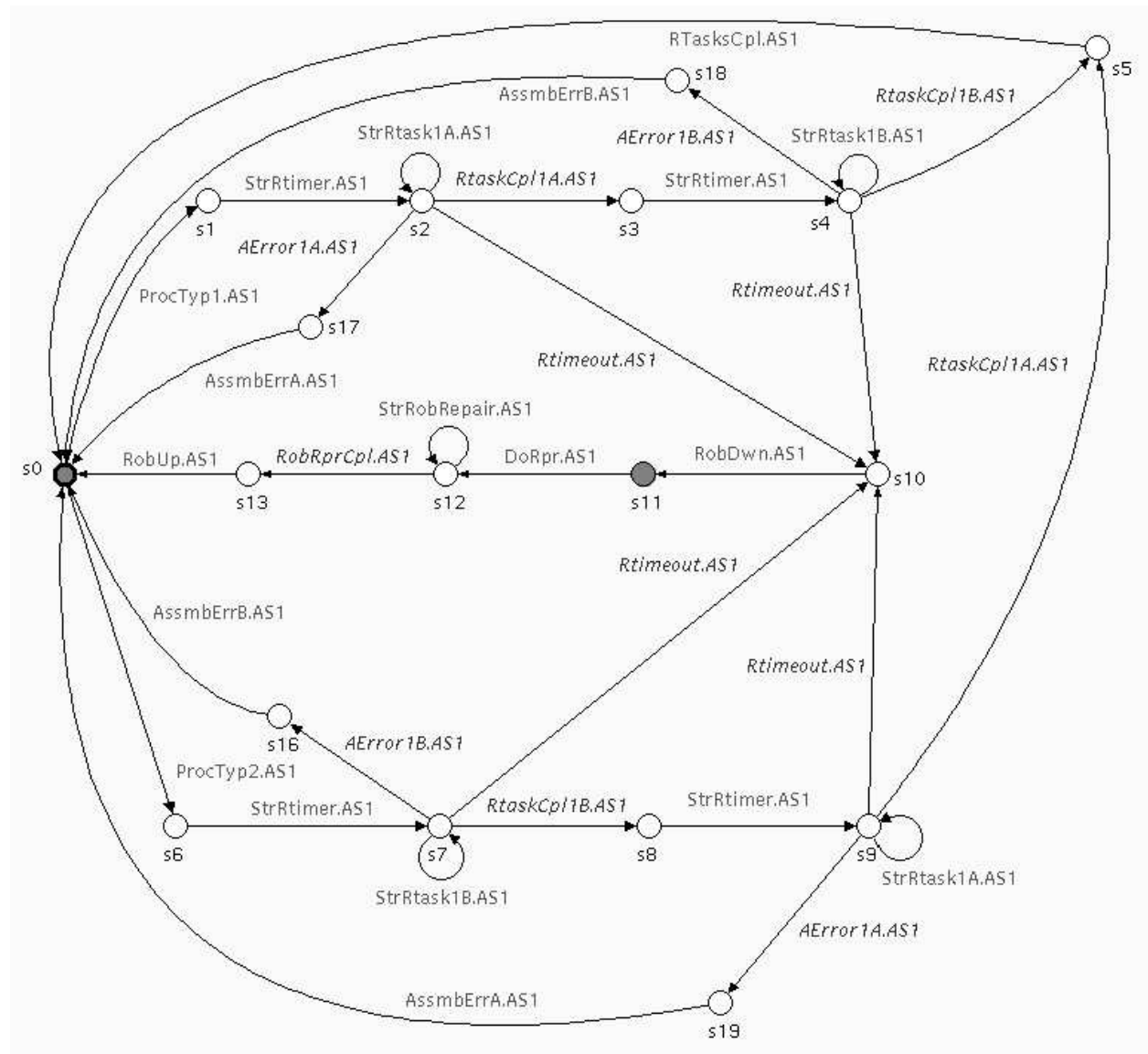


Fig. 18. Supervisor DoRobotTasks.AS1

and can substitute for either AS1 or AS2 when they are down. Component AS3 (low-level 3) contains the 27 DES listed in Figure 17. The diagram gives the definition of Component AS3's subsystem G_{L_3} , plant component $G_{L_3}^p$, and supervisor component S_{L_3} .

Supervisor *HndlPallet.AS3*, shown in Figure 19, provides an example of AS3's behavior. The supervisor handles the task of processing a pallet once it reaches the extractor. It first reads the pallet's label, and then performs any needed repairs. If repairs were required, *HndlPallet.AS3* updates the pallet's label, releases the pallet, and reports the results by "signalling through the station's interface" (allowing only the answer event with the correct meaning to occur). If no repairs were required, the pallet is presented to the robot, and the appropriate tasks are performed on the pallet. *HndlPallet.AS3* then allows the pallet to leave the assembly station and reports on the success of the processing operation by updating the pallet's label, and signalling through the station's interface.

One of the difficulties in designing an HISC system is to ensure a clear line of dependency (when required) between events in the high level, interfaces, and low levels. AS3 provides an excellent example. Taking $X = 3$ in Figure 9, we see that a pallet must reach *pallet sensor PS 3.4* before it can leave *pallet gate 3.2*. In the modelling used in [35] and [36], this was captured in a single DES. However, the "pallet has arrived at the gate" information is needed at the high level to determine when to activate the assembly station, while the "pallet leaving the gate" information is needed internally to express the relationship that a pallet cannot arrive at *pallet stop SP 3.2* (located at the extractor) until it has left the gate. The latter relationship is captured by subplant *DepGateNExtraSen.AS3*, shown in Figure 20 ($j = AS3$). The dependency between these two actions can be established by making request event *Proc.Pallet.AS3* dependent on the pallet arriving at the gate (subplant *PalletArvGateSenEL.2.AS3* in Figure 21) and the pallet leaving the gate dependent on event *Proc.Pallet.AS3* (subplant *CapGateEL.2.AS3* in Figure 22). This preserves the dependency between the pallet arriving at the gate and the pallet leaving the gate, while at the same time ensuring that the internal events of AS3 do not have direct dependencies on external events.

D. Low-Levels TU1 and TU2

We now describe the low-level subsystems that represent transport units 1 and 2. As they are identical, we will describe them collectively as *component r*, where $r = TU1, TU2$. We also

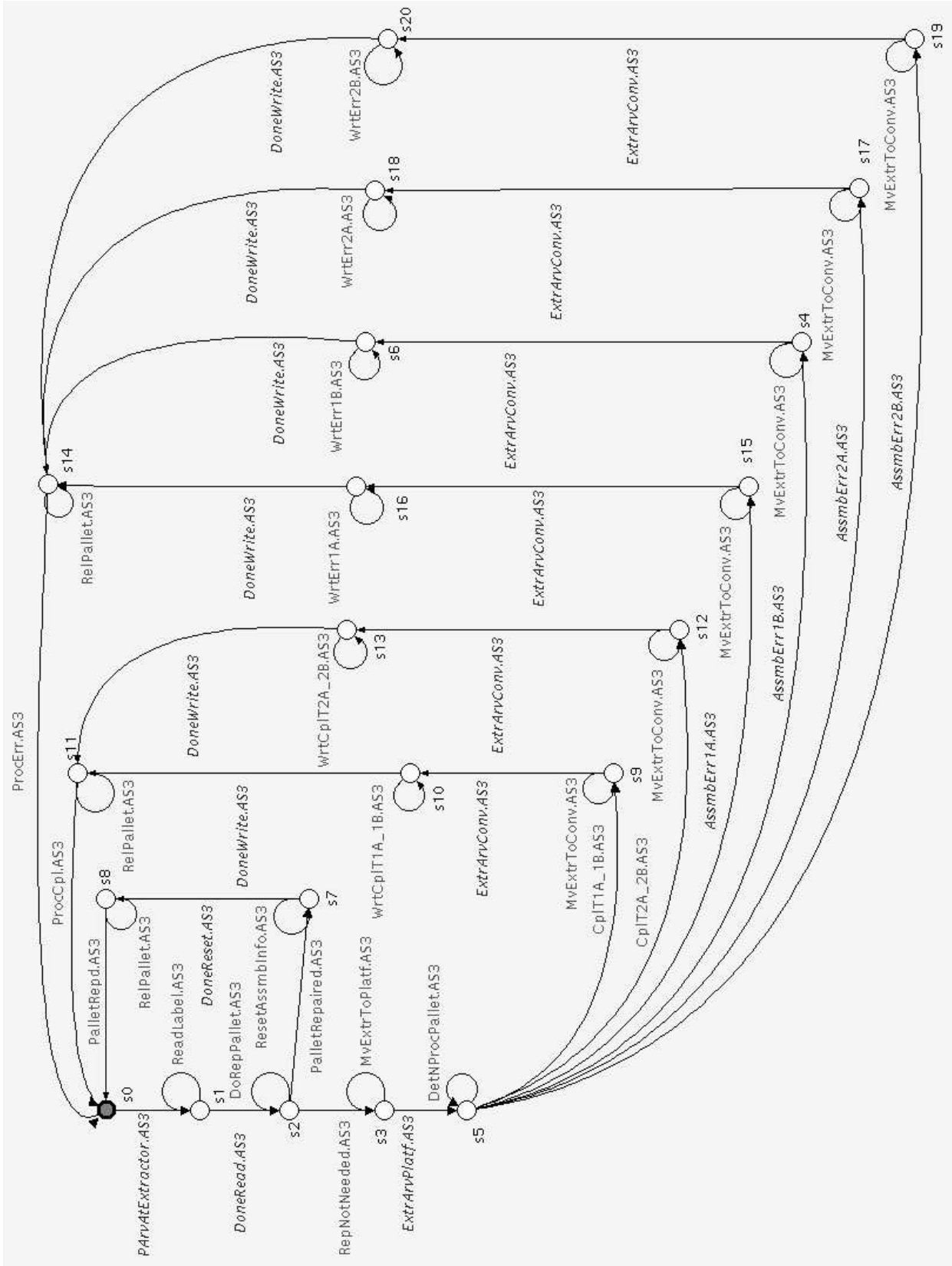


Fig. 19. Supervisor HndIPallet.AS3

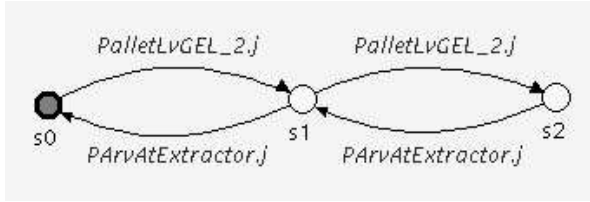


Fig. 20. DepGateNExtrSen.j

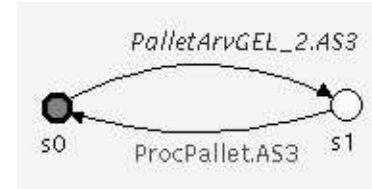


Fig. 21. PalletArvGateSenEL_2.AS3

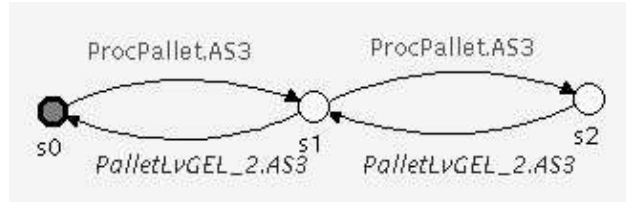


Fig. 22. CapGateEL_2.AS3

CapGateCL.r	CapGateEL_1.r	CapTUDrwToExit.r	
CapTUDrwToGateCL.r	CapTUDrwToGateEL_1.r		
PalletGateCL.r	PalletGateEL_1.r	PalletStopCL.r	
PalletStopEL_1.r	QueryDrwLoc.r	RWDevice.r	
TUDrawer.r	TUNewEvents.r	ChkErr.r	
QueryErrNewEvents.r	CheckOpNeededNewEvts.r		G_{Lv}^p
QueryTypNcpl.r			
HndlComEvents.r	HndlLibPallet.r		G_{Lv}
HndlTrnsfELToCL.r	HndlTrnsfToEL.r	S_{Lv}	
Intf-r-QueryErrors.r	Intf-r-CheckOpNeeded.r		
DoChkErr.r	DetIfOpNeeded.r		

Fig. 23. AIP Low-Level $r = TU1, TU2$

define the companion index $X = 1, 2$, which takes its values relative to r (e.g.. $X = 1$ when $r = TU1$). For diagrams in this section, we set index $i = r$. Component r (low-level $v = 4, 5$) contains the 25 DES listed in Figure 23. The diagram gives the definition of component r 's subsystem G_{Lv} , plant component G_{Lv}^p , and supervisor component S_{Lv} .

Component r provides the functionality specified in its interface, shown in Figure 24. 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). Component r has two entry points for pallets, the central loop gate, and the external loop gate. If a pallet is at the EL gate, subsystem r transfers the pallet to the central loop. If a pallet is at the CL gate, component r 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, component r 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.

As an example, we discuss supervisor $HndlTrnsfELToCL.r$, shown in Figure 26. It handles transporting pallets from EL X to the central loop. As requests to transfer pallets from an external loop to the central loop are always honored, $HndlTrnsfELToCL.r$ only needs to issue the required commands to operate the appropriate transfer drawer and pallet stop.

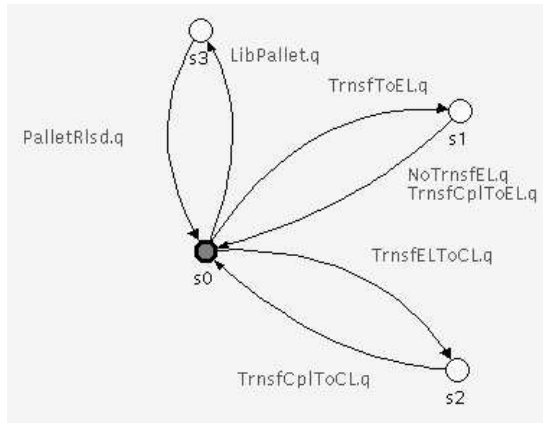


Fig. 24. Interface to low-level $q = TU1, TU2, TU4$.

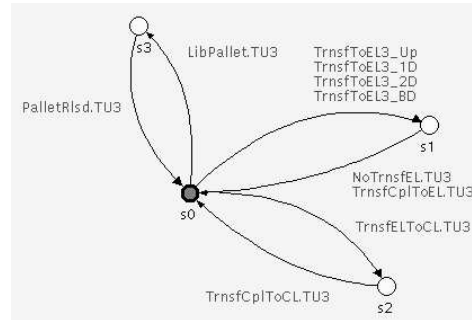


Fig. 25. Interface to *low-level* TU3.

E. Low-Level TU3

low-level TU3 provides the functionality specified in its interface, shown in Figure 25. 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 6) contains the 29 DES listed in Figure 27. The diagram gives the definition of Component TU3's subsystem G_{L_6} , plant component $G_{L_6}^p$, and supervisor component S_{L_6} .

TU3 differs from TU1 and TU2 primarily in its logic to transfer pallets from the central loop to its external loop. This is handled by supervisor *HndlTrnsfToEL.TU3*, shown in Figure 30. *HndlTrnsfToEL.TU3* will only transfer pallets to EL3 if they are damaged, or if the next assembly operation required by the pallet is performed by an assembly station that is down.

To determine if a substitute assembly operation is required, *HndlTrnsfToEL.TU3* makes use of supervisor *HndlSelCheck.TU3*, shown in Figure 29. *HndlSelCheck.TU3* maps the request events *TrnsfToEL3_Up*, *TrnsfToEL3_ID*, *TrnsfToEL3_2D*, and *TrnsfToEL3_BD*, to the appropriate local command to check to see if substitution is required. These request events were encoded by the high-level with the breakdown status of assembly stations 1 and 2, by only allowing the event with the correct meaning to occur. This is analogous to passing a parameter to a function in a

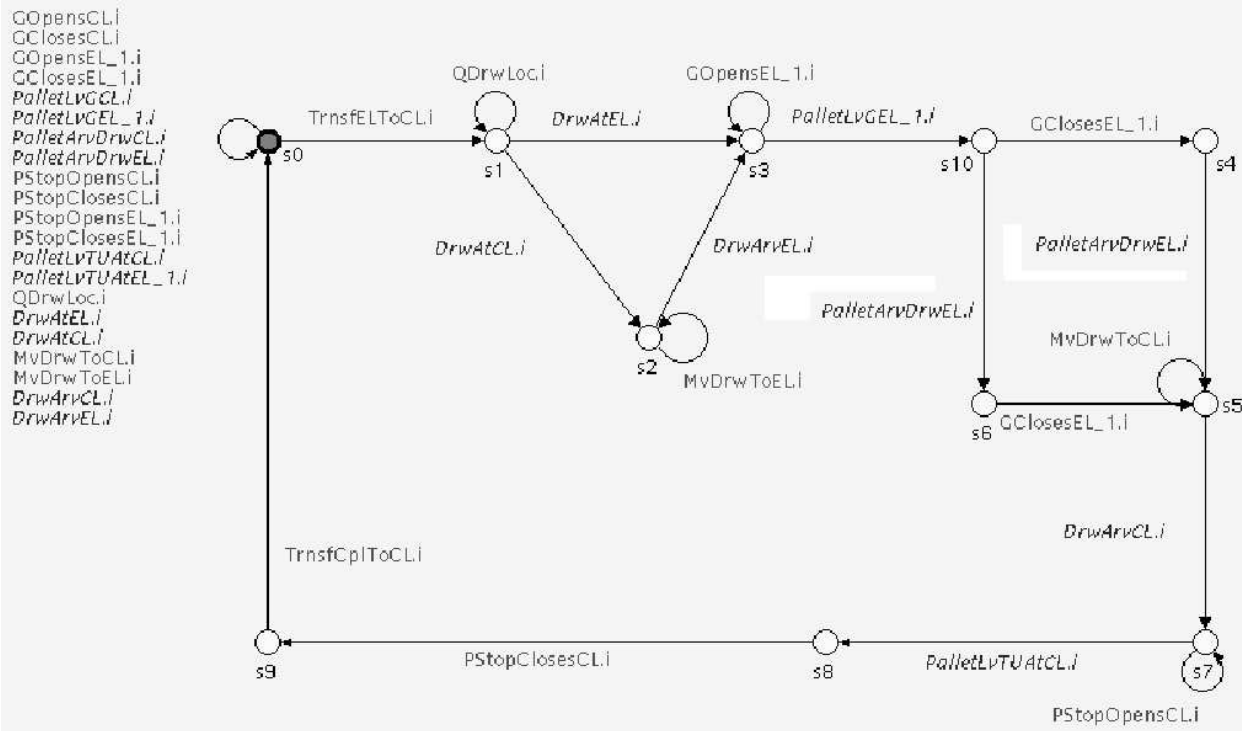


Fig. 26. HndlTrnsfELToCL.i

software program.

F. Low-Level TU4

low-level TU4 provides the functionality specified in its interface, shown in Figure 24, and contains the 19 DES listed in Figure 28. The diagram gives the definition of component TU4's subsystem G_{L_7} , plant component $G_{L_7}^p$, and supervisor component S_{L_7} .

Component TU4 (low-level 7) 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 (ie. 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 assembly tasks have been successfully performed on the pallet. This is handled by supervisor *HndlTrnsfToEL.TU4*, shown in Figure 31.

Component TU4 (low level 7) contains the 19 DES listed in Figure 28. The diagram gives the definition of component TU4's subsystem G_{L_7} , plant component $G_{L_7}^p$, and supervisor component

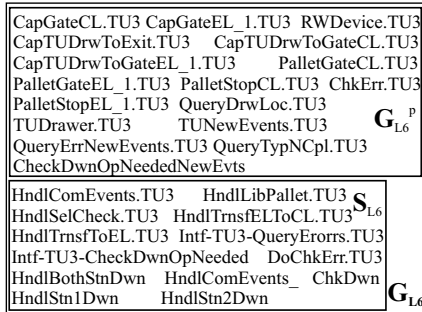


Fig. 27. AIP Low-Level TU3

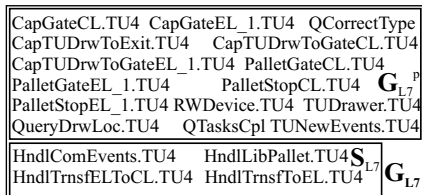


Fig. 28. AIP Low-Level TU4

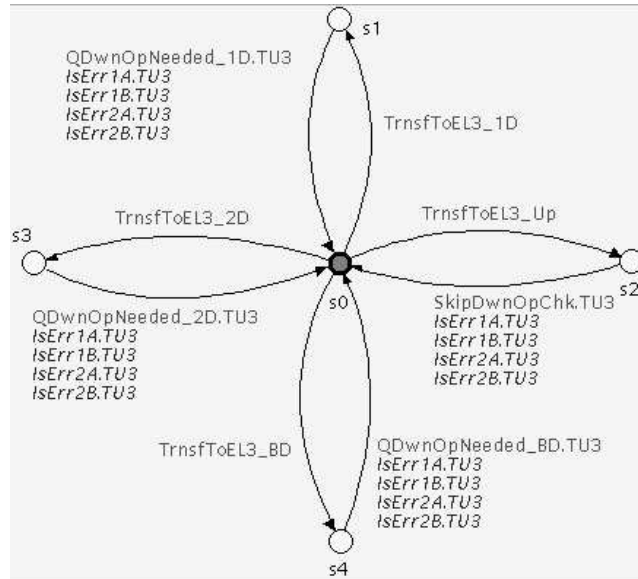


Fig. 29. HndlSelCheck.TU3

 S_{L7} .

V. DISCUSSION OF RESULTS

Applying our research tool to the system, we find that it is level-wise non-blocking, level-wise controllable, and interface consistent. By *Theorems 1* and *2*, we can conclude that the flat system is nonblocking and the flat system's supervisor is controllable for the flat plant.

This example contains 181 DES in total, with an estimated closed-loop state space of 2.9×10^{21} . This estimate was calculated by determining the closed-loop state space of the high-level, and each low-level and then multiplying these together to create a worst case state estimate. Similarly, we estimated the state space size of the open loop plant model to be 2×10^{43} . For both estimates, it's quite likely that the actual system will be considerably smaller. The computation ran for 25 minutes, using 760MB of memory. The machine used was a 750MHz Athlon system, with 512MB of RAM, 2GB of swap, running Redhat Linux 6.2. A standard nonblocking verification was also attempted on the monolithic system, but it quickly failed due to lack of memory.

Table I shows the sizes of the various subsystem automata used in the AIP calculations. First, the size of the state space of each component without being synchronized with their respective interfaces (Standalone) is given and then state space size when synchronized with their interface DES (G_H is synchronized with all seven interfaces). The last two columns give the size of the

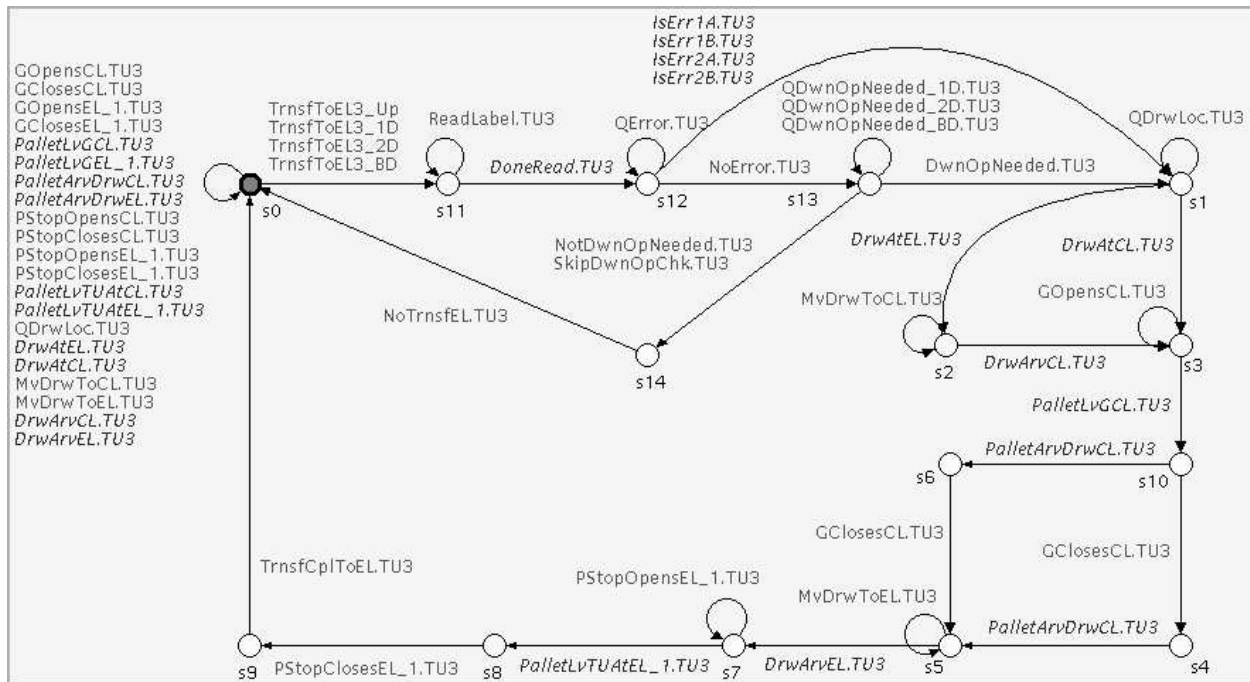


Fig. 30. Supervisor HndlTrnsfToEL.TU3

interfaces for the high-level and each low-level. Letting N_H denote the size of the state space of \mathbf{G}_H , while N_I and N_L are upper bounds for the state space size of \mathbf{G}_{I_j} and \mathbf{G}_{L_j} ($j = 1, \dots, n$), respectively, we find that the limiting factor for a monolithic algorithm² would be $N_H N_L^n$ and similarly $N_H N_I^n$ for the HISC method [30]. If we substitute actual data from Table I, we get $N_L^n = (120)^2(203)(98)^2(204)(152) = 8.71 \times 10^{14}$ and $N_I^n = (4)^2(2)(4)^4 = 8,192$. This is a potential savings of 11 orders of magnitude! In fact, instead of multiplying $N_H = 1,480,864$ by a factor of 8,192, adding the interfaces only doubles the state space of the high-level. For low-level AS1, synchronizing with its interface actually causes the state space to decrease from 1,795 to 120 states, an order of magnitude reduction.

We note that the prototype tool used for these calculations did not make use of IDD/BDD [37] and symbolic techniques such as those used in [38], [39]. We conjecture that using HISC methods with tools utilizing symbolic techniques should allow the method to scale up to considerably larger systems as has been the case with the application of symbolic techniques to monolithic supervisory control calculations.

²A monolithic algorithm is performed on the composite system, which is based on the cartesian product of subsystems.

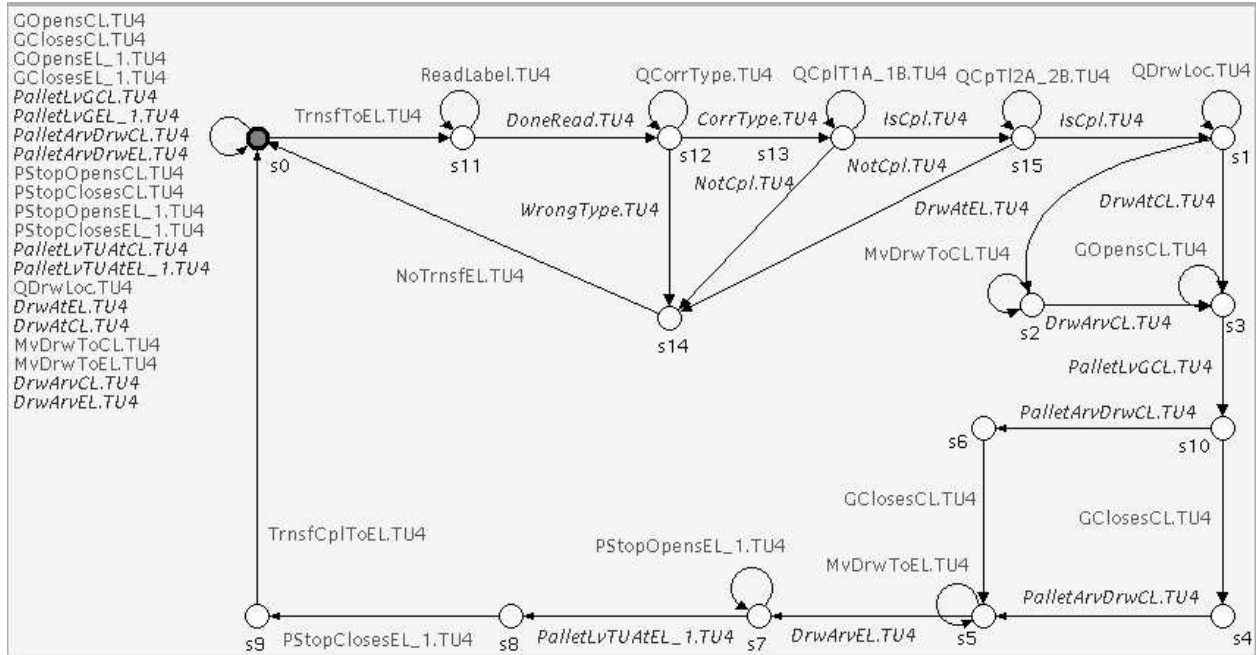


Fig. 31. HndTrnsfToEL.TU4

A. Localization of Changes

When modelling and designing supervisors for large systems like the AIP, one will be working with a large number of automata. As a result, an important concern is the effort involved in making changes to your existing design. If the plant is modified or the control specifications are changed, it is desirable to restrict the effect of these changes to as few DES as possible.

In a flat system where supervisors can see and disable any event, a small change to one part of the system could require changes to a large number of DES throughout the system. There is nothing to guarantee localization of the changes. We would also have to reverify the entire system after the changes.

For HISC, the information hiding approach creates a compartmentalization effect. As long as the interface for a low-level is unchanged, modifications to that low-level will only affect the plant and supervisor DES of that low-level. All other low-levels will be unchanged and we will only have to reverify the HISC conditions for the modified low-level. If the interface for a low-level is changed as well, this will at most cause changes to the high-level. All other low-levels will be unaffected. Finally, if the high-level is modified, no interfaces or low-levels would need

to be changed; we would only need to reverify the high-level's HISC conditions.

The HISC conditions and structure can also be used to create reusable libraries. Once an interface is defined, the low-level can be modelled, supervisors designed, and HISC conditions verified completely separately from the rest of the system. This low-level can then be used in any system without having to recheck the low-level's HISC conditions. This would allow a company that sells a manufacturing machine, for instance, to specify an interface, and then model, design, and verify the low-level for the given machine. The company can then provide the interface and low-level as a preverified package, allowing a customer to add the machine to their system, and then just design a high-level to operate the machine's interface. This could represent a considerable saving of effort for a customer.

B. Related Work

The most significant feature which distinguishes this paper from some current work along similar lines (e.g. [22]) is the results on nonblocking, although Endsley et al. later extended their work to include a form of deadlock detection in [40]. Also, the focus of this paper is on how the HISC theory can be applied to provide safe, non-blocking supervision of a non-trivial industrial system.

One way to improve the scalability of modular and decentralized schemes is to exploit the existing architecture of the system. In [41] the concept of a specification that is separable over the component subsystems is introduced and shown to be necessary and sufficient for a decentralized control scheme to exist that optimally meets the specification. The work does not consider nonblocking supervision. These results are extended to a more general architecture in [42] that deals with nonblocking by detecting potential blocking states locally and then backtracking globally to determine their reachability. The structure associated with the event sets of subsystems is exploited in [3] to obtain a reduction in complexity for the non-conflicting check of modular control. Similarly the standard controllability definition has been refined and localized in [43] to check on a per subplant basis only those uncontrollable events that can occur locally.

The scale of the AIP system is much larger than that of the illustrative example system given in [44]. There the open loop system could be represented by a 14 place Petri Net with no more than one token per place in any state. Thus the open-loop state space size has upper bound of $2^{14} = 16384$. The controller has three places which have the restriction that the sum of tokens

for the three places is always 1 (i.e. there are three states for the controller) giving an upper bound on the closed loop state space of size of $3 * 2^{14} = 49152$. The most generous estimate of the state space size is $2^{17} = 131072$ for arbitrary control policies on the three control places. Application of the methods of [44] to an example of the scale of the AIP system would permit a more direct comparison of the scalability of the method. We note that while [44] also relied upon identification of specific structural properties of the system, “the class of flexible manufacturing systems called MRF1 (multiple reentrant flow lines with no self-loops, with jobs that require only one resource, with no two consequent jobs using the same resource, with no choice jobs and no assembly jobs and [sic] with shared resources)”, they consider is not comparable to the class of systems to which HISC can be applied. For instance, the AIP example used to illustrate HISC violates the “no machine failures,” and “no self-loops” conditions, while the MRF1 class of system does not impose that same architectural restriction upon the plant subsystems.

Subsystem	Standalone		with \mathbf{G}_{I_j}		Size of \mathbf{G}_{I_j}	
	States	Transitions	States	Transitions	States	Transitions
\mathbf{G}_H	1,480,864	14,419,512	3,306,240	28,442,432	8,192	104,448
AS1	1,795	4,418	120	191	4	6
AS2	1,795	4,418	120	191	4	6
AS3	1,199	2,448	203	330	2	4
TU1	98	120	98	120	4	7
TU2	98	120	98	120	4	7
TU3	204	266	204	266	4	10
TU4	152	180	152	180	4	7

TABLE I

SIZE OF AIP SUBSYSTEM AUTOMATA MODELS

In [39], Ma *et al.* extended an earlier version (from [28]) of our AIP example by changing *Specification 2* of Section III-C to allow up to 10 pallets in each of the two external loops. Admittedly, their version is larger but it is important to note that they are using *binary decision diagrams* which represent the system as compact predicates. It has been shown that symbolic techniques alone ([45], [38]) can allow significant gains in the size of systems that can be handled, irrespective of hierarchical methods, provided an appropriate variable ordering can be

found to exploit the structure of the system. In our HISC method, we were able to handle the AIP example using only automata based algorithms, that extensionally represent the states and transitions of the DES. Use of symbolic techniques should allow the method to scale up to considerably larger systems.

VI. CONCLUSIONS

Hierarchical interface-based supervisory control offers an effective means to model systems with a natural master-slave structure. Using multiple ($n \geq 1$) low-level subsystems allows the subsystems to be independently modelled and verified, while still allowing a high degree of concurrent operation. Because each of the interface conditions can be verified using a single subsystem and its interface(s), the complete system model never needs to be stored in memory or traversed, offering potentially significant savings in computational resources. This allowed us to be able to quickly verify a large system that was previously far beyond our means. An added benefit is that the independent subsystems can be more easily understood with a higher degree of re-usability. Any individual subsystem can be replaced without requiring the other subsystems to be altered or re-verified.

These benefits were illustrated by a large example (181 DES with a plant model that has an estimated worst case open loop state space size of 2×10^{43}) based on the automated manufacturing system of the Atelier Inter-établissement de Productique (AIP). The example demonstrates how the HISC method can be applied to interesting systems of realistic complexity, even though symbolic techniques have not yet been incorporated into the approach.

A. *Limitations and Future Research*

In the HISC approach, the limiting factor is the state space size of the synchronous product of the high-level subsystem and the interfaces to all of the low-level subsystems. When the interfaces can be designed to have smaller state spaces than the low-level subsystems (true for the AIP example), the state space of the high-level synchronized with the interfaces will be considerably smaller than the state space of the flat system model. However, in the worst case the high-level's state space can still grow exponentially in the number of components. To address this problem, future research will focus on extending the method to a multi-level hierarchy.

Currently, we only provide a method to verify if a system is globally nonblocking and controllable but we do not provide a means to synthesize a maximally permissive supervisor. We intend to address this problem by developing a synthesis method that will respect the interface conditions and thus be able to take advantage of the benefits of our HISC method. The result should be maximally permissive for the system as constrained by the interfaces, but in general will be more restrictive than the maximally permissive behavior without interface constraints.

APPENDIX

EQUIVALENCE OF NEW INTERFACE DEFINITIONS

In the original Hierarchical Interface-based Supervisory Control definitions presented in [29], [30], [28], systems are divided into serial systems (single low level) and parallel systems ($n \geq 1$ low levels), where a serial system is a special case of a parallel system with degree $n = 1$. For each type of system, a set of interface conditions were developed, with the parallel system definitions building upon the serial system definitions.

In this report, we use a slightly modified set of definitions, which no longer treats serial systems (referred to as the “serial case”) and parallel systems (referred to as the “parallel case”) differently. We present a single set of definitions (*interface consistency* from Section II-B, *level-wise nonblocking* from Section II-C, and *level-wise controllable* from Section II-D) for the parallel case that is expressed directly in terms of the components of a parallel system. The new definitions introduced in this work are more concise and clear as a result. The new definitions also make it clear exactly what checks need to be performed, and on which component. In this appendix, we will show that these new definitions are equivalent to the original ones.

In the following sections we will restate the original definitions for ease of reference, adding “(ORIG)” to the titles that are the same as our new definitions to prevent confusion. We will first present the interface consistency definitions, then nonblocking, and finally controllability. We will then show that each is equivalent to the corresponding new definition.

We start by introducing a few useful propositions for later proofs. However, we first need to introduce the following definition:

Definition 6: For natural projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$ for some $\Sigma_o \subseteq \Sigma$, we say that language $L \subseteq \Sigma^*$ is *P_o -invariant* if $P_o^{-1}(P_o(L)) = L$.

In short, events in $\Sigma - \Sigma_o$ (Σ with the events of Σ_o removed) have no affect on membership in L . If L was the language generated by some DES \mathbf{G} , then these events would be selflooped at every state in \mathbf{G} .

Our first proposition is for an n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{L_1}, \mathbf{G}_{L_2}, \dots, \mathbf{G}_{L_n}$ as defined in Section II-B. We first need to define the natural projection, $P_j : \Sigma^* \rightarrow \Sigma'(j)^*$, where Σ is given by (1) in Section II-B, $j \in \{1, \dots, n\}$, and $\Sigma'(j) = \Sigma - \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{L_k}$. The proposition below essentially states that the indicated languages are P_j -invariant, a property that will be useful in the following proofs. The result follows from the fact that the languages are inverse projections of closed and marked languages of DES whose alphabets are subsets of $\Sigma'(j)$.

Proposition 1: With $\mathcal{H}, \mathcal{H}_m, \mathcal{I}_k, \mathcal{I}_{m_k}$ ($k = \{1 \dots n\}$), \mathcal{L}_j , and \mathcal{L}_{m_j} as defined in Section II-B, we have:

- (a) $P_j^{-1}(P_j(\mathcal{H})) = \mathcal{H}$
- (b) $P_j^{-1}(P_j(\mathcal{H}_m)) = \mathcal{H}_m$
- (c) $P_j^{-1}(P_j(\mathcal{I}_k)) = \mathcal{I}_k, k = \{1 \dots n\}$
- (d) $P_j^{-1}(P_j(\mathcal{I}_{m_k})) = \mathcal{I}_{m_k}, k = \{1 \dots n\}$
- (e) $P_j^{-1}(P_j(\mathcal{L}_j)) = \mathcal{L}_j$
- (f) $P_j^{-1}(P_j(\mathcal{L}_{m_j})) = \mathcal{L}_{m_j}$

Proof:

Point (a): Show $P_j^{-1}(P_j(\mathcal{H})) = \mathcal{H}$

By definition, $\mathcal{H} = P_{IH}^{-1}(L(G_H))$. It is thus sufficient to show:

$$P_j^{-1} \cdot P_j \cdot P_{IH}^{-1}(L(G_H)) = P_{IH}^{-1}(L(G_H))$$

We then note that $P_{IH} : \Sigma^* \rightarrow \Sigma_{IH}^*$, and $\Sigma'(j) = \Sigma_{IH} \cup \Sigma_{L_j}$, by definition.

We thus have $\Sigma_{IH} \subseteq \Sigma'(j)$.

We can now apply *Proposition 6* from [28] by taking $\Sigma_b = \Sigma_{IH}$ and $\Sigma_a = \Sigma'(j)$, and conclude:

$$P_j^{-1} \cdot P_j \cdot P_{IH}^{-1} = P_{IH}^{-1}$$

It follows immediately that: $P_j^{-1} \cdot P_j \cdot P_{IH}^{-1}(L(G_H)) = P_{IH}^{-1}(L(G_H))$

Point (b)-(f):

Proofs are identical to **Point (a)** after appropriate substitutions. ■

Our next proposition develops some useful properties of P -invariant languages. **Point (a)** essentially says that removing events from $\Sigma - \Sigma_1$ does not affect membership in languages L_k defined in the proposition. **Point (b)** says that the natural projection P and set intersection commute for P -invariant languages. **Point (c)** says that the intersection of P -invariant languages is also P -invariant. **Point (d)** provides a useful relationship between strings $P(s)$ and s for P -invariant languages.

Proposition 2: Let $\Sigma_1 \subseteq \Sigma$, language $L_k \subseteq \Sigma^*$, $k = 1, 2, \dots, n$, natural projection $P : \Sigma^* \rightarrow \Sigma_1^*$. If $P^{-1}(P(L_k)) = L_k$, then

- (a) $P(L_k) \subseteq L_k$;
- (b) $P(L_1) \cap P(L_2) \cap \dots \cap P(L_n) = P(L_1 \cap L_2 \cap \dots \cap L_n)$;
- (c) $P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)) = L_1 \cap L_2 \cap \dots \cap L_n$.
- (d) $(\forall s \in \Sigma^*) P(s) \in P(L_k) \Rightarrow s \in L_k$

Proof: Assume $P^{-1}(P(L_k)) = L_k$, $k = 1, 2, \dots, n$. **(A.1)**

Point (a): Show $P(L_k) \subseteq L_k$.

Let $s \in P(L_k)$. Sufficient to show $s \in L_k$.

$$s \in P(L_k) \Rightarrow s \in \Sigma_1^*$$

$$\Rightarrow P(s) = s \text{ thus } P(s) \in P(L_k).$$

$$s \in P^{-1}(P(L_k)), \text{ by definition of } P^{-1}.$$

We then have $s \in P^{-1}(P(L_k)) = L_k$ (by **(A.1)**) as required.

Point (b): Show $P(L_1) \cap P(L_2) \cap \dots \cap P(L_n) = P(L_1 \cap L_2 \cap \dots \cap L_n)$

We need to show:

$$\text{(b.I) } P(L_1) \cap P(L_2) \cap \dots \cap P(L_n) \subseteq P(L_1 \cap L_2 \cap \dots \cap L_n) \text{ and}$$

$$\text{(b.II) } P(L_1 \cap L_2 \cap \dots \cap L_n) \subseteq P(L_1) \cap P(L_2) \cap \dots \cap P(L_n).$$

(b.I) Show $P(L_1) \cap P(L_2) \cap \dots \cap P(L_n) \subseteq P(L_1 \cap L_2 \cap \dots \cap L_n)$.

Let $s \in P(L_1) \cap P(L_2) \cap \dots \cap P(L_n)$. **(A.2)**

Must show implies $s \in P(L_1 \cap L_2 \cap \dots \cap L_n)$

By **Point (a)**, **(A.1)**, and **(A.2)**, we have $s \in L_1 \cap L_2 \cap \dots \cap L_n$

$\Rightarrow P(s) \in P(L_1 \cap L_2 \cap \dots \cap L_n)$ **(A.3)**

Since $s \in P(L_1)$ (by **(A.2)**), we have $s \in \Sigma_1^*$ by definition of the natural projection P .

We then have $P(s) = s$, and thus $s \in P(L_1 \cap L_2 \cap \dots \cap L_n)$ (by **(A.3)**), as required.

(b.II) Show $P(L_1 \cap L_2 \cap \dots \cap L_n) \subseteq P(L_1) \cap P(L_2) \cap \dots \cap P(L_n)$.

Let $s \in P(L_1 \cap L_2 \cap \dots \cap L_n)$. **(A.4)**

Must show implies $s \in P(L_1) \cap P(L_2) \cap \dots \cap P(L_n)$.

From **(A.4)**, we know: $(\exists s' \in L_1 \cap L_2 \cap \dots \cap L_n) P(s') = s$ **(A.5)**

We next note that $s' \in L_k \Rightarrow P(s') \in P(L_k)$, $k = 1, \dots, n$.

$\Rightarrow s = P(s') \in P(L_1) \cap P(L_2) \cap \dots \cap P(L_n)$ (by **(A.5)**), as required.

By **Part (b.I)** and **Part (b.II)**, we can now conclude:

$$P(L_1) \cap P(L_2) \cap \dots \cap P(L_n) = P(L_1 \cap L_2 \cap \dots \cap L_n)$$

Point (c): Show $P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)) = L_1 \cap L_2 \cap \dots \cap L_n$

We need to show:

(c.I) $P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)) \subseteq L_1 \cap L_2 \cap \dots \cap L_n$ and

(c.I) $L_1 \cap L_2 \cap \dots \cap L_n \subseteq P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n))$.

(c.I) Show $P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)) \subseteq L_1 \cap L_2 \cap \dots \cap L_n$.

Let $s \in P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n))$ **(A.6)**

Must show implies $s \in L_1 \cap L_2 \cap \dots \cap L_n$

From **(A.6)** and definition of P^{-1} , we have:

$$P(s) \in P(L_1 \cap L_2 \cap \dots \cap L_n).$$

$\Rightarrow P(s) \in P(L_1) \cap P(L_2) \cap \dots \cap P(L_n)$, by **Point (b)**.

$\Rightarrow s \in P^{-1}P(L_k), k = 1, \dots, n.$

$\Rightarrow s \in L_k, k = 1, \dots, n,$ by **(A.1)**.

$\Rightarrow s \in L_1 \cap L_2 \cap \dots \cap L_n,$ as required.

(c.II) Show $L_1 \cap L_2 \cap \dots \cap L_n \subseteq P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)).$

Let $s \in L_1 \cap L_2 \cap \dots \cap L_n$ **(A.7)**

Must show implies $s \in P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n))$

From **(A.7)** and definition of P , we can conclude:

$$P(s) \in P(L_1 \cap L_2 \cap \dots \cap L_n)$$

By definition of P^{-1} , we can conclude:

$$s \in P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n))$$

By **Part (c.I)** and **Part (c.II)**, we can now conclude:

$$P^{-1}(P(L_1 \cap L_2 \cap \dots \cap L_n)) = L_1 \cap L_2 \cap \dots \cap L_n$$

Point (d): Show $(\forall s \in \Sigma^*) P(s) \in P(L_k) \Rightarrow s \in L_k.$

Let $s \in \Sigma^*$ and assume $P(s) \in P(L_k).$

$\Rightarrow s \in P^{-1}(P(L_k)),$ by definition of $P^{-1}.$

$\Rightarrow s \in L_k$ as $P^{-1}(P(L_k)) = L_k$ by **(A.1)**.

■

The next proposition provides a useful result about the controllability definition. If, in the left side of the *iff* condition below, we equate Σ_b with the set of uncontrollable events, L_2 with the language of the plant, L_3 with the language of the supervisor, and $L_1 = L_2 \cap L_3$, we have the controllability definition. The proposition essentially says that if L_k is P -invariant, controllability is not affected by removing all events in $\Sigma - \Sigma_a$.

Proposition 3: For alphabet Σ , with event sets $\Sigma_b \subseteq \Sigma_a \subseteq \Sigma$, languages $L_1, L_2, L_3 \subseteq \Sigma^*$, and natural projection $P : \Sigma^* \rightarrow \Sigma_a^*$, if $P^{-1}(P(L_k)) = L_k, k = 1, 2, 3$, then

$$(\forall s \in L_1) \text{Elig}_{L_2}(s) \cap \Sigma_b \subseteq \text{Elig}_{L_3}(s) \Leftrightarrow (\forall s' \in P(L_1)) \text{Elig}_{P(L_2)}(s') \cap \Sigma_b \subseteq \text{Elig}_{P(L_3)}(s')$$

Proof:

$$\text{Assume } P^{-1}(P(L_k)) = L_k, \quad k = 1, 2, 3. \quad (\mathbf{A.1})$$

By the definition of $\text{Elig}()$ operator, it is sufficient to show:

$$[(\forall s \in L_1)(\forall \sigma \in \Sigma_b) \quad s\sigma \in L_2 \Rightarrow s\sigma \in L_3] \quad \Leftrightarrow \quad (\mathbf{A.2})$$

$$[(\forall s' \in P(L_1))(\forall \sigma' \in \Sigma_b) \quad s'\sigma' \in P(L_2) \Rightarrow s'\sigma' \in P(L_3)] \quad (\mathbf{A.3})$$

We must show: **I** $(\mathbf{A.2}) \Rightarrow (\mathbf{A.3})$ and **II** $(\mathbf{A.3}) \Rightarrow (\mathbf{A.2})$.

(I) Show $(\mathbf{A.2}) \Rightarrow (\mathbf{A.3})$.

Assume $(\mathbf{A.2})$.

$$\text{Let } s' \in P(L_1), \sigma' \in \Sigma_b, \text{ and assume } s'\sigma' \in P(L_2). \quad (\mathbf{A.4})$$

Must show implies $s'\sigma' \in P(L_3)$.

By *Proposition 2(a)* and $(\mathbf{A.1})$, we have $P(L_1) \subseteq L_1$

$\Rightarrow s' \in L_1$, by $(\mathbf{A.4})$.

Similarly, we have $s'\sigma' \in L_2$.

$\Rightarrow s'\sigma' \in L_3$, as, $\sigma' \in \Sigma_b$, and by $(\mathbf{A.2})$

$$\Rightarrow P(s'\sigma') \in P(L_3). \quad (\mathbf{A.5})$$

By $(\mathbf{A.4})$, $s' \in P(L_1) \subseteq \Sigma_1^*$, thus $P(s') = s'$, by definition of P .

$\Rightarrow P(s'\sigma') = s'P(\sigma') = s'\sigma'$, by definition of P , and fact $\Sigma_b \subseteq \Sigma_a$.

$\Rightarrow s'\sigma \in P(L_3)$ (by $(\mathbf{A.5})$), as required.

Part (I) complete.

(II) Show $(\mathbf{A.3}) \Rightarrow (\mathbf{A.2})$.

Assume $(\mathbf{A.3})$.

$$\text{Let } s \in L_1, \sigma \in \Sigma_b, \text{ and assume } s\sigma \in L_2. \quad (\mathbf{A.6})$$

Must show implies $s\sigma \in L_3$.

From **(A.6)**, we have:

$P(s) \in P(L_1)$ and, $P(s\sigma) = P(s)\sigma \in P(L_2)$, by definition of P , and fact $\Sigma_b \subseteq \Sigma_a$.

$\Rightarrow P(s)\sigma \in P(L_3)$, by **(A.3)**, after taking $s' = P(s)$ and $\sigma' = \sigma$.

$\Rightarrow P(s\sigma) \in P(L_3)$, as $\sigma \in \Sigma_b$ and $\Sigma_b \subseteq \Sigma_a$.

$\Rightarrow s\sigma \in L_3$, by **(A.1)**, and *Proposition 2(d)*.

Part (II) complete.

By **Parts (I)** and **(II)**, we have **(A.2)** \Leftrightarrow **(A.3)**, as required. ■

A. Interface Consistency

In the original interface consistency definition, we first defined the *serial interface consistency* definition for the serial system consisting of DES \mathbf{G}'_H , \mathbf{G}_L , and \mathbf{G}_I .³ We then used the concept of *serial system extractions* to extend the serial definition to the parallel case. We will first define some notation for the serial case, restate the original definitions, and then finally we will show that the original interface consistency definition is equivalent to the new one.

We assume that the alphabet partition for a serial system is specified by $\Sigma' := \Sigma'_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$, and then we introduce the following event sets, natural projections, and useful languages:

$$\begin{aligned} \Sigma_I &:= \Sigma_R \dot{\cup} \Sigma_A, & P_{IH'} &: \Sigma'^* \rightarrow \Sigma_{IH'}^* \\ \Sigma_{IH'} &:= \Sigma'_H \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A, & P_{IL} &: \Sigma'^* \rightarrow \Sigma_{IL}^* \\ \Sigma_{IL} &:= \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A, & P_I &: \Sigma'^* \rightarrow \Sigma_I^* \\ \mathcal{H}' &:= P_{IH'}^{-1}(L(\mathbf{G}'_H)), & \mathcal{H}'_m &:= P_{IH'}^{-1}(L_m(\mathbf{G}'_H)) \subseteq \Sigma'^* \\ \mathcal{L} &:= P_{IL}^{-1}(L(\mathbf{G}_L)), & \mathcal{L}_m &:= P_{IL}^{-1}(L_m(\mathbf{G}_L)) \subseteq \Sigma'^* \\ \mathcal{I} &:= P_I^{-1}(L(\mathbf{G}_I)), & \mathcal{I}_m &:= P_I^{-1}(L_m(\mathbf{G}_I)) \subseteq \Sigma'^* \end{aligned}$$

When a system contained only one low level (serial case), we used the serial interface consistency definition given below.

³Some primes (“’”) have been added to avoid confusion with the definitions in Section II-B.

Definition 7: The system composed of DES G'_H , G_L and G_I , is *serial interface consistent* with respect to the alphabet partition $\Sigma' := \Sigma'_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$, if the following properties hold,

Multi-level Properties

- 1) The event set of G'_H is Σ'_{IH} , and the event set of G_L is Σ_{IL} .
- 2) G_I is a command-pair interface.

High Level Properties

- 3) $(\forall s \in \mathcal{H}' \cap \mathcal{I}) \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_A \subseteq \text{Elig}_{\mathcal{H}'}(s)$

Low Level Properties

- 4) $(\forall s \in \mathcal{L} \cap \mathcal{I}) \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_R \subseteq \text{Elig}_{\mathcal{L}}(s)$
- 5) $(\forall s \in \Sigma'^* \cdot \Sigma_R \cap \mathcal{L} \cap \mathcal{I})$

$$\text{Elig}_{\mathcal{L} \cap \mathcal{I}}(s \Sigma_L^*) \cap \Sigma_A = \text{Elig}_{\mathcal{I}}(s) \cap \Sigma_A \quad \text{where}$$

$$\text{Elig}_{\mathcal{L} \cap \mathcal{I}}(s \Sigma_L^*) := \bigcup_{l \in \Sigma_L^*} \text{Elig}_{\mathcal{L} \cap \mathcal{I}}(sl)$$

- 6) $(\forall s \in \mathcal{L} \cap \mathcal{I})$

$$s \in \mathcal{I}_m \Rightarrow (\exists l \in \Sigma_L^*) sl \in \mathcal{L}_m \cap \mathcal{I}_m$$

It's clear that for $n = 1$ and after appropriate relabeling, the interface consistency definition (Definition 3) reduces to the serial interface consistency definition; thus any result (such as in [29]) using the serial interface consistency definition would be immediately satisfied by Definition 3, with $n = 1$.

For the general case ($n \geq 1$ low levels), we need to extend our serial case definitions to the parallel case. As the event set of each low level is disjoint from the event sets of the other low levels, we can consider the parallel interface system as n serial interface systems (referred to as *serial system extractions*) by choosing one low level and ignoring the others. This is shown conceptually in Fig. 32. The full definition is given below.

Definition 8: For the n^{th} degree ($n \geq 1$) parallel interface system composed of DES $G_H, G_{L_1}, \dots, G_{L_n}, G_{I_1}, \dots, G_{I_n}$, with alphabet partition given by (1), the j^{th} *serial system extraction* (subsystem form), denoted by $system(j)$, is composed of the following elements:

$$G'_H(j) := G_H || G_{I_1} || \dots || G_{I_{(j-1)}} || G_{I_{(j+1)}} || \dots || G_{I_n}$$

$$G_L(j) := G_{L_j}, \quad G_I(j) := G_{I_j}$$

$$\Sigma'_H(j) := \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{I_k} \dot{\cup} \Sigma_H$$

$$\begin{aligned}
\Sigma_L(j) &:= \Sigma_{L_j}, & \Sigma_R(j) &:= \Sigma_{R_j}, & \Sigma_A(j) &:= \Sigma_{A_j} \\
\Sigma'(j) &:= \Sigma'_H(j) \dot{\cup} \Sigma_L(j) \dot{\cup} \Sigma_R(j) \dot{\cup} \Sigma_A(j) \\
&= \Sigma - \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{L_k}
\end{aligned}$$

We are now ready to state the original interface consistency definition, for the parallel case.

Definition 9: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $G_H, G_{L_1}, \dots, G_{L_n}, G_{I_1}, \dots, G_{I_n}$, is *interface consistent (ORIG)* with respect to alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction of the system is serial interface consistent.

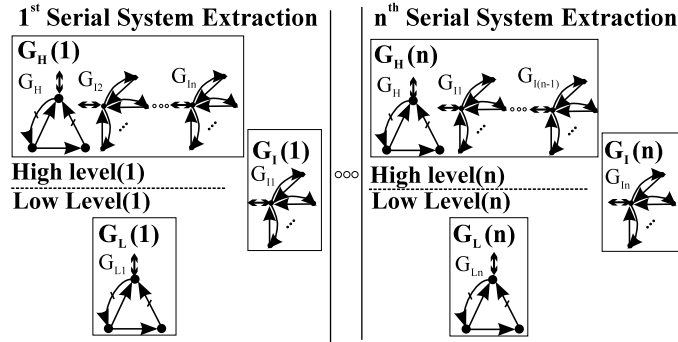


Fig. 32. The Serial System Extractions

Our next step is to introduce an intermediate form of the *interface consistency* definition, created from unrolling the interface consistency (ORIG) definition by applying the serial system extraction. This new form is easily obtainable from Definition 9 and has the same structure as Definition 3. This will make it easier to show that the two definitions are equivalent. To construct this new form of the definition, we will equate the components of a serial extraction system with the components of a serial system, and then interpret the notation of a serial interface system in the obvious way.

Definition 10: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $G_H, G_{I_1}, G_{L_1}, \dots, G_{I_n}, G_{L_n}$, is *interface consistent (IntmORIG)* with respect to the alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$, the following conditions are satisfied:

Multi-level Properties

- 1) The event set of $G'_H(j)$ is Σ_{IH} , and the event set of G_{L_j} is Σ_{IL_j} .
- 2) G_{I_j} is a command-pair interface.

High Level Property

$$3) (\forall s \in \mathcal{H}'(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H}'(j)}(s)$$

Low Level Properties

$$4) (\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}(j)}(s)$$

$$5) (\forall s \in \Sigma^* \cdot \Sigma_{R_j} \cap \mathcal{L}(j) \cap \mathcal{I}(j))$$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \quad \text{where}$$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) := \bigcup_{l \in \Sigma_{L_j}^*} \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(sl)$$

$$6) (\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j))$$

$$s \in \mathcal{I}_m(j) \Rightarrow (\exists l \in \Sigma_{L_j}^*) sl \in \mathcal{L}_m(j) \cap \mathcal{I}_m(j).$$

We will now show that the intermediate form is equivalent to the original form of the interface consistency definition.

Proposition 4: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is interface consistent (IntmORIG) (Definition 10) with respect to alphabet partition given by (1), iff, the system is interface consistent (ORIG) (Definition 9) with respect to alphabet partition given by (1).

Proof:

We will prove this by starting with Definition 9 and converting it into the form of Definition 10.

If Definition 9 is satisfied, then for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction (subsystem form), denoted by $\text{system}(j)$, is serial interface consistent.

That means for all $j \in \{1, \dots, n\}$, the following conditions are satisfied: (A.1)

$$1) \text{ The event set of } G'_H(j) \text{ is } \Sigma'_{IH}(j), \text{ and the event set of } G_L(j) \text{ is } \Sigma_{IL}(j).$$

$$2) G_I(j) \text{ is a command-pair interface.}$$

$$3) (\forall s \in \mathcal{H}'(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_A(j) \subseteq \text{Elig}_{\mathcal{H}'(j)}(s)$$

$$4) (\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_R(j) \subseteq \text{Elig}_{\mathcal{L}(j)}(s)$$

$$5) (\forall s \in \Sigma^*(j) \cdot \Sigma_R(j) \cap \mathcal{L}(j) \cap \mathcal{I}(j))$$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_L^*(j)) \cap \Sigma_A(j) = \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_A(j) \quad \text{where}$$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_L^*(j)) := \bigcup_{l \in \Sigma_L^*(j)} \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(sl)$$

$$6) (\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) \\ s \in \mathcal{I}_m(j) \Rightarrow (\exists l \in \Sigma_L^*(j)) sl \in \mathcal{L}_m(j) \cap \mathcal{I}_m(j)$$

We next note the following facts:

- From Definition 8, we know that $G_L(j) = G_{L_j}$, $G_I(j) = G_{I_j}$, $\Sigma_L(j) = \Sigma_{L_j}$, $\Sigma_R(j) = \Sigma_{R_j}$, $\Sigma_A(j) = \Sigma_{A_j}$, and $\Sigma'(j) = \Sigma - \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{L_k}$ **(A.2)**
- From Proposition 21 in [28], we know that G_{I_j} is defined over event set Σ_{IL_j} . **(A.3)**
- From Proposition 23 in [28], we know that $\Sigma'_{IH}(j) = \Sigma_{IH}$, and $\Sigma_{IL}(j) = \Sigma_{IL_j}$. **(A.4)**
- We note that $\Sigma^*.\Sigma_{R_j} \cap \mathcal{L}(j) \cap \mathcal{I}(j) = \Sigma'^*(j).\Sigma_{R_j} \cap \mathcal{L}(j) \cap \mathcal{I}(j)$ as $\Sigma'^*(j).\Sigma_{R_j} \subseteq \Sigma^*.\Sigma_{R_j}$, $\mathcal{L}(j) \subseteq \Sigma'^*(j)$ thus $(\Sigma^*.\Sigma_{R_j} - \Sigma'^*(j).\Sigma_{R_j}) \cap \mathcal{L}(j) = \emptyset$. **(A.5)**

Now, substituting the results of (A.2) - (A.5) into (A.1), we can conclude that, for all $j \in \{1, \dots, n\}$, the following conditions are satisfied:

- 1) The event set of $\mathbf{G}'_H(j)$ is Σ_{IH} , and the event set of \mathbf{G}_{L_j} is Σ_{IL_j} .
- 2) \mathbf{G}_{I_j} is a command-pair interface.
- 3) $(\forall s \in \mathcal{H}'(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H}'(j)}(s)$
- 4) $(\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}(j)}(s)$
- 5) $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}(j) \cap \mathcal{I}(j))$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \quad \text{where}$$

$$\text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) := \bigcup_{l \in \Sigma_{L_j}^*} \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(sl)$$

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

which is exactly equal to Definition 10, as required. ■

We conclude this section by presenting our theorem that shows that our new definition of interface consistency is equivalent to the original.

Theorem 3: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is interface consistent (Definition 3) with respect to alphabet partition given by (1), iff, the system is interface consistent (ORIG) (Definition 9) with respect to alphabet partition given by (1).

Proof:

We first note that as Definition 10 is equivalent to Definition 9 by *Proposition 4*, it is sufficient to show:

Definition 3 \Leftrightarrow Definition 10

As the two definitions are almost exactly of the same form, we will prove this point by point, for each of the six points of the two definitions.

1) For **point (1)**, the two definitions are almost the same already, thus we only have to account for the difference.

a) Assume Definition 10 is satisfied.

From *Proposition 21* in [28], we can immediately conclude that the event set of \mathbf{G}_H is Σ_{IH} .

b) Assume Definition 3 is satisfied.

Must show this implies the event set of $\mathbf{G}'_H(j)$ is Σ_{IH} .

By definition, $\mathbf{G}'_H(j) = G_H || G_{I_1} || \dots || G_{I_{(j-1)}} || G_{I_{(j+1)}} || \dots || G_{I_n}$.

From **point (1)** of Definition 3, we know that the event set of \mathbf{G}_H is Σ_{IH} . From the command-pair interface definition, and **point (1)** of Definition 3, we know that the event set of G_{I_k} is Σ_{I_k} ($k = 1, \dots, j-1, j+1, \dots, n$).

We thus have the event set of $\mathbf{G}'_H(j)$ is:

$$\Sigma_{\mathbf{G}'_H(j)} = \cup_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{I_k} \cup \Sigma_{IH} = \Sigma_{IH}$$

2) **Point (2)** is automatic.

3) For **Point (3)**, we need to show:

$$\text{(Definition 10)} \quad (\forall s \in \mathcal{H}'(j) \cap \mathcal{I}(j)) \quad \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H}'(j)}(s) \quad \Leftrightarrow \quad \text{(A.1)}$$

$$\text{(Definition 3)} \quad (\forall s \in \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k) \quad \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k}}(s) \quad \text{(A.2)}$$

We will start by massaging **(A.1)** into the correct form so that we can apply *Proposition 3*.

We first note that by *Proposition 1*, the languages \mathcal{H} and \mathcal{I}_k ($k = 1, \dots, n$) are P_j -invariant. That means that we can apply *Proposition 2* to them.

From *Proposition 23* of [28], we have:

$$\begin{aligned}
\mathcal{H}'(j) &= P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, (j-1), (j+1), \dots, n} P_j(\mathcal{I}_k) \\
&= P_j(\mathcal{H}) \cap \bigcap_{k \neq j} P_j(\mathcal{I}_k) \\
&= P_j(\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k), \quad \text{by Proposition 2(b)} \quad (\text{A.3})
\end{aligned}$$

$$\mathcal{I}(j) = P_j(\mathcal{I}_j) \quad (\text{A.4})$$

$$\begin{aligned}
\mathcal{H}'(j) \cap \mathcal{I}(j) &= P_j(\mathcal{H}) \cap \bigcap_{k \neq j} P_j(\mathcal{I}_k) \cap P_j(\mathcal{I}_j) \\
&= P_j(\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k), \quad \text{by Proposition 2(b)} \quad (\text{A.5})
\end{aligned}$$

Substituting from (A.3)-(A.5) into (A.1), we have:

$$(\forall s \in P_j(\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k)) \text{Elig}_{P_j(\mathcal{I}_j)}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{P_j(\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k)}(s) \quad (\text{A.6})$$

By *Proposition 1* and *Proposition 2(c)*, we have:

$$P_j^{-1}(P_j(\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k)) = \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k \text{ and } P_j^{-1}(P_j(\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k)) = \mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k$$

In other words, languages $\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k$ and $\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k$ are P_j -invariant. (A.7)

We next note that, by *Proposition 1*, we have $P_j^{-1}(P_j(\mathcal{I}_j)) = \mathcal{I}_j$, and by definition, we have $\Sigma_{A_j} \subseteq \Sigma'^*(j)$.

We now take $\Sigma_b = \Sigma_{A_j}$, $\Sigma_a = \Sigma'^*(j)$, $P = P_j$, $L_1 = \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k$, $L_2 = \mathcal{I}_j$, $L_3 = \mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k$, and we can conclude by *Proposition 3*, and (A.7) that:

$$\begin{aligned}
&(\forall s \in P_j(\mathcal{H} \cap \bigcap_{k \in \{1, \dots, n\}} \mathcal{I}_k)) \text{Elig}_{P_j(\mathcal{I}_j)}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{P_j(\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k)}(s) \\
&\Leftrightarrow (\forall s \in \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k) \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \subseteq \text{Elig}_{\mathcal{H} \cap \bigcap_{k \neq j} \mathcal{I}_k}(s)
\end{aligned}$$

We can now conclude by (A.6), that (A.1) \Leftrightarrow (A.2), as required.

4) For **Point (4)**, we need to show:

$$(\text{Definition 10}) (\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}(j)}(s) \quad \Leftrightarrow \quad (\text{A.8})$$

$$(\text{Definition 3}) (\forall s \in \mathcal{L}_j \cap \mathcal{I}_j) \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}_j}(s) \quad (\text{A.9})$$

From *Proposition 23* of [28], we have $\mathcal{I}(j) = P_j(\mathcal{I}_j)$, and $\mathcal{L}(j) = P_j(\mathcal{L}_j)$. We also note that by *Proposition 1*, \mathcal{L}_j and \mathcal{I}_j are P_j -invariant. **(A.10)**

$\Rightarrow P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) = P_j(\mathcal{L}_j \cap \mathcal{I}_j)$, by *Proposition 2(b)*.

Substituting this and **(A.10)** into **(A.8)**, we get:

$$(\forall s \in P_j(\mathcal{L}_j \cap \mathcal{I}_j)) \text{Elig}_{P_j(\mathcal{I}_j)}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{P_j(\mathcal{L}_j)}(s) \quad \text{(A.11)}$$

By *Proposition 1* and *Proposition 2(c)*, we have:

$$P_j^{-1}(P_j(\mathcal{L}_j \cap \mathcal{I}_j)) = \mathcal{L}_j \cap \mathcal{I}_j$$

In other words, the language $\mathcal{L}_j \cap \mathcal{I}_j$ is P_j -invariant. **(A.12)**

We next note that, by definition, $\Sigma_{R_j} \subseteq \Sigma'^*(j)$.

We now take $\Sigma_b = \Sigma_{R_j}$, $\Sigma_a = \Sigma'^*(j)$, $L_1 = \mathcal{L}_j \cap \mathcal{I}_j$, $L_2 = \mathcal{I}_j$, $L_3 = \mathcal{L}_j$, and we can conclude by *Proposition 3*, **(A.10)**, and **(A.12)** that:

$$\begin{aligned} & (\forall s \in P_j(\mathcal{L}_j \cap \mathcal{I}_j)) \text{Elig}_{P_j(\mathcal{I}_j)}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{P_j(\mathcal{L}_j)}(s) \\ \Leftrightarrow & (\forall s \in \mathcal{L}_j \cap \mathcal{I}_j) \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{R_j} \subseteq \text{Elig}_{\mathcal{L}_j}(s) \end{aligned}$$

We can now conclude by **(A.11)**, that **(A.8)** \Leftrightarrow **(A.9)**, as required.

5) For **Point (5)**, we need to show:

(Definition 10)

$$(\forall s \in \Sigma^* \cdot \Sigma_{R_j} \cap \mathcal{L}(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}(j)}(s) \cap \Sigma_{A_j} \quad \Leftrightarrow \text{(A.13)}$$

$$\text{where } \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(s \Sigma_{L_j}^*) := \bigcup_{l \in \Sigma_{L_j}^*} \text{Elig}_{\mathcal{L}(j) \cap \mathcal{I}(j)}(sl)$$

$$\text{(Definition 3)} \quad (\forall s \in \Sigma^* \cdot \Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j) \text{Elig}_{\mathcal{L}_j \cap \mathcal{I}_j}(s \Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{\mathcal{I}_j}(s) \cap \Sigma_{A_j} \quad \text{(A.14)}$$

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

From *Proposition 23* of [28], we have $\mathcal{I}(j) = P_j(\mathcal{I}_j)$, and $\mathcal{L}(j) = P_j(\mathcal{L}_j)$. We also note that by *Proposition 1*, \mathcal{L}_j and \mathcal{I}_j are P_j -invariant. **(A.15)**

Substituting into **(A.13)**, we get:

$$\begin{aligned}
& (\forall s \in \Sigma^*.\Sigma_{R_j} \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)) \\
& \text{Elig}_{P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)}(s\Sigma_{L_j}^*) \cap \Sigma_{A_j} = \text{Elig}_{P_j(\mathcal{I}_j)}(s) \cap \Sigma_{A_j} \\
& \text{where } \text{Elig}_{P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)}(s\Sigma_{L_j}^*) := \bigcup_{l \in \Sigma_{L_j}^*} \text{Elig}_{P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)}(sl)
\end{aligned}$$

By the definition of the Elig() operator, it is sufficient to show:

$$(\forall s \in \Sigma^*.\Sigma_{R_j} \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j))(\forall \alpha \in \Sigma_{A_j}) \quad \text{(A.16)}$$

$$[s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset \Leftrightarrow s\alpha \in P_j(\mathcal{I}_j)] \quad \Leftrightarrow$$

$$(\forall s' \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)(\forall \alpha' \in \Sigma_{A_j}) [s'\Sigma_{L_j}^* \alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset \Leftrightarrow s'\alpha' \in \mathcal{I}_j] \quad \text{(A.17)}$$

We first observe that: (A.18)

$$(\forall l \in \Sigma_{L_j}^*) P_j(l) = l, \text{ as } \Sigma_{L_j} \subseteq \Sigma'(j).$$

$$(\forall \rho \in \Sigma_{R_j}) P_j(\rho) = \rho, \text{ as } \Sigma_{R_j} \subseteq \Sigma'(j).$$

$$(\forall \alpha \in \Sigma_{A_j}) P_j(\alpha) = \alpha, \text{ as } \Sigma_{A_j} \subseteq \Sigma'(j).$$

In order to prove **(A.16)** \Leftrightarrow **(A.17)**, we need to show: **(I)** **(A.16)** \Rightarrow **(A.17)** and **(II)** **(A.17)** \Rightarrow **(A.16)**

(I) Show **(A.16)** \Rightarrow **(A.17)**.

Assume **(A.16)**. Must show implies **(A.17)**.

$$\text{Let } s' \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j \text{ and } \alpha' \in \Sigma_{A_j} \quad \text{(A.19)}$$

Must show: **(I.a)** $s'\Sigma_{L_j}^* \alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset \Rightarrow s'\alpha' \in \mathcal{I}_j$ and **(I.b)** $s'\alpha' \in \mathcal{I}_j \Rightarrow s'\Sigma_{L_j}^* \alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset$.

We first note that as $s' \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j$ (by **(A.19)**), it follows that:

$$P_j(s') \in P_j(\Sigma^*.\Sigma_{R_j}) \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$$

As $P_j(\Sigma^*.\Sigma_{R_j}) = \Sigma'^*(j).\Sigma_{R_j}$ (by **(A.18)** and definition of P_j) and $\Sigma'^*(j).\Sigma_{R_j} \subseteq \Sigma^*.\Sigma_{R_j}$ (by definition of $\Sigma'^*(j)$), we have:

$$P_j(s') \in \Sigma^*.\Sigma_{R_j} \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \quad \text{(A.20)}$$

(I.a) Show $s'\Sigma_{L_j}^* \alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset \Rightarrow s'\alpha' \in \mathcal{I}_j$

$$\text{Assume } s'\Sigma_{L_j}^* \alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset \quad \text{(A.21)}$$

Must show implies $s'\alpha' \in \mathcal{I}_j$

From **(A.21)**, we can conclude:

$$\begin{aligned} & (\exists l \in \Sigma_{L_j}^*) s'l\alpha' \in \mathcal{L}_j \cap \mathcal{I}_j \\ \Rightarrow & P_j(s'l\alpha') \in P_j(\mathcal{L}_j \cap \mathcal{I}_j) \\ \Rightarrow & P_j(s'l\alpha') \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j), \text{ by } \mathbf{(A.15)}, \mathbf{(A.18)}, \text{ and } \textit{Proposition 2(b)}. \\ \Rightarrow & P_j(s')\Sigma_{L_j}^*\alpha' \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset \end{aligned} \tag{A.22}$$

By **(A.20)**, **(A.19)**, **(A.22)**, and taking $s = P_j(s')$, $\alpha = \alpha'$, we can apply **(A.16)** and conclude:

$$\begin{aligned} & P_j(s')\alpha' \in P_j(\mathcal{I}_j). \\ \Rightarrow & P_j(s'\alpha') \in P_j(\mathcal{I}_j), \text{ by } \mathbf{(A.18)} \\ \Rightarrow & s'\alpha' \in \mathcal{I}_j, \text{ by } \mathbf{(A.15)}, \text{ and } \textit{Proposition 2(d)}. \end{aligned}$$

Part (I.a) complete.

(I.b) Show $s'\alpha' \in \mathcal{I}_j \Rightarrow s'\Sigma_{L_j}^*\alpha' \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset$.

Let $s'\alpha' \in \mathcal{I}_j$. **(A.23)**

Must show $(\exists l \in \Sigma_{L_j}^*) s'l\alpha' \in \mathcal{L}_j \cap \mathcal{I}_j$.

From **(A.23)**, we can conclude:

$$\begin{aligned} & P_j(s'\alpha') \in P_j(\mathcal{I}_j) \\ \Rightarrow & P_j(s')\alpha' \in P_j(\mathcal{I}_j), \text{ by } \mathbf{(A.18)}. \end{aligned} \tag{A.24}$$

By **(A.20)**, **(A.19)**, **(A.24)**, and taking $s = P_j(s')$, $\alpha = \alpha'$, we can apply **(A.16)** and conclude:

$$\begin{aligned} & (\exists l \in \Sigma_{L_j}^*) P_j(s')l\alpha' \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \\ \Rightarrow & P_j(s'l\alpha') \in P_j(\mathcal{L}_j \cap \mathcal{I}_j), \text{ by } \mathbf{(A.18)}, \mathbf{(A.15)}, \text{ and } \textit{Proposition 2(b)}. \\ \Rightarrow & s'l\alpha' \in \mathcal{L}_j \cap \mathcal{I}_j, \text{ by } \mathbf{(A.15)}, \text{ and } \textit{Proposition 2(c), (d)}. \end{aligned}$$

Part (I.b) complete.

By **Parts (I.a)** and **(I.b)**, we have **(A.16)** \Rightarrow **(A.17)**.

Part (I) complete.

(II) Show **(A.17)** \Rightarrow **(A.16)**.

Assume **(A.17)**. Must show implies **(A.16)**.

Let $s \in \Sigma^* \cdot \Sigma_{R_j} \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$ and $\alpha \in \Sigma_{A_j}$. **(A.25)**

Must show: **(II.a)** $s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset \Rightarrow s\alpha \in P_j(\mathcal{I}_j)$ and **(II.b)** $s\alpha \in P_j(\mathcal{I}_j) \Rightarrow s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset$.

We first note that by **(A.25)**, we have:

$$s \in P_j(\mathcal{L}_j) \text{ and } s \in P_j(\mathcal{I}_j)$$

$\Rightarrow s \in \mathcal{L}_j \cap \mathcal{I}_j$ by **(A.15)** and *Proposition 2(a)*.

$\Rightarrow s \in \Sigma^* \cdot \Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j$, by **(A.25)**. **(A.26)**

(II.a) Show $s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset \Rightarrow s\alpha \in P_j(\mathcal{I}_j)$.

Assume $s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset$ **(A.27)**

Must show implies $s\alpha \in P_j(\mathcal{I}_j)$.

From **(A.27)**, we can conclude:

$$(\exists l \in \Sigma_{L_j}^*) sl\alpha \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$$

$\Rightarrow sl\alpha \in \mathcal{L}_j \cap \mathcal{I}_j$, by **(A.15)** and *Proposition 2(a)*.

$\Rightarrow s\Sigma_{L_j}^* \alpha \cap \mathcal{L}_j \cap \mathcal{I}_j \neq \emptyset$ **(A.28)**

By **(A.25)**, **(A.26)**, **(A.28)**, and taking $s' = s$ and $\alpha' = \alpha$, we can apply **(A.17)** and conclude:

$$s\alpha \in \mathcal{I}_j$$

$\Rightarrow P_j(s\alpha) \in P_j(\mathcal{I}_j)$ **(A.29)**

We first note that by **(A.25)**, we have $s \in P_j(\mathcal{L}_j) \subseteq \Sigma'^*(j)$

$\Rightarrow P_j(s\alpha) = s\alpha$, by **(A.18)**, and definition of P_j .

$\Rightarrow s\alpha \in P_j(\mathcal{I}_j)$, by **(A.29)**.

Part (II.a) complete.

(II.b) Show $s\alpha \in P_j(\mathcal{I}_j) \Rightarrow s\Sigma_{L_j}^* \alpha \cap P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \neq \emptyset$.

Assume $s\alpha \in P_j(\mathcal{I}_j)$. **(A.30)**

Must show $(\exists l \in \Sigma_{L_j}^*) sl\alpha \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$.

From **(A.15)**, **(A.30)**, and *Proposition 2(a)*, we can conclude: $s\alpha \in \mathcal{I}_j$ **(A.31)**

By **(A.26)**, **(A.25)**, **(A.31)**, and taking $s' = s$ and $\alpha' = \alpha$, we can apply **(A.17)** and conclude:

$$(\exists l \in \Sigma_{L_j}^*) sl\alpha \in \mathcal{L}_j \cap \mathcal{I}_j \quad \textbf{(A.32)}$$

$$\Rightarrow P_j(sl\alpha) \in P_j(\mathcal{L}_j \cap \mathcal{I}_j)$$

$$\Rightarrow P_j(sl\alpha) \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j), \text{ by } \textbf{(A.15)} \text{ and } \textit{Proposition 2(b)}.$$

We next note that by **(A.25)**, we have $s \in P_j(\mathcal{L}_j) \subseteq \Sigma'^*(j)$

$$\Rightarrow P_j(sl\alpha) = sl\alpha, \text{ by } \textbf{(A.18)}, \textbf{(A.32)}, \text{ and definition of } P_j.$$

$$\Rightarrow sl\alpha \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j), \text{ as required.}$$

Part (II.b) complete.

By **Parts (II.a)** and **(II.b)**, we have **(A.17)** \Rightarrow **(A.16)**.

Part (II) complete.

By **Parts (I)** and **Part (II)**, we have **(A.16)** \Leftrightarrow **(A.17)**, as required.

6) For **Point (6)**, we need to show:

(Definition 10)

$$(\forall s \in \mathcal{L}(j) \cap \mathcal{I}(j)) s \in \mathcal{I}_m(j) \Rightarrow (\exists l \in \Sigma_{L_j}^*) sl \in \mathcal{L}_m(j) \cap \mathcal{I}_m(j) \quad \Leftrightarrow \textbf{(A.33)}$$

$$\textbf{(Definition 3)} (\forall s' \in \mathcal{L}_j \cap \mathcal{I}_j) s' \in \mathcal{I}_{m_j} \Rightarrow (\exists l' \in \Sigma_{L_j}^*) s'l' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j} \quad \textbf{(A.34)}$$

From *Proposition 23* of [28], we have $\mathcal{I}(j) = P_j(\mathcal{I}_j)$, $\mathcal{L}(j) = P_j(\mathcal{L}_j)$, $\mathcal{I}_m(j) = P_j(\mathcal{I}_{m_j})$,

and $\mathcal{L}_m(j) = P_j(\mathcal{L}_{m_j})$. We also note that by *Proposition 1*, \mathcal{L}_j , \mathcal{I}_j , \mathcal{L}_{m_j} , and \mathcal{I}_{m_j} are P_j -invariant. (A.35)

Substituting into (A.33), we get:

$$(\forall s \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)) s \in P_j(\mathcal{I}_{m_j}) \Rightarrow (\exists l \in \Sigma_{L_j}^*) sl \in P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j}) \quad (\text{A.36})$$

It is thus sufficient to show (A.36) \Leftrightarrow (A.34).

To do this, we need to show: (I) (A.36) \Rightarrow (A.34) and (II) (A.34) \Rightarrow (A.36).

(I) Show (A.36) \Rightarrow (A.34).

Assume (A.36). Must show implies (A.34).

Let $s' \in \mathcal{L}_j \cap \mathcal{I}_j$. Assume $s' \in \mathcal{I}_{m_j}$. (A.37)

We must show implies $(\exists l' \in \Sigma_{L_j}^*) s'l' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

From (A.37), we can conclude:

$$\begin{aligned} & P_j(s') \in P_j(\mathcal{L}_j \cap \mathcal{I}_j) \text{ and } P_j(s') \in P_j(\mathcal{I}_{m_j}) \\ \Rightarrow & P_j(s') \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \text{ and } P_j(s') \in P_j(\mathcal{I}_{m_j}) \text{ by (A.35), and Proposition 2(b).} \end{aligned}$$

We can now apply (A.36) by taking $s = P_j(s')$, and conclude:

$$(\exists l' \in \Sigma_{L_j}^*) P_j(s')l' \in P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j}).$$

As $\Sigma_{L_j}^* \subseteq \Sigma'(j)$, and by (A.35), *Proposition 2(b)*, and definition of P_j , we can conclude:

$$P_j(s'l') \in P_j(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}).$$

We can now conclude by (A.35), and *Proposition 2(c)*, (d):

$$s'l' \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}, \text{ as required.}$$

Part (I) complete.

(II) Show (A.34) \Rightarrow (A.36).

Assume (A.34). Must show implies (A.36).

Let $s \in P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$, and assume $s \in P_j(\mathcal{I}_{m_j})$. (A.38)

We must show implies $(\exists l \in \Sigma_{L_j}^*) sl \in P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j})$.

From **(A.38)**, we can conclude:

$$s \in P_j(\mathcal{L}_j), s \in P_j(\mathcal{I}_j), \text{ and } s \in P_j(\mathcal{I}_{m_j})$$

$\Rightarrow s \in \mathcal{L}_j \cap \mathcal{I}_j$ and $s \in \mathcal{I}_{m_j}$, by **(A.35)**, and *Proposition 2(a)*.

We can now apply **(A.34)** by taking $s' = s$, and conclude:

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

$$\Rightarrow P_j(sl) \in P_j(\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j})$$

$\Rightarrow P_j(sl) \in P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j})$, by **(A.35)**, and *Proposition 2(b)*.

We next note that by **(A.38)**, we have $s \in P_j(\mathcal{L}_j) \subseteq \Sigma'^*(j)$.

We thus have $P_j(sl) = sl$ as $\Sigma_{L_j}^* \subseteq \Sigma'^*(j)$, and by definition of P_j .

$\Rightarrow sl \in P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j})$, as required.

Part (II) complete.

By **Parts (I)** and **(II)**, we have **(A.36)** \Leftrightarrow **(A.34)**, as required.

We have now shown that all six points of the two definitions are equivalent, and we can thus conclude that the parallel system satisfies Definition 3 *iff* it satisfies Definition 9. ■

B. Level-wise Nonblocking

In the original level-wise nonblocking definition, we first defined the *serial level-wise nonblocking* definition for the serial system consisting of DES \mathbf{G}'_H , \mathbf{G}_L , and \mathbf{G}_I . We then used the concept of serial system extractions (Definition 8) to extend the serial definition to the parallel case.

We now restate the serial level-wise nonblocking definition below. It is clear that for $n = 1$ and after appropriate relabeling, the level-wise nonblocking definition (Definition 4) reduces to the serial level-wise nonblocking definition; thus any result (such as in [29]) using the serial level-wise nonblocking definition would be immediately satisfied by Definition 4, with $n = 1$.

Definition 11: The system composed of DES G'_H , G_L , and G_I , is said to be *serial level-wise nonblocking* with respect to the alphabet partition $\Sigma' := \Sigma'_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$, if the following conditions are satisfied:

$$(I) \quad \overline{\mathcal{H}'_m \cap \mathcal{I}_m} = \mathcal{H}' \cap \mathcal{I}$$

$$(II) \quad \overline{\mathcal{L}_m \cap \mathcal{I}_m} = \mathcal{L} \cap \mathcal{I}$$

We now define the original level-wise nonblocking definition by extending the serial level-wise nonblocking definition to the parallel case, using Definition 8.

Definition 12: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES G_H , $G_{L_1}, \dots, G_{L_n}, G_{I_1}, \dots, G_{I_n}$, is *level-wise nonblocking (ORIG)* with respect to the alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction of the system is serial level-wise nonblocking.

Before we can prove the equivalence of our new level-wise nonblocking definition, we first need a useful corollary. In Section A, we made extensive use of *Proposition 23* in [28]. However, this proposition required that the parallel system be interface consistent (ORIG). However, for the parts of *Proposition 23* in [28] we need for our next theorem, a weaker condition is sufficient. The restriction on the alphabet of the DES belonging to the parallel system that we use has always been implicit in the level-wise nonblocking definitions; we are only make it explicit so we can use it in the corollary below.

Corollary 1: If the n^{th} degree ($n \geq 1$) parallel interface system composed of DES G_H , $G_{L_1}, \dots, G_{L_n}, G_{I_1}, \dots, G_{I_n}$ with alphabet partition given by (1), is as defined in Section II-B with respect to the alphabets of the given DES, then for the j^{th} serial system extraction, *system(j)*, the following is true:

- (i) The following event sets are: $\Sigma_I(j) = \Sigma_{I_j}$, $\Sigma_{IH'}(j) = \Sigma_{IH}$, and $\Sigma_{IL}(j) = \Sigma_{IL_j}$
- (ii) The following inverse natural projections are: $P_{IH'}(j)^{-1} = P_j \cdot P_{IH}^{-1}$, $P_{IL}(j)^{-1} = P_j \cdot P_{IL_j}^{-1}$, and $P_I(j)^{-1} = P_j \cdot P_{I_j}^{-1}$
- (iii) The indicated languages satisfy the following statements:

$$\mathcal{H}'(j) = P_j(\mathcal{H}) \cap [\cap_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} P_j(\mathcal{I}_k)]$$

$$\mathcal{H}'_m(j) = P_j(\mathcal{H}_m) \cap [\cap_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} P_j(\mathcal{I}_{m_k})]$$

$$\mathcal{L}(j) = P_j(\mathcal{L}_j)$$

$$\mathcal{L}_m(j) = P_j(\mathcal{L}_{m_j})$$

$$\begin{aligned}\mathcal{I}(j) &= P_j(\mathcal{I}_j) \\ \mathcal{I}_m(j) &= P_j(\mathcal{I}_{m_j})\end{aligned}$$

Proof: Results follow immediately from the proofs of the corresponding parts of *Proposition 23* in [28]. ■

We conclude this section by presenting our theorem that shows that our new definition of level-wise nonblocking is equivalent to the original.

Theorem 4: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$ as defined in Section II-B with respect to the alphabets of the given DES, is level-wise nonblocking (Definition 4) with respect to alphabet partition given by (1), iff, the system is level-wise nonblocking (ORIG) (Definition 12) with respect to alphabet partition given by (1).

Proof:

Assume that the n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{L_1}, \dots, \mathbf{G}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$ is defined as in Section II-B with respect to the alphabets of the given DES. (A.1)

We start by converting Definition 12 into a more useful form.

If Definition 12 is satisfied, then for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction (subsystem form), denoted by $system(j)$, is serial level-wise nonblocking.

We thus have Definition 12 is equivalent to:

$$\begin{aligned}(\forall j \in \{1, \dots, n\}), \text{ system}(j) \text{ satisfies:} & \tag{A.2} \\ \text{(I) } \overline{\mathcal{H}'_m(j)} \cap \overline{\mathcal{I}_m(j)} &= \mathcal{H}'(j) \cap \mathcal{I}(j) \\ \text{(II) } \overline{\mathcal{L}_m(j)} \cap \overline{\mathcal{I}_m(j)} &= \mathcal{L}(j) \cap \mathcal{I}(j)\end{aligned}$$

We can now apply *Corollary 1* and conclude that $\mathcal{H}'(j) = P_j(\mathcal{H}) \cap [\bigcap_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} P_j(\mathcal{I}_k)]$, $\mathcal{H}'_m(j) = P_j(\mathcal{H}_m) \cap [\bigcap_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} P_j(\mathcal{I}_{m_k})]$, $\mathcal{L}(j) = P_j(\mathcal{L}_j)$, $\mathcal{L}_m(j) = P_j(\mathcal{L}_{m_j})$, $\mathcal{I}(j) = P_j(\mathcal{I}_j)$, and $\mathcal{I}_m(j) = P_j(\mathcal{I}_{m_j})$.

Substituting into (A.2) and simplifying, we find that Definition 12 is equivalent to: (A.3)

$$\text{(I) for all } j \in \{1, \dots, n\}, \quad \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})} = P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k)$$

$$(II) \text{ for all } j \in \{1, \dots, n\}, \quad \overline{P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j})} = P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j)$$

In order to prove the system satisfies Definition 12 *iff* it satisfies Definition 4, it is thus sufficient to show **(A.3)** *iff* Definition 4.

As **(A.3)** and Definition 4 are of the same form, we will prove equivalence point by point.

We first note that for all $j \in \{1, \dots, n\}$, the languages \mathcal{H} , \mathcal{H}_m , \mathcal{I}_k , \mathcal{I}_{m_k} ($k \in \{1 \dots n\}$), \mathcal{L}_j , and \mathcal{L}_{m_j} are P_j -invariant, by *Proposition 1*. **(A.4)**

(I) For **Point I**, we need to show:

$$(A.3) \forall j \in \{1, \dots, n\}, \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})} = P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k) \quad \Leftrightarrow \quad (A.5)$$

$$(Definition 4) \quad \overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}} = \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k \quad (A.6)$$

To do this, we need to show: **(I.a)** **(A.5)** \Rightarrow **(A.6)** and **(I.b)** **(A.6)** \Rightarrow **(A.5)**.

(I.a) Show **(A.5)** \Rightarrow **(A.6)**.

Assume **(A.5)**. Must show implies **(A.6)**.

As $\overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}} \subseteq \mathcal{H} \cap \bigcap_{k \in \{1 \dots n\}} \mathcal{I}_k$ is automatic, we only need to show:

$$\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k \subseteq \overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}}.$$

Let $s \in \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k$ **(A.7)**

Must show implies $s \in \overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}}$.

Assume $j \in \{1, \dots, n\}$. From **(A.7)**, we can conclude:

$$P_j(s) \in P_j(\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k)$$

By **(A.4)** and *Proposition 2(b)*, we have:

$$P_j(s) \in P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k)$$

By **(A.5)**, we can conclude:

$$P_j(s) \in \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})}$$

$$\Rightarrow (\exists s' \in \Sigma'(j)^*) P_j(s)s' \in P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})$$

As $s' \in \Sigma'(j)^*$, we thus have $P_j(s') = s'$, by definition of P_j .

$$\Rightarrow P_j(ss') \in P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k}), \text{ as } P_j \text{ is catenative.}$$

By **(A.4)** and *Proposition 2(b)*, we have:

$$P_j(ss') \in P_j(\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k})$$

$\Rightarrow ss' \in \mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}$, by **(A.4)**, and *Proposition 2(c),(d)*.

$\Rightarrow s \in \overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}}$, as required.

Part (I.a) complete.

(I.b) Show **(A.6)** \Rightarrow **(A.5)**.

Assume **(A.6)**. Must show implies **(A.5)**.

Let $j \in \{1, \dots, n\}$. As $\overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})} \subseteq P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k)$ is automatic, we only need to show:

$$P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k) \subseteq \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})}$$

Let $s \in P_j(\mathcal{H}) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_k)$ **(A.8)**

Must show implies $s \in \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})}$.

From **(A.4)**, **(A.8)**, and *Proposition 2(b)*, we can conclude:

$$s \in P_j(\mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k)$$

$\Rightarrow s \in \mathcal{H} \cap \bigcap_{k=1, \dots, n} \mathcal{I}_k$, by **(A.4)**, and *Proposition 2(c),(a)*.

By **(A.6)**, we can conclude:

$$s \in \overline{\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}}$$

$\Rightarrow (\exists s' \in \Sigma^*) ss' \in \mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k}$

$\Rightarrow P_j(ss') \in P_j(\mathcal{H}_m \cap \bigcap_{k=1, \dots, n} \mathcal{I}_{m_k})$

$\Rightarrow P_j(ss') \in P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})$, by **(A.4)**, and *Proposition 2(b)*.

$\Rightarrow P_j(s)P_j(s') \in P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})$, as P_j is catenative.

As $s \in P_j(\mathcal{H}) \subseteq \Sigma'(j)^*$ (by **(A.8)**), we have $P_j(s) = s$ (by definition of P_j).

$\Rightarrow sP_j(s') \in P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})$

$s \in \overline{P_j(\mathcal{H}_m) \cap \bigcap_{k=1, \dots, n} P_j(\mathcal{I}_{m_k})}$, as required.

Part (I.b) complete.

By **Parts (I.a)** and **(I.b)**, we have **(A.5)** \Leftrightarrow **(A.6)**, as required.

(II) For **Point II**, and letting $j \in \{1, \dots, n\}$, we need to show:

$$\text{(A.3)} \quad \overline{P_j(\mathcal{L}_{m_j}) \cap P_j(\mathcal{I}_{m_j})} = P_j(\mathcal{L}_j) \cap P_j(\mathcal{I}_j) \quad \Leftrightarrow \quad \text{(A.9)}$$

$$\text{(Definition 4)} \quad \overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j \quad \text{(A.10)}$$

The proof here is identical to **Part I**, after appropriate relabelling.

By **Parts I** and **II**, we can conclude **(A.3)** iff Definition 4, as required. ■

C. Level-wise Controllability

In the original level-wise controllability definition, we first defined the *serial level-wise controllability* definition for the serial system consisting of plant components $\mathbf{G}_H^{p'}$, \mathbf{G}_L^p , supervisors \mathbf{S}'_H , \mathbf{S}_L , and interface G_I . We then used the concept of serial system extractions (Definition 14 below) to extend the serial definition to the parallel case.

We assume that the alphabet partition for a serial system is specified by $\Sigma' := \Sigma'_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$, and then we introduce the following useful languages:

$$\begin{aligned} \mathcal{H}^{p'} &:= P_{IH}^{-1}L(\mathbf{G}_H^{p'}), & \mathcal{S}'_H &:= P_{IH}^{-1}L(\mathbf{S}'_H), & \subseteq \Sigma^* \\ \mathcal{L}^p &:= P_{IL}^{-1}L(\mathbf{G}_L^p), & \mathcal{S}_L &:= P_{IL}^{-1}L(\mathbf{S}_L), & \subseteq \Sigma^* \end{aligned}$$

We now restate the serial level-wise controllability definition below. It's clear that for $n = 1$ and after appropriate relabelling, the level-wise controllability definition (Definition 5) reduces to the serial level-wise controllability definition; thus any result (such as in [29]) using the serial level-wise controllability definition would be immediately satisfied by Definition 5, with $n = 1$.

Definition 13: The system composed of plant components $\mathbf{G}_H^{p'}$, \mathbf{G}_L^p , supervisors \mathbf{S}'_H , \mathbf{S}_L , and interface \mathbf{G}_I , is said to be *serial level-wise controllable* with respect to the alphabet partition $\Sigma' := \Sigma'_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$, if the following conditions are satisfied:

- (I) The alphabet of $\mathbf{G}_H^{p'}$ and \mathbf{S}'_H is Σ_{IH} , the alphabet of \mathbf{G}_L^p and \mathbf{S}_L is Σ_{IL} , and the alphabet of \mathbf{G}_I is Σ_I .

$$(II) (\forall s \in \mathcal{L}^p \cap \mathcal{S}_L \cap \mathcal{I}) \text{Elig}_{\mathcal{L}^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_L \cap \mathcal{I}}(s)$$

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

We now restate the general form of the serial system extraction needed for the controllability definition. We simply refer to the j^{th} serial system extraction, as the type of the parallel system (general form or subsystem form) will make clear which definition is intended.

Definition 14: For the n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \dots, \mathbf{G}_{L_n}^p, \mathbf{S}_H, \mathbf{S}_{L_1}, \dots, \mathbf{S}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, with alphabet partition given by (1), the j^{th} serial system extraction (general form), denoted by $\text{system}(j)$, is composed of the following elements:

$$\begin{aligned} \mathbf{G}_H^{p'}(j) &:= \mathbf{G}_H^p \parallel \mathbf{G}_{I_1} \parallel \dots \parallel \mathbf{G}_{I_{(j-1)}} \parallel \mathbf{G}_{I_{(j+1)}} \parallel \dots \parallel \mathbf{G}_{I_n} \\ \mathbf{S}'_H(j) &:= \mathbf{S}_H, \quad \mathbf{G}_L^p(j) := \mathbf{G}_{L_j}^p, \quad \mathbf{S}_L(j) := \mathbf{S}_{L_j}, \quad \mathbf{G}_I(j) := \mathbf{G}_{I_j} \\ \Sigma'_H(j) &:= \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{I_k} \dot{\cup} \Sigma_H \\ \Sigma_L(j) &:= \Sigma_{L_j}, \quad \Sigma_R(j) := \Sigma_{R_j}, \quad \Sigma_A(j) := \Sigma_{A_j} \\ \Sigma'(j) &:= \Sigma'_H(j) \dot{\cup} \Sigma_L(j) \dot{\cup} \Sigma_R(j) \dot{\cup} \Sigma_A(j) \\ &= \Sigma - \dot{\cup}_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{L_k} \end{aligned}$$

We are now ready to state the original level-wise controllability definition, for the parallel case.

Definition 15: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \dots, \mathbf{G}_{L_n}^p, \mathbf{S}_H, \mathbf{S}_{L_1}, \dots, \mathbf{S}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is *level-wise controllable (ORIG)* with respect to alphabet partition given by (1), if for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction of the system is serial level-wise controllable.

We now need to provide a counterpart to *Proposition 1* for languages of a general form system. For $j \in \{1, \dots, n\}$, the proposition below essentially states that the indicated languages are P_j -invariant.

Proposition 5: With $\mathcal{H}^p, \mathcal{S}_H, \mathcal{L}_j^p$, and \mathcal{S}_{L_j} as defined in Section II-D, we have:

- (a) $P_j^{-1}(P_j(\mathcal{H}^p)) = \mathcal{H}^p$
- (b) $P_j^{-1}(P_j(\mathcal{S}_H)) = \mathcal{S}_H$
- (c) $P_j^{-1}(P_j(\mathcal{L}_j^p)) = \mathcal{L}_j^p$
- (d) $P_j^{-1}(P_j(\mathcal{S}_{L_j})) = \mathcal{S}_{L_j}$

Proof:

Point (a)-(d) Proofs are identical to **Point (a)** of *Proposition 1* after appropriate substitutions. ■

We conclude this section by presenting our theorem that shows that our new definition of level-wise controllability is equivalent to the original.

Theorem 5: The n^{th} degree ($n \geq 1$) parallel interface system composed of DES $\mathbf{G}_H^p, \mathbf{G}_{L_1}^p, \dots, \mathbf{G}_{L_n}^p, \mathbf{S}_H, \mathbf{S}_{L_1}, \dots, \mathbf{S}_{L_n}, \mathbf{G}_{I_1}, \dots, \mathbf{G}_{I_n}$, is level-wise controllable (Definition 5) with respect to alphabet partition given by (1), iff, the system is level-wise controllable (ORIG) (Definition 15) with respect to alphabet partition given by (1).

Proof:

We start by converting Definition 15 into a more useful form.

If Definition 15 is satisfied, then for all $j \in \{1, \dots, n\}$, the j^{th} serial system extraction (general form), denoted by $\text{system}(j)$, is serial level-wise controllable.

We thus have Definition 15 is equivalent to:

($\forall j \in \{1, \dots, n\}$), $\text{system}(j)$ satisfies: (A.1)

(I) The alphabet of $\mathbf{G}_H^{p'}(j)$ and $\mathbf{S}'_H(j)$ is $\Sigma_{IH}(j)$, the alphabet of $\mathbf{G}_L^p(j)$ and $\mathbf{S}_L(j)$ is $\Sigma_{IL}(j)$, and the alphabet of $\mathbf{G}_I(j)$ is $\Sigma_I(j)$.

(II) ($\forall s \in \mathcal{L}^p(j) \cap \mathcal{S}_L(j) \cap \mathcal{I}(j)$) $\text{Elig}_{\mathcal{L}^p(j)}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_L(j) \cap \mathcal{I}(j)}(s)$

(III) ($\forall s \in \mathcal{H}^{p'}(j) \cap \mathcal{I}(j) \cap \mathcal{S}'_H(j)$) $\text{Elig}_{\mathcal{H}^{p'}(j) \cap \mathcal{I}(j)}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}'_H(j)}(s)$

It is thus sufficient to show that system satisfies (A.1) iff it satisfies Definition 5.

As (A.1) and Definition 5 are of the same form, we will prove equivalence point by point.

(I) For **Point I**, we must show:

(A.1) ($\forall j \in \{1, \dots, n\}$) the alphabet of $\mathbf{G}_H^{p'}(j)$ and $\mathbf{S}'_H(j)$ is $\Sigma_{IH}(j)$, the (A.2)
alphabet of $\mathbf{G}_L^p(j)$ and $\mathbf{S}_L(j)$ is $\Sigma_{IL}(j)$, and the alphabet of $\mathbf{G}_I(j)$ is $\Sigma_I(j)$ \Leftrightarrow

(Defn. 5) ($\forall j \in \{1, \dots, n\}$) the alphabet of \mathbf{G}_H^p and \mathbf{S}_H is Σ_{IH} , the alphabet of (A.3)
 $\mathbf{G}_{L_j}^p$ and \mathbf{S}_{L_j} is Σ_{IL_j} , and the alphabet of \mathbf{G}_{I_j} is Σ_{I_j}

(I.a) Show (A.2) \Rightarrow (A.3)

Assume **(A.2)**. Must show implies **(A.3)**.

This follows immediately from *Proposition 28* of [28].

(I.b) Show **(A.3)** \Rightarrow **(A.2)**

Assume **(A.3)**. Must show implies **(A.2)**.

By definition, $\mathbf{G}_H^{p'}(j) = \mathbf{G}_H^p \parallel \mathbf{G}_{I_1} \parallel \dots \parallel \mathbf{G}_{I_{(j-1)}} \parallel \mathbf{G}_{I_{(j+1)}} \parallel \dots \parallel \mathbf{G}_{I_n}$, $\mathbf{S}'_H(j) = \mathbf{S}_H$,
 $\mathbf{G}_L^p(j) = \mathbf{G}_{L_j}^p$, $\mathbf{S}_L(j) = \mathbf{S}_{L_j}$, and $\mathbf{G}_I(j) := \mathbf{G}_{I_j}$. **(A.4)**

From *Proposition 30* of [28], we can conclude:

$$\Sigma_{IH}(j) = \Sigma_{IH}, \quad \Sigma_{IL}(j) = \Sigma_{IL_j}, \quad \text{and} \quad \Sigma_I(j) = \Sigma_{I_j}.$$

Combining with **(A.4)** and substituting into **(A.2)**, we can conclude that it is sufficient to show that:

$(\forall j \in \{1, \dots, n\})$ the alphabet of $\mathbf{G}_H^p \parallel \mathbf{G}_{I_1} \parallel \dots \parallel \mathbf{G}_{I_{(j-1)}} \parallel \mathbf{G}_{I_{(j+1)}} \parallel \dots \parallel \mathbf{G}_{I_n}$ 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} **(A.5)**

All of this follows immediately from **(A.3)**, except showing that the alphabet of $\mathbf{G}_H^{p'}(j) = \mathbf{G}_H^p \parallel \mathbf{G}_{I_1} \parallel \dots \parallel \mathbf{G}_{I_{(j-1)}} \parallel \mathbf{G}_{I_{(j+1)}} \parallel \dots \parallel \mathbf{G}_{I_n}$ is Σ_{IH} .

From the definition of the synchronous product and **(A.3)**, we can conclude that the alphabet of $\mathbf{G}_H^{p'}(j)$ is:

$$\Sigma_{\mathbf{G}_H^{p'}(j)} = \cup_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} \Sigma_{I_k} \cup \Sigma_{IH} = \Sigma_{IH}$$

Part I.b complete.

By **Parts I.a** and **I.b**, we can conclude **(A.2)** \Leftrightarrow **(A.3)**, as required.

(II) For **Point II**, and $j \in \{1, \dots, n\}$, we must show:

$$\mathbf{(A.1)} \quad (\forall s \in \mathcal{L}^p(j) \cap \mathcal{S}_L(j) \cap \mathcal{I}(j)) \text{Elig}_{\mathcal{L}^p(j)}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_L(j) \cap \mathcal{I}(j)}(s) \quad \Leftrightarrow \quad \mathbf{(A.6)}$$

$$\mathbf{(Defn. 5)} \quad (\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j) \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s) \quad \mathbf{(A.7)}$$

We start by converting **(A.6)** into a more useful form.

From *Proposition 30* of [28], we can conclude:

$$\mathcal{L}^p(j) = P_j(\mathcal{L}_j^p), \quad \mathcal{S}_L(j) = P_j(\mathcal{S}_{L_j}), \quad \mathcal{I}(j) = P_j(\mathcal{I}_j)$$

Substituting into **(A.6)**, we find that **(A.6)** is equivalent to:

$$(\forall s \in P_j(\mathcal{L}_j^p) \cap P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)) \text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \cap \Sigma_u \subseteq \text{Elig}_{P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)}(s) \quad (\mathbf{A.8})$$

It is thus sufficient to show **(A.8)** \Leftrightarrow **(A.7)**.

We first note that, by *Propositions 1* and *5*, languages \mathcal{L}_j^p , \mathcal{S}_{L_j} , and \mathcal{I}_j are P_j -invariant. **(A.9)**

$$\Rightarrow \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j \text{ and } \mathcal{S}_{L_j} \cap \mathcal{I}_j \text{ are } P_j\text{-invariant, by } \textit{Proposition 2(c)}. \quad (\mathbf{A.10})$$

By **(A.9)** and *Proposition 2(b)*, we can also conclude:

$$P_j(\mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j) = P_j(\mathcal{L}_j^p) \cap P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j) \text{ and } P_j(\mathcal{S}_{L_j} \cap \mathcal{I}_j) = P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j) \quad (\mathbf{A.11})$$

We next note that **(A.7)** and **(A.8)** are almost in the correct form to apply *Proposition 3*. The only problem is that Σ_u is not necessarily a subset of $\Sigma'(j)$.

Claim 1:

$$(\forall s \in P_j(\mathcal{L}_j^p) \cap P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)) \quad (\dagger)$$

$$\text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \cap \Sigma_u \subseteq \text{Elig}_{P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)}(s) \Leftrightarrow \text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)}(s)$$

Let $s \in P_j(\mathcal{L}_j^p) \cap P_j(\mathcal{S}_{L_j}) \cap P_j(\mathcal{I}_j)$

We first note that $P_j(\mathcal{L}_j^p) \subseteq \Sigma'(j)$, by definition of P_j .

$$\Rightarrow \text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \subseteq \Sigma'(j), \text{ by definition of the Elig operator.}$$

$$\Rightarrow \text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \cap \Sigma'(j) = \text{Elig}_{P_j(\mathcal{L}_j^p)}(s)$$

The result follows immediately, thus (*dagger*) holds. **Claim 1** complete.

Claim 2:

$$(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j) \quad (\ddagger)$$

$$\text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s) \Leftrightarrow \text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$$

Let $s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j$ **(A.12)**

$$\mathbf{(2.a)} \quad \text{Show } \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s) \Rightarrow \text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$$

This is automatic as $\text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u$

(2.b) Show $\text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s) \Rightarrow \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$

Assume $\text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$ **(A.13)**

Let $\sigma \in \text{Elig}_{\mathcal{L}_j^p}(s) \cap \Sigma_u$. **(A.14)**

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

Sufficient to show that $s\sigma \in \mathcal{S}_{L_j} \cap \mathcal{I}_j$, by definition of Elig operator.

From **(A.14)**, we have two possibilities: $\sigma \in \Sigma'(j) \cap \Sigma_u$ or $\sigma \in \Sigma_u - \Sigma'(j)$

For case $\sigma \in \Sigma'(j) \cap \Sigma_u$, the results follow immediately from **(A.13)**.

For case $\sigma \in \Sigma_u - \Sigma'(j)$, we start by noting that this implies that $P_j(\sigma) = \epsilon$, by definition of P_j **(A.15)**

We next note that $s \in \mathcal{S}_{L_j} \cap \mathcal{I}_j$ (by **(A.12)**) implies that:

$$P_j(s) \in P_j(\mathcal{S}_{L_j} \cap \mathcal{I}_j)$$

$\Rightarrow P_j(s)P_j(\sigma) \in P_j(\mathcal{S}_{L_j} \cap \mathcal{I}_j)$, by **(A.15)**.

$\Rightarrow P_j(s\sigma) \in P_j(\mathcal{S}_{L_j} \cap \mathcal{I}_j)$, as P_j is catenative.

$\Rightarrow s\sigma \in \mathcal{S}_{L_j} \cap \mathcal{I}_j$, by **(A.10)** and *Proposition 2(d)*.

Part 2.b complete.

By **Parts 2.a** and **2.b**, we can conclude that (\ddagger) holds. **Claim 2** complete.

By **(A.11)**, **Claims 1** and **2**, we see that to prove **(A.8)** \Leftrightarrow **(A.7)**, it is sufficient to prove:

$$(\forall s \in P_j(\mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)) \text{Elig}_{P_j(\mathcal{L}_j^p)}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{P_j(\mathcal{S}_{L_j} \cap \mathcal{I}_j)}(s) \quad \Leftrightarrow \quad \textbf{(A.16)}$$

$$(\forall s \in \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j) \text{Elig}_{\mathcal{L}_j^p}(s) \cap (\Sigma_u \cap \Sigma'(j)) \subseteq \text{Elig}_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s) \quad \textbf{(A.17)}$$

We can now take $\Sigma_a = \Sigma'(j)$, $\Sigma_b = \Sigma_u \cap \Sigma'(j)$, $P = P_j$, $L_1 = \mathcal{L}_j^p \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j$, $L_2 = \mathcal{L}_j^p$, $L_3 = \mathcal{S}_{L_j} \cap \mathcal{I}_j$, and conclude by **(A.9)**, **(A.10)**, and *Proposition 3* that:

$$\textbf{(A.16)} \Leftrightarrow \textbf{(A.17)}$$

Part II complete.

(III) For **Point III**, we must show:

$$(A.1) \quad (\forall s \in \mathcal{H}^{p'}(j) \cap \mathcal{I}(j) \cap \mathcal{S}'_H(j)) \text{Elig}_{\mathcal{H}^{p'}(j) \cap \mathcal{I}(j)}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}'_H(j)}(s) \quad \Leftrightarrow \quad (A.18)$$

$$(Defn. 5) \quad (\forall s \in \mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k] \cap \mathcal{S}_H) \text{Elig}_{\mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k]}(s) \cap \Sigma_u \subseteq \text{Elig}_{\mathcal{S}_H}(s) \quad (A.19)$$

We start by converting **(A.18)** into a more useful form.

From *Proposition 30* of [28], we can conclude:

$$\mathcal{H}^{p'}(j) = P_j(\mathcal{H}^p) \cap [\cap_{k \in \{1, \dots, (j-1), (j+1), \dots, n\}} P_j(\mathcal{I}_k)], \quad \mathcal{S}'_H(j) = P_j(\mathcal{S}_H), \quad \mathcal{I}(j) = P_j(\mathcal{I}_j)$$

Substituting into **(A.18)**, we find that **(A.18)** is equivalent to:

$$\begin{aligned} & (\forall s \in P_j(\mathcal{H}^p) \cap [\cap_{k \in \{1, \dots, n\}} P_j(\mathcal{I}_k)] \cap P_j(\mathcal{S}_H)) \\ & \quad \text{Elig}_{P_j(\mathcal{H}^p) \cap [\cap_{k \in \{1, \dots, n\}} P_j(\mathcal{I}_k)]}(s) \cap \Sigma_u \subseteq \text{Elig}_{P_j(\mathcal{S}_H)}(s) \end{aligned} \quad (A.20)$$

It is thus sufficient to show **(A.20)** \Leftrightarrow **(A.19)**.

We first note that, by *Propositions 1* and *5*, languages \mathcal{H}^p , \mathcal{S}_H , and \mathcal{I}_k ($k = 1, \dots, n$) are P_j -invariant. (A.21)

$$\Rightarrow \mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k] \cap \mathcal{S}_H \text{ and } \mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k] \text{ are } P_j\text{-invariant (Proposition 2(c)).} \quad (A.22)$$

By **(A.21)** and *Proposition 2(b)*, we can also conclude:

$$\begin{aligned} P_j(\mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k] \cap \mathcal{S}_H) &= P_j(\mathcal{H}^p) \cap [\cap_{k \in \{1, \dots, n\}} P_j(\mathcal{I}_k)] \cap P_j(\mathcal{S}_H) \text{ and } P_j(\mathcal{H}^p \cap [\cap_{k \in \{1, \dots, n\}} \mathcal{I}_k]) \\ &= P_j(\mathcal{H}^p) \cap [\cap_{k \in \{1, \dots, n\}} P_j(\mathcal{I}_k)] \end{aligned} \quad (A.23)$$

The remainder of the proof is identical to **Part II**, after suitable relabelling.

Part III complete. ■

We thus conclude by **Points I, II**, and **III**, that Definition 5 is equivalent to Definition 15, as required.

REFERENCES

- [1] N. Alsop, "Formal techniques for the procedural control of industrial processes," Ph.D. dissertation, Department of Chemical Engineering and Chemical Technology, Imperial College of Science, Technology and Medicine, London, 1996.
- [2] R. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.

- [3] M. Queiroz and J. Cury, "Modular supervisory control of large scale discrete event systems," in *Proceedings of WODES 2000*, Ghent, Belgium, Aug 2000, pp. 103–110.
- [4] G. Stremersch and R. Boel, "Decomposition of the supervisory control problem for Petri nets under preservation of maximal permissiveness," *IEEE Trans. Automatic Control*, vol. 46, no. 9, pp. 1490–1496, 2001.
- [5] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*, Department of Electrical and Computer Engineering, University of Toronto, July 2004, Monograph and TCT software can be downloaded at <http://www.control.toronto.edu/DES/>.
- [6] G. Barrett and S. Lafortune, "Decentralized supervisory control with communicating controllers," *IEEE Trans. Automatic Control*, vol. 45, no. 9, pp. 1620–1638, 2000.
- [7] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observations," in *Proc. 27th IEEE Conf. Decision Contr.*, Dec 1988, pp. 1125–1130.
- [8] K. Rudie and J. C. Willems, "The computational complexity of decentralized discrete-event control problems," *IEEE Trans. Automatic Control*, vol. 44, no. 7, pp. 1313–1319, 1995.
- [9] K. Rudie and W. M. Wonham, "Think globally, act locally: decentralized supervisory control," *IEEE Trans. on Automatic Control*, vol. 37, no. 11, pp. 1692–1708, Nov 1992, reprinted in F.A. Sadjadi (Ed.), *Selected Papers on Sensor and Data Fusion*, 1996; ISBN 0-8194-2265-7.
- [10] T. Yoo and S. Lafortune, "A general architecture for decentralized supervisory control of discrete-event systems," in *Proc. of WODES 2000*, Ghent, Belgium, Aug 2000, pp. 111–118.
- [11] S. Chen, "Control of discrete-event systems of vector and mixed structural type," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.
- [12] Y. Li and W. Wonham, "Control of vector discrete-event systems: I - The base model," *IEEE Trans. Automatic Control*, vol. 38, no. 8, pp. 1214–1227, August 1993.
- [13] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems using Petri Nets*. Kluwer Academic Publishers, 1998.
- [14] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.
- [15] P. Caines and Y. Wei, "The hierarchical lattices of a finite machine," *Systems Control Letters*, vol. 25, pp. 257–263, July 1995.
- [16] H. Chen and H.-M. Hanisch, "Model aggregation for hierarchical control synthesis of discrete event systems," in *Proc. 39th Conf. Decision Contr.*, Sydney, Australia, December 2000, pp. 418–423.
- [17] G. Shen and P. E. Caines, "Hierarchically accelerated dynamic programming for finite-state machines," *IEEE Trans. Automatic Control*, vol. 47, no. 2, pp. 271–283, 2002.
- [18] K. Wong, "Discrete-event control architecture: An algebraic approach," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1994.
- [19] W. Wu, H. Su, J. Chu, and H. Zhai, "Hierarchical control of DES based on colored Petri nets," in *Proc. of IEEE Systems, Man, and Cybernetics*, vol. 3, 2001, pp. 1571–1576.
- [20] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Trans. on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, Oct 1990.
- [21] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," *IEEE Trans. on Automatic Control*, vol. 38, no. 12, pp. 1803–1819, Dec 1993.

- [22] E. W. Endsley, M. R. Lucas, and D. M. Tilbury, "Modular design and verification of logic control for reconfigurable machining systems," Oct. 2000, [ONLINE]. Available: <http://www-personal.engin.umich.edu/~tilbury/papers.html>.
- [23] P. Gohari-Moghadam, "A linguistic framework for controlled hierarchical DES," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1998.
- [24] H. Liu, J. Park, and R. Miller, "On hybrid synthesis for hierarchical structured Petri nets," Department of Computer Science, University of Maryland, College Park, MD, Tech. Rep., 1996.
- [25] B. Wang, "Top-down design for RW supervisory control theory," Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1995.
- [26] R. Leduc, B. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part I: Serial case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1322–1335, Sept. 2005.
- [27] R. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part II: Parallel case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1336–1348, Sept. 2005.
- [28] R. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2002, [ONLINE] Available: <http://www.cas.mcmaster.ca/~leduc>.
- [29] R. Leduc, B. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part I: Serial case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1322–1335, Sept. 2005, see also SQRL Report No. 12, Dept. of Computing and Software, McMaster University, Hamilton, ON. [ONLINE] http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.
- [30] R. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control, part II: Parallel case," *IEEE Trans. Automatic Control*, vol. 50, no. 9, pp. 1336–1348, Sept. 2005, see also SQRL Report No. 13, Dept. of Computing and Software, McMaster University, Hamilton, ON. [ONLINE] http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.
- [31] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 3, pp. 259–66, Mar. 1985.
- [32] P. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control Optim*, vol. 25, no. 1, pp. 206–230, 1987.
- [33] W. M. Wonham and P. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optim*, vol. 25, no. 3, pp. 637–659, 1987.
- [34] P. Dai, "Synthesis method for hierarchical interface-based supervisory control," Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, in preparation.
- [35] B. Brandin and F. 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*, Troy, Oct 1994, pp. 319–324.
- [36] F. Charbonnier, "Commande par supervision des systèmes à événements discrets: application à un site expérimental l'Atelier Inter-établissement de Productique," Laboratoire d'Automatique de Grenoble, Grenoble, France, Tech. Rep., 1994.
- [37] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.
- [38] Z. Zhang, "Smart TCT: an efficient algorithm for supervisory control design." Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2001.
- [39] C. Ma and W. M. Wonham, "Control of state tree structures," in *Proc. 11th Mediterranean Conference on Control and Automation*, June 2003, paper T4-005 (6pp.).

- [40] E. W. Endsley and D. M. Tilbury, "Modular verification of modular finite state machines," in *Proc. 43th Conf. Decision Contr.*, vol. 1, Atlantis, Paradise Island, Bahamas, December 2004, pp. 972–979.
- [41] Y. Willner and M. Heymann, "Supervisory control of concurrent discrete-event systems," *International Journal of Control*, vol. 54, no. 5, pp. 1143–1169, 1991.
- [42] S. Abdelwahed, "Interacting discrete event systems: Modeling, verification and supervisory control," Ph.D. dissertation, Dept. of Elec. and Comp. Eng., University of Toronto, 2002.
- [43] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. of the IFAC World Congress on Automatic Control*, Barcelona, Spain, 2002.
- [44] S. Bogdan, F. L. Lewis, Z. Kovačić, A. Gürel, and M. Štajdohar, "An implementation of the matrix-based supervisory controller of flexible manufacturing systems," *IEEE Trans. on Control Systems Technology*, vol. 10, no. 5, pp. 709–716, Sept. 2002.
- [45] J. Burch, E. M. Clarke, and K. McMillan, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, pp. 142–170, 1992.