# Symbolic Synthesis and Verification of Hierarchical Interface-based Supervisory Control

Raoguang Song and Ryan J. Leduc

Dept. of Computing and Software, McMaster University
email: songr,leduc@mcmaster.ca

***Abstract*— Hierarchical Interface-based Supervisory Control (HISC) decomposes a discrete-event system (DES) into a high-level subsystem which communicates with $n \geq 1$ low-level subsystems, through separate interfaces which restrict the interaction of the subsystems. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability. The current HISC verification and synthesis algorithms are based upon explicit state and transition listings which limit the size of a given level to about $10^7$ states when 1GB of memory is used.**

**In this paper, we extend the HISC approach by introducing a set of predicate based fixed point operators that allow us to do a per level synthesis to construct for each level a maximally permissive supervisor that satisfies the corresponding HISC conditions. We prove that these fixpoint operators compute the required level-wise supremal languages. We then present algorithms that implement the fixpoint operators. Based on these algorithms, a symbolic implementation is briefly discussed which can be implemented using Binary Decision Diagrams.**

**We also discuss a method to implement our synthesized supervisors in a more compact manner.**

**A large manufacturing system example (worst case state space on the order of $10^{30}$) extended from the AIP example is briefly discussed. The example showed that we can now handle a given level with a statespace as large as $10^{15}$ states, using less than 160MB of memory. This represents a significant improvement in the size of systems that can be handled by the HISC approach. A software tool for synthesis and verification of HISC systems using our approach was also developed.**

## I. Introduction

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

The Hierarchical Interface-based Supervisory Control(HISC) framework was proposed by Leduc *et al.* in [7]–[11] to alleviate the state explosion problem. The HISC approach decomposes a system into a *high-level subsystem* which communicates with $n \geq 1$ parallel *low-level subsystems* through separate interfaces that restrict the interaction of the subsystems. This structure permits the derivation of 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 [4], Dai and Leduc introduced a HISC based synthesis method that replaced each level's supervisor with a corresponding specification DES, and then does a per level synthesis to construct for each level a maximally permissive supervisor that satisfies the corresponding HISC conditions. However, both the HISC synthesis and verification method are based upon explicit state and transition listings which limit the size of a given level to about $10^7$ states when 1GB of memory is used.

As each subsystem in HISC is typically modeled as a group of plant DES and a group of specification/supervisor DES, each subsystem-wide state can be represented as a vector, where each element of the vector is the state of a component DES. Therefore, we can use Binary Decision Diagrams (BDD: [2], [6]) to represent the state space and transitions for each subsystem, and develop algorithms based on BDD representations to verify or synthesize supervisors for our HISC system. As we will see, by using BDD representation and algorithms we will be able to handle much larger subsystems, allowing HISC to be applied to even larger systems.

Using IDD (Integer Decision Diagram, an extension of BDD) to verify and synthesize flat DES have previously been investigated by Zhang et al [18], while Vahidi et al [15] have investigated applying BDD to flat systems.

Ma et al [12], [13] presented a top-down multi-level design model called State Tree Structures (STS), which was initiated in [16] by using the idea of state charts [5]. Ma used BDD based algorithms that allowed him to model and synthesize a state-based supervisor for a system with an estimated state-space on the order of $10^{24}$.

In this paper, we first discuss DES and predicate preliminaries, and then introduce the HISC approach in Sections III-IV. As the HISC method has already been explained and justified in detail in [10], [11], and [8], we will only discuss it briefly here. For a small illustrative HISC example, please see [10].

For the remainder of the paper we will be presenting our new results from [14], beginning with the predicate based fixed point operators we developed. They allow us to do a per level synthesis to construct for each level a maximally permissive supervisor that satisfies the corresponding HISC

conditions. We prove that these fixpoint operators compute the required level-wise supremal languages.

We then present algorithms that implement the fixpoint operators. We also briefly discuss a predicate based HISC verification method derived from the synthesis method. Based on these algorithms, a symbolic implementation is briefly discussed which can be implemented using Binary Decision Diagrams.

We next discuss a method to implement our synthesized supervisors in a more compact manner. Finally, we discuss a large manufacturing example (estimated worst case state-space on the order of $10^{30}$) extended from the AIP example in [7], [8]. The example demonstrates the improved scalability that our symbolic approach offers.

## II. Preliminaries

Supervisory control theory 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 [17]. Below, we present a summary of the terminology that we use in this paper.

### A. DES

Let $\Sigma$ be a finite set of distinct symbols (*events*), and $\Sigma^*$ be the set of all finite sequences of events, including $\epsilon$, the *empty string.* Let $L \subseteq \Sigma^*$ be a *language* over $\Sigma$. 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 Pwr($\Sigma$) denote the power set of $\Sigma$ (i.e. all possible subsets of $\Sigma$).

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

The reachable state subset of DES $\mathbf{G}$, denoted $Y_r$, is: $Y_r := \{y \in Y |\ (\exists s \in \Sigma^*)\, \delta(y_o, s) = y\}$. A DES $\mathbf{G}$ is *reachable* if $Y_r = Y$. The coreachable state subset, denoted by $Y_{cr}$, is $Y_{cr} := \{y \in Y |\ (\exists s \in \Sigma^*)\, \delta(y, s) \in Y_m\}$. A DES is *coreachable* if $Y_{cr} = Y$. We say the DES is *trim* if it is both reachable and coreachable. We will always assume that a DES has a finite state and event set and is deterministic.

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

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

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

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

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

The synchronous product of DES $\mathbf{G}_1 = (Y_1, \Sigma_1, \delta_1, y_{o_1}, Y_{m_1})$ and $\mathbf{G}_2 = (Y_2, \Sigma_2, \delta_2, y_{o_2}, Y_{m_2})$, denoted $\mathbf{G}_1||\mathbf{G}_2$, is defined to be a reachable DES $\mathbf{G}$ with and event set $\Sigma = \Sigma_1 \cup \Sigma_2$ and properties:

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

A DES $\mathbf{G}$ is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$. This is equivalent to saying that every reachable state is also coreachable. We say that DES $\mathbf{G}$ represents a language $K \subseteq \Sigma^*$ if $\mathbf{G}$ is nonblocking and $L_m(\mathbf{G}) = K$. We thus have $L(\mathbf{G}) = \overline{K}$.

For $\mathbf{G}_1$ and $\mathbf{G}_2$, if $\Sigma_1 = \Sigma_2$ we can define the *product* of the two DES as:

$$\mathbf{G_1} \times \mathbf{G_2} := (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_{1_0}, q_{2_0}), Q_{1_m} \times Q_{2_m}),$$

where $\delta_1 \times \delta_2 : Q_1 \times Q_2 \times \Sigma \to Q_1 \times Q_2$ is given by $(\delta_1 \times \delta_2)((q_1, q_2), \sigma) := (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$, whenever $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$.

### B. Predicates and Predicate Transformers

We only give a brief introduction here. Please refer to [14] for more details.

Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be a DES. A *predicate* $P$ defined on $Q$ is a function $P : Q \to \{1, 0\}$. $P$ is identified by a corresponding state subset $Q_P := \{q \in Q | P(q) = 1\} \subseteq Q$. If $q \in Q_P$, we write $q \models P$ and say $q$ *satisfies* $P$. We write $Pred(Q)$ for the set of all predicates defined on $Q$. For predicates $P_1$ and $P_2$, we define $P_1 - P_2 = P_1 \wedge \neg P_2$.

The two special predicates *true* and *false* are identified by $Q$ and $\emptyset$, respectively. The predicate $P_m$ is identified by $Q_m$. If $Q$ is understood, for $Q_1 \subseteq Q$, denote the predicate identified by $Q_1$ as $pr(Q_1)$.

A *predicate transformer* is a function $f : Pred(Q) \to Pred(Q)$. We now introduce several predicate transformers for $P \in Pred(Q)$ which will be used later on. See [14] for more formal definitions.

- The *reachability predicate* $R(\mathbf{G}, P)$ is defined to hold precisely on those states that can be reached in $\mathbf{G}$ from $q_0$ via states satisfying $P$.
- The *coreachability predicate* $CR(\mathbf{G}, P)$ is defined to hold precisely on those states that can reach a marker state in $\mathbf{G}$ via states satisfying $P$.
- With $\mathbf{G}$ and $\Sigma' \subseteq \Sigma$ fixed, $TR(\mathbf{G}, P, \Sigma')$ is defined to hold precisely on those states that can reach a state satisfying $P$ in $\mathbf{G}$ only via transitions with events in $\Sigma'$.
- With DES $\mathbf{G}$, $P' \in Pred(Q)$ and $\Sigma' \subseteq \Sigma$ fixed, $\mathcal{CR}(\mathbf{G}, P', \Sigma', P)$ is defined to hold precisely on those states that can reach a state satisfying $P'$ in $\mathbf{G}$ via states satisfying $P$ and transitions with events in $\Sigma'$.

Finally, for $P \in Pred(Q)$ we define $L(\mathbf{G}, P)$ to be the closed language *induced* by $P$ as $L(\mathbf{G}, P) := \{w \in \Sigma^* | (\forall v \leq w)\delta(q_0, v) \models P\}$. Similarly, we define $L_m(\mathbf{G}, P)$ to be the marked language induced by $P$ as $L_m(\mathbf{G}, P) := \{w \in L(\mathbf{G}, P) | \delta(q_0, w) \in Q_m\}$.

## III. HISC Overview

An HISC system currently is a two-level system which includes one *high-level subsystem* and $n \geq 1$ *low-level subsystems*. The high-level subsystem communicates with each low-level subsystem through a separate *interface.* We will also refer to the high-level or a given low-level as a module.

In order to restrict the information flow at the interface, the system alphabet is partitioned into pairwise disjoint alphabets:

$$\Sigma := \Sigma_H \dot\cup \dot\bigcup_{k\in\{1,\ldots,n\}} [\Sigma_{L_k} \dot\cup \Sigma_{R_k} \dot\cup \Sigma_{A_k}] \tag{1}$$

The events in $\Sigma_H$ are *high-level events* and the events in $\Sigma_L$ *low-level events* as these events appear only in the high-level and low-level subsystems, respectively. As the *interface* is only concerned with communication between the two subsystems, it only contains 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 $\Sigma_I$. The events in $\Sigma_R$, called *request events*, represent commands sent from the high-level subsystem to the low-level subsystem. The events in $\Sigma_A$ are *answer events* and represent the low-level subsystem's responses to the request events.

In the remainder of this paper, $j$ is always an index with range $\{1, \ldots, n\}$. The high-level subsystem is modeled by DES $\mathbf{G}_H$, which is the product of *the high-level plant* $\mathbf{G}^p_H$ and *the high-level supervisor* $\mathbf{S}_H$ (both are defined over event set $\Sigma_H \dot\cup (\dot\cup_{k\in\{1,\ldots,n\}}[\Sigma_{R_k} \dot\cup \Sigma_{A_k}])$). The $j^{th}$ low-level subsystem is modeled by DES $\mathbf{G}_{L_j}$, which is the product of *the $j^{th}$ low-level plant* $\mathbf{G}^p_{L_j}$ and *the $j^{th}$ low-level supervisor* $\mathbf{S}_{L_j}$ (both are defined over event set $\Sigma_{L_j} \dot\cup \Sigma_{R_j} \dot\cup \Sigma_{A_j}$), and the $j^{th}$ interface is modeled by $\mathbf{G}_{I_j}$(defined over event set $\Sigma_{R_j} \dot\cup \Sigma_{A_j}$). The overall system structure is shown in Figure 1.

For controllability, the event set $\Sigma$ is also partitioned as $\Sigma = \Sigma_c \dot\cup \Sigma_u$, where $\Sigma_c$ is the controllable event set and $\Sigma_u$ is the uncontrollable event set.

We refer to DES $\boldsymbol{\mathcal{G}}_H := \mathbf{G}_H \parallel \mathbf{G}_{I_1} \parallel \ldots \parallel \mathbf{G}_{I_n}$ as the *high-level* and DES $\boldsymbol{\mathcal{G}}_{L_j} := \mathbf{G}_{I_j} \parallel \mathbf{G}_{L_j}$ as the *$j^{th}$ low-level*. For convenience, the following event sets are also defined:

$$\begin{aligned}
\Sigma_{I_j} &:= \Sigma_{R_j} \dot\cup \Sigma_{A_j} & \Sigma_A &:= \dot\cup_{k\in\{1,\ldots,n\}} \Sigma_{A_k} \\
\Sigma_{IH} &:= (\dot\cup_{k\in\{1,\ldots,n\}} \Sigma_{I_k}) \dot\cup \Sigma_H & \Sigma_{IL_j} &:= \Sigma_{L_j} \dot\cup \Sigma_{I_j} \\
\Sigma_{IL} &:= \dot\cup_{k\in\{1,\ldots,n\}} \Sigma_{IL_k} & \Sigma_{hu} &:= \Sigma_{IH} \cap \Sigma_u \\
\Sigma_{hc} &:= \Sigma_{IH} \cap \Sigma_c & \Sigma_{lu_j} &:= \Sigma_{IL_j} \cap \Sigma_u \\
\Sigma_{lc_j} &:= \Sigma_{IL_j} \cap \Sigma_c
\end{aligned}$$

The interface DES ensures that communication occurs between levels 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. To enforce such a mechanism, the interface DES is required to be a *command-pair interface* as defined below.

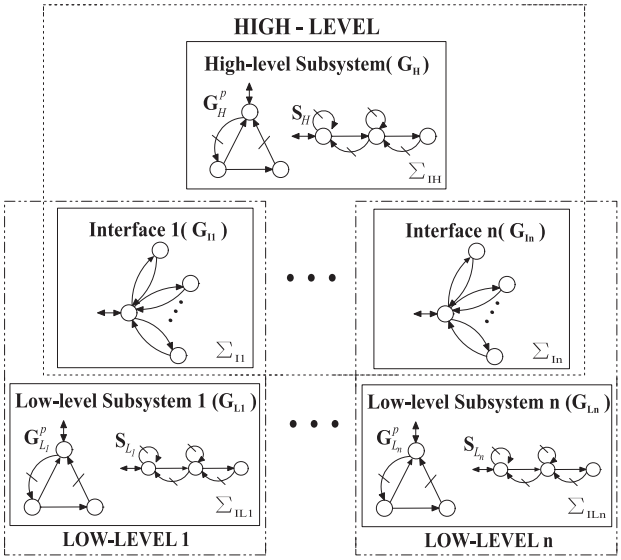


Fig. 1. HISC system structure

*Definition 1: For the $n^{th}$ degree interface system composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, the $j^{th}$ interface DES $\mathbf{G}_{I_j} = (X_j, \Sigma_{R_j} \dot\cup \Sigma_{A_j}, \xi_j, x_{0_j}, X_{j_m})$ is a* command-pair interface *if:*

(A) $L(\mathbf{G}_{I_j}) \subseteq \overline{(\Sigma_{R_j}.\Sigma_{A_j})^*}$

(B) $L_m(\mathbf{G}_{I_j}) = (\Sigma_{R_j}.\Sigma_{A_j})^* \cap L(\mathbf{G}_{I_j})$

An HISC system is actually a structured *flat system*. The *flat plant* is defined as $\mathbf{PLANT} := \mathbf{G}^p_H \parallel \mathbf{G}^p_{L_1} \parallel \cdots \parallel \mathbf{G}^p_{L_n}$ and the *flat supervisor* is defined as $\mathbf{SUP} := \mathbf{S}_H \parallel \mathbf{S}_{L_1} \parallel \cdots \parallel \mathbf{S}_{L_n} \parallel \mathbf{G}_{I_1} \parallel \cdots \parallel \mathbf{G}_{I_n}$.

Therefore, both **PLANT** and **SUP** are defined over event set $\Sigma$. The whole flat system is the product of the flat supervisor and the flat plant: $\mathbf{SYSTEM} := \mathbf{SUP} \times \mathbf{PLANT}$

We want to ensure **SYSTEM** satisfies:

1) $\underline{L(\mathbf{SYSTEM})}\Sigma_u \cap L(\mathbf{PLANT}) \subseteq L(\mathbf{SYSTEM})$
2) $\overline{L_m(\mathbf{SYSTEM})} = L(\mathbf{SYSTEM})$

## IV. Local Conditions

We now present a set of local properties that will allow us to verify the global properties of controllability and nonblocking. The conditions here are based on [8] and [9]. Please refer to them for a more detailed discussion.

To make our discussion of synthesis simpler later, we wish to express our high-level and low-level as product DES. To do this, we need to extend[1] the event sets of the interfaces.

*Definition 2: For the $n^{th}$ degree interface system that respects the alphabet partition given by (1) and is composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, for*

[1]The **selfloop** operator here is as defined in TCT [17] and simply adds selfloop transitions for each symbol in the indicated event set, at every state in the DES.

*all* $j \in \{1,\ldots,n\}$*, define:* $\mathbf{G}^h_{I_j} := \mathbf{selfloop}(\mathbf{G}_{I_j}, \Sigma_{IH} - \Sigma_{I_j})$, $\mathbf{G}^l_{I_j} := \mathbf{selfloop}(\mathbf{G}_{I_j}, \Sigma_{L_j})$ *and* $\mathbf{G}^h_I := \mathbf{G}^h_{I_1} \times \cdots \times \mathbf{G}^h_{I_n}$.

We now present the properties that the system must satisfy to ensure that it interacts with the interfaces correctly. This definition is a bit different from the one presented in [8], but equivalent as we will show. We use it as it makes our proofs easier.

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

- *Multi-level Properties*
  1) *The event set of* $\mathbf{G}^p_H$ *and* $\mathbf{S}_H$ *is* $\Sigma_{IH}$*, and the event set of* $\mathbf{G}^p_{L_j}$ *and* $\mathbf{S}_{L_j}$ *is* $\Sigma_{IL_j}$.
  2) $\mathbf{G}_{I_j}$ *is a command-pair interface.*
- *High-level Properties*
  3. $L(\boldsymbol{\mathcal{G}}_H)\Sigma_{A_j} \cap L(\mathbf{G}^h_{I_j}) \subseteq L(\boldsymbol{\mathcal{G}}_H)$
- *Low-level Properties*
  4. $L(\boldsymbol{\mathcal{G}}_{L_j})\Sigma_{R_j} \cap L(\mathbf{G}^l_{I_j}) \subseteq L(\boldsymbol{\mathcal{G}}_{L_j})$
  5. $(\forall s \in L(\boldsymbol{\mathcal{G}}_{L_j}))(\forall \rho \in \Sigma_{R_j})(\forall \alpha \in \Sigma_{A_j})$ $s\rho\alpha \in L(\mathbf{G}^l_{I_j}) \Rightarrow (\exists l \in \Sigma^*_{L_j})\ s\rho l\alpha \in L(\boldsymbol{\mathcal{G}}_{L_j})$
  6. $(\forall s \in L(\boldsymbol{\mathcal{G}}_{L_j}))\ \ s \in L_m(\mathbf{G}^l_{I_j}) \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in L_m(\boldsymbol{\mathcal{G}}_{L_j})$

*Proposition 1:* Definition 3 is equivalent to the *interface consistency* definition from [8].

*Proof:* See proof in [14]. ■

The next definition ensures that each module is locally nonblocking.

*Definition 4: The* $n^{th}$ *degree interface system composed of plant components* $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$*, supervisors* $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$*, and interfaces* $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$ *is said to be* level-wise nonblocking *with respect to the alphabet partition given by (1), if the following two conditions are satisfied:*

1) $\overline{L_m(\boldsymbol{\mathcal{G}}_H)} = L(\boldsymbol{\mathcal{G}}_H)$
2) $(\forall j \in \{1,\ldots,n\})\ \overline{L_m(\boldsymbol{\mathcal{G}}_{L_j})} = L(\boldsymbol{\mathcal{G}}_{L_j})$

*Theorem 1 (From [8]):* If the $n^{th}$ degree interface system composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is level-wise nonblocking and interface consistent with respect to the alphabet partition given by (1), then

$$\overline{L_m(\mathbf{SYSTEM})} = L(\mathbf{SYSTEM})$$

We next give the controllability requirements for each module.

*Definition 5: The* $n^{th}$ *degree interface system composed of plant components* $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$*, supervisors* $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$*, and interfaces* $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$ *is said to be* level-wise controllable *with respect to the alphabet partition given by (1), if for all* $j \in \{1,\ldots,n\}$ *the following two conditions are satisfied:*

1) *The alphabet of* $\mathbf{G}^p_H$ *and* $\mathbf{S}_H$ *is* $\Sigma_{IH}$*, the alphabet of* $\mathbf{G}^p_{L_j}$ *and* $\mathbf{S}_{L_j}$ *is* $\Sigma_{IL_j}$*, and the alphabet of* $\mathbf{G}_{I_j}$ *is* $\Sigma_{I_j}$
2) $L(\boldsymbol{\mathcal{G}}_{L_j})\Sigma_{lu_j} \cap L(\mathbf{G}^p_{L_j}) \subseteq L(\boldsymbol{\mathcal{G}}_{L_j})$
3) $L(\boldsymbol{\mathcal{G}}_H)\Sigma_{hu} \cap L(\mathbf{G}^p_H \times \mathbf{G}^h_I) \subseteq L(\boldsymbol{\mathcal{G}}_H)$

*Theorem 2 (From [8]):* If the $n^{th}$ degree interface system composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, supervisors $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, is level-wise controllable with respect to the alphabet partition given by (1), then

$$L(\mathbf{SYSTEM})\Sigma_u \cap L(\mathbf{PLANT}) \subseteq L(\mathbf{SYSTEM})$$

## V. Synthesis of HISC

In [7], [8], [10], [11], the supervisors in an HISC system were designed by hand, and then the system was checked to see if it was interface consistent, level-wise controllable and nonblocking. If not, it was modified by the designer until it was. However, for a complicated system it is very desirable to synthesize the supervisors from specifications. In this section, we will discuss how to construct supervisors whose marked languages are supremal controllable sublanguages of a set of per module specifications that satisfies the HISC conditions for the given high or low-level. We say such supervisors are *locally maximally permissible* for their level. We will then give predicate based algorithms. The algorithms can easily be implemented by using BDD.

In the previous section, we specified that a $n^{th}$ degree interface system is composed of plants, supervisors and interfaces. For synthesis, we will assume that our interface system is composed of plants, specifications and interfaces. Essentially, we will replace the high-level supervisor $\mathbf{S}_H$ by a specification DES $\mathbf{E}_H$ (defined over $\Sigma_{IH}$), and for all $j \in \{1,\ldots,n\}$, we will replace the $j^{th}$ low-level supervisor $\mathbf{S}_{L_j}$ by a specification DES $\mathbf{E}_{L_j}$ (defined over $\Sigma_{IL_j}$). We will refer to such a system as a $n^{th}$ *degree specification interface system*, and call the original system with supervisors a $n^{th}$ *degree supervisor interface system.* We will refer to a system as a $n^{th}$ *degree interface system* when we do not wish to make a distinction.

For clarity, we now interpret some definitions used in the previous section in terms of a $n^{th}$ *degree specification interface system.*

$$\mathbf{G}_H := \mathbf{E}_H \times \mathbf{G}^p_H \qquad \mathbf{G}_{L_j} := \mathbf{E}_{L_j} \times \mathbf{G}^p_{L_j}$$
$$\boldsymbol{\mathcal{G}}_H := \mathbf{E}_H \times \mathbf{G}^p_H \times \mathbf{G}^h_{I_j} \qquad \boldsymbol{\mathcal{G}}_{L_j} := \mathbf{E}_{L_j} \times \mathbf{G}^p_{L_j} \times \mathbf{G}^l_{I_j}$$

We now give the starting point for the synthesis process. These are conditions the system must meet as a minimum, and correspond to parts of the interface consistency and level-wise controllability definitions that can not be corrected (if the system fails to satisfy these conditions) in the synthesis process that we will present.

*Definition 6: The* $n^{th}$ *degree interface specification system composed of plant components* $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$*, specifications* $\mathbf{E}_H, \mathbf{E}_{L_1}, \ldots, \mathbf{E}_{L_n}$*, and interfaces* $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$*, is* HISC-valid *with respects to the alphabet partition given by (1), if for all* $j \in \{1,\ldots,n\}$*, the following conditions are satisfied:*

1) *The alphabet of* $\mathbf{G}^p_H$ *and* $\mathbf{E}_H$ *is* $\Sigma_{IH}$.
2) *The alphabet of* $\mathbf{G}^p_{L_j}$ *and* $\mathbf{E}_{L_j}$ *is* $\Sigma_{IL_j}$.

*3)* $\mathbf{G}_{I_j}$ *is a command-pair interface.*

Let $\mathbf{\Phi}$ be a $n^{th}$ degree HISC-valid specification interface system that respects the alphabet partition given by (1) and is composed of plant components $\mathbf{G}^p_H, \mathbf{G}^p_{L_1}, \ldots, \mathbf{G}^p_{L_n}$, specifications $\mathbf{E}_H, \mathbf{E}_{L_1}, \ldots, \mathbf{E}_{L_n}$, and interfaces $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$.

As the predicate algorithms will operate on the states of the DES, we give the tuple definitions for the following DES for later reference:

$$\begin{aligned}
\boldsymbol{\mathcal{G}}_H &:= (Q_H, \Sigma_{IH}, \delta_H, q_{H_0}, Q_{H_m}) \\
\mathbf{G}^p_H &:= (Y_H, \Sigma_{IH}, \eta_H, y_{H_0}, Y_{H_m}) \\
\mathbf{E}_H &:= (Z_H, \Sigma_{IH}, \zeta_H, z_{H_0}, Z_{H_m}) \\
\mathbf{G}^h_{I_j} &:= (X^h_j, \Sigma_{IH}, \xi^h_j, x^h_{j_0}, X^h_{j_m}) \\
\mathbf{G}^h_I &:= (X^h, \Sigma_{IH}, \xi^h, x^h_0, X^h_m) \\
\boldsymbol{\mathcal{G}}_{L_j} &:= (Q_{L_j}, \Sigma_{IL_j}, \delta_{L_j}, q_{Lj_0}, Q_{Lj_m}) \\
\mathbf{G}^p_{L_j} &:= (Y_{L_j}, \Sigma_{IL_j}, \eta_{L_j}, y_{Lj_0}, Y_{Lj_m}) \\
\mathbf{E}_{L_j} &:= (Z_{L_j}, \Sigma_{IL_j}, \zeta_{L_j}, z_{Lj_0}, Z_{Lj_m}) \\
\mathbf{G}^l_{I_j} &:= (X^l_j, \Sigma_{IL_j}, \xi^l_j, x^l_{j_0}, X^l_{j_m}).
\end{aligned}$$

So,
$$\begin{aligned}
\boldsymbol{\mathcal{G}}_H &= (Z_H \times Y_H \times X^h, \Sigma_{IH}, \zeta_H \times \eta_H \times \\
&\quad \xi^h, (z_{H_0}, y_{H_0}, x^h_0), Z_{H_m} \times Y_{H_m} \times X^h_m) \\
\boldsymbol{\mathcal{G}}_{L_j} &= (Z_{L_j} \times Y_{L_j} \times X^l_j, \Sigma_{IL_j}, \zeta_{L_j} \times \eta_{L_j} \times \\
&\quad \xi^l_j, (z_{Lj_0}, y_{Lj_0}, x^l_{j_0}), Z_{Lj_m} \times Y_{Lj_m} \times X^l_{j_m}) \\
\mathbf{G}^h_I &= (X^h_1 \times \cdots \times X^h_n, \Sigma_{IH}, \xi^h_1 \times \cdots \times \\
&\quad \xi^h_n, (x^h_{1_0}, \cdots, x^h_{n_0}), X^h_{1_m} \times \cdots \times X^h_{n_m})
\end{aligned}$$

### A. High-level Supervisor Synthesis

In this section, we show how to synthesize a locally maximally permissible high-level supervisor for the system $\mathbf{\Phi}$. Below are the properties the marked language of our high-level supervisor must satisfy for HISC. They correspond to point 3 of Defn. 3 and point 3 of Defn. 5.

*Definition 7: For system* $\mathbf{\Phi}$, *let* $K \subseteq \Sigma^*_{IH}$. $K$ *is* high-level interface controllable*(HIC)* with respect to $\mathbf{\Phi}$ *if for all* $j \in \{1, \ldots, n\}$,

*1)* $\overline{K}\Sigma_{hu} \cap L(\mathbf{G}^p_H \times \mathbf{G}^h_I) \subseteq \overline{K}$
*2)* $\overline{K}\Sigma_{A_j} \cap L(\mathbf{G}^h_{I_j}) \subseteq \overline{K}$

Clearly, the empty language $\emptyset$ is HIC with respect to $\mathbf{\Phi}$.

For an arbitrary language $E \subseteq \Sigma^*_{IH}$, we define the set of all sublanguages of $E$ that are HIC with respect to $\mathbf{\Phi}$ as

$$\mathcal{C}_H(E) := \{K \subseteq E | K \text{ is HIC with respect to } \mathbf{\Phi}.\}$$

*Proposition 2:* For system $\mathbf{\Phi}$, $\mathcal{C}_H(E)$ is nonempty and is closed under arbitrary unions. In particular, $\mathcal{C}_H(E)$ contains a (unique) supremal element, denoted $\sup\mathcal{C}_H(E)$.

*Proof:* See proof in [14]. ■

For system $\mathbf{\Phi}$, if we compute $\sup\mathcal{C}_H(L_m(\boldsymbol{\mathcal{G}}_H))$, then a DES representing this sublanguage is a locally maximally permissible high-level supervisor. As the supervisor would be nonblocking, our new system would thus also satisfy point 1 of Defn. 4.

To compute $\sup\mathcal{C}_H(L_m(\boldsymbol{\mathcal{G}}_H))$, we want to define a suitable fixpoint operator. For function $f : X \to X$ where $X$ is an arbitrary set, an element $x \in X$ is a *fixpoint* of $f$ if $x = f(x)$. We will also use the notation $f^k(x)$, $k \in \{0, 1, \ldots\}$, to mean $k$ applications of $f$ in a row with $f^0(x) := x$.

If we were defining a function that would operate on a language $K \in Pwr(\Sigma^*_{IH})$, we would want it to evaluate: $L_m(\boldsymbol{\mathcal{G}}_H) \cap \sup\mathcal{C}_H(\overline{K})$. However, we need a predicate based operator that we can apply to the states of our system.

Let $Pred(Q_H)$ be the set of all predicates on $Q_H$, the state set of $\boldsymbol{\mathcal{G}}_H$. For any $q \in Q_H$, as $\boldsymbol{\mathcal{G}}_H := \mathbf{E}_H \times \mathbf{G}^p_H \times \mathbf{G}^h_{I_j}$ there must exist unique $z \in Z_H, y \in Y_H$ and $x \in X^h$ such that $q = (z, y, x)$. For state $x \in X^h$, there must also exist unique $x_1 \in X^h_1, \ldots, x_n \in X^h_n$ such that $x = (x_1, \ldots, x_n)$. For $P \in Pred(Q_H)$, we now show how to compute $\sup\mathcal{C}_H(L(\boldsymbol{\mathcal{G}}_H, P))$.

*Definition 8: For system* $\mathbf{\Phi}$, *define the function* PHIC : $Pred(Q_H) \to Pred(Q_H)$ *for* $P \in Pred(Q_H)$ *as:*

$$\text{PHIC}(P) := \neg TR(\boldsymbol{\mathcal{G}}_H, \neg P \vee pr(\text{Bad}_H), \Sigma_{hu} \cup \Sigma_A),$$

*where* $\text{Bad}_H := \{q \in Q_H | \ ((\exists \sigma_u \in \Sigma_{hu})\ (\eta_H \times \xi^h((y, x), \sigma_u)! \ \&\ \zeta_H(z, \sigma_u)\ \not!))$ *or* $((\exists j \in \{1, \ldots, n\})(\exists \sigma_{a_j} \in \Sigma_{A_j})\ (\xi^h_j(x_j, \sigma_{a_j})! \ \&\ \zeta_H \times \eta_H \times \xi^h_1 \times \cdots \times \xi^h_{j-1} \times \xi^h_{j+1} \times \cdots \times \xi^h_n((z, y, x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n), \sigma_{a_j})\ \not!))\}$

We show in [14] for arbitrary $P \in Pred(Q_H)$, that $\sup\mathcal{C}_H(L(\boldsymbol{\mathcal{G}}_H, P)) = L(\boldsymbol{\mathcal{G}}_H, \text{PHIC}(P))$.

We now provide a predicate based method to compute $L_m(\boldsymbol{\mathcal{G}}_H) \cap L(\boldsymbol{\mathcal{G}}_H, \text{PHIC}(P))$.

*Proposition 3:* For system $\mathbf{\Phi}$, the following holds:
$(\forall P \in Pred(Q_H))$
$L_m(\boldsymbol{\mathcal{G}}_H) \cap L(\boldsymbol{\mathcal{G}}_H, P) = L_m(\boldsymbol{\mathcal{G}}_H, CR(\boldsymbol{\mathcal{G}}_H, P))$

*Proof:* See proof in [14]. ■

**The Algorithm to Compute** $\sup\mathcal{C}_H(L_m(\boldsymbol{\mathcal{G}}_H))$

We now put everything together for the high-level and construct our algorithm.

*Definition 9: For system* $\mathbf{\Phi}$, *define the function* $\Gamma_H$ : $Pred(Q_H) \to Pred(Q_H)$ *according to*

$$(\forall P \in Pred(Q_H))\ \ \Gamma_H(P) := CR(\boldsymbol{\mathcal{G}}_H, \text{PHIC}(P))$$

*Theorem 3:* For system $\mathbf{\Phi}$, the following two points hold:

1) There exists $k \in \{0, 1, 2, \ldots\}$ such that $k \leq |Q_H|$ and $\Gamma^k_H(true)$ is the greatest fixpoint of the function $\Gamma_H$.
2) $\sup\mathcal{C}_H(L_m(\boldsymbol{\mathcal{G}}_H)) = L_m(\boldsymbol{\mathcal{G}}_H, \Gamma^k_H(true))$.

*Proof:* See proof in [14]. ■

We can thus take our high-level supervisor to be $\mathbf{S}_H$ defined over event set $\Sigma_{IH}$, with $L(\mathbf{S}_H) = L(\boldsymbol{\mathcal{G}}_H, \Gamma^k_H(true))$ and $L_m(\mathbf{S}_H) = L_m(\boldsymbol{\mathcal{G}}_H, \Gamma^k_H(true))$, where $k$ is from Theorem 3. The supervisor $\mathbf{S}_H$ can be built by removing states from $\boldsymbol{\mathcal{G}}_H$ that do not satisfy $\Gamma^k_H(true)$. It can then be made trim by removing states that do not satisfy $R(\boldsymbol{\mathcal{G}}_H, \Gamma^k_H(true))$. The algorithm to compute $\Gamma^k_H(true)$ is given below.

### B. Low-level Supervisor Synthesis

In this section, we show how to synthesize a locally maximally permissible $j^{th}$ low-level supervisor for the system $\mathbf{\Phi}$. We first only discuss part of the conditions (ones similar to the high-level conditions) that the marked language of

our $j^{th}$ low-level supervisor must satisfy for HISC. They correspond to point 4 of Defn. 3 and point 2 of Defn. 5.

*Definition 10: Let $K \subseteq \Sigma^*_{IL_j}$. $K$ is $j^{th}$* low-level P4 interface controllable ($\mathrm{LPC}_j$) *with respect to $\mathbf{\Phi}$ if the following conditions are satisfied:*

*1)* $\overline{K}\Sigma_{lu_j} \cap L(\mathbf{G}^p_{L_j}) \subseteq \overline{K}$

*2)* $\overline{K}\Sigma_{R_j} \cap L(\mathbf{G}^l_{I_j}) \subseteq \overline{K}$

Clearly, the empty language $\emptyset$ is $\mathrm{LPC}_j$ with respect to $\mathbf{\Phi}$.

**Algorithm 1** Computing the greatest fixpoint of $\Gamma_H$

1: $P_{bad_H} \leftarrow pr(\mathrm{Bad}_H)$;
2: $P_1 \leftarrow true$;
3: **repeat**
4: $\quad P_2 \leftarrow P_1$;
5: $\quad P_1 \leftarrow CR(\boldsymbol{\mathcal{G}}_H, \neg TR(\boldsymbol{\mathcal{G}}_H, \neg P_2 \vee P_{bad_H}, \Sigma_{hu} \cup \Sigma_A))$;
6: **until** $P_1 = P_2$
7: return $P_1$;

For an arbitrary language $E \subseteq \Sigma^*_{IL_j}$, we define the set of all sublanguages of $E$ that are $\mathrm{LPC}_j$ with respect to $\mathbf{\Phi}$ as

$$\mathcal{LPC}_j(E) := \{K \subseteq E \mid K \text{ is } \mathrm{LPC}_j \text{ with respect to } \mathbf{\Phi}.\}$$

*Proposition 4:* For system $\mathbf{\Phi}$, $\mathcal{LPC}_j(E)$ is nonempty and is closed under arbitrary unions. In particular, $\mathcal{LPC}_j(E)$ contains a (unique) supremal element, denoted $\sup \mathcal{LPC}_j(E)$.

*Proof:* See proof in [14]. ∎

Let $Pred(Q_{L_j})$ be the set of all predicates on $Q_{L_j}$, the state set of $\boldsymbol{\mathcal{G}}_{L_j}$. For any $q \in Q_{L_j}$, as $\boldsymbol{\mathcal{G}}_{L_j} = \mathbf{S}_{L_j} \times \mathbf{G}_{L_j} \times \mathbf{G}^l_{I_j}$, there must exist $z \in Z_{L_j}, y \in Y_{L_j}$ and $x \in X^l_j$ such that $q = (z, y, x)$.

*Definition 11: For system $\mathbf{\Phi}$, define the function* $\mathrm{PLPC}_j$ : $Pred(Q_{L_j}) \rightarrow Pred(Q_{L_j})$ *for $P \in Pred(Q_{L_j})$ as:*

$$\mathrm{PLPC}_j(P) := \neg TR(\boldsymbol{\mathcal{G}}_{L_j}, \neg P \vee pr(\mathrm{Bad}_{L_j}), \Sigma_{lu_j} \cup \Sigma_{R_j}),$$

*where* $\mathrm{Bad}_{L_j} = \{q \in Q_{L_j} | ((\exists \sigma_u \in \Sigma_{lu_j})\ \eta_{L_j}(y, \sigma_u)! \,\&\, \zeta_{L_j} \times \xi^l_j((z,x), \sigma_u)\ \not!) \text{ or } ((\exists \sigma_{r_j} \in \Sigma_{R_j})\ \xi^l_j(x, \sigma_{r_j})! \,\&\, \zeta_{L_j} \times \eta_{L_j}((z,y), \sigma_{r_j})\ \not!)\}$

We show in [14] for arbitrary $P \in Pred(Q_{L_j})$, that $\sup \mathcal{LPC}_j(L(\boldsymbol{\mathcal{G}}_{L_j}, P)) = L(\boldsymbol{\mathcal{G}}_{L_j}, \mathrm{PLPC}_j(P))$.

We now add the remaining low-level conditions, namely points 5 and 6 of Defn. 3.

*Definition 12: Let $K \subseteq \Sigma^*_{IL_j}$. $K$ is $j^{th}$* low-level interface controllable ($\mathrm{LIC}_j$) *with respect to $\mathbf{\Phi}$ if the following conditions are satisfied:*

*1) $K$ is $j^{th}$ low-level P4 interface controllable.*

*2)* $(\forall s \in \overline{K})(\forall \rho \in \Sigma_{R_j})(\forall \alpha \in \Sigma_{A_j})$
$s\rho\alpha \in L(\mathbf{G}^l_{I_j}) \Rightarrow (\exists l \in \Sigma^*_{L_j})\ s\rho l\alpha \in \overline{K}$

*3)* $(\forall s \in \overline{K})\ \ s \in L_m(\mathbf{G}^l_{I_j}) \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in K$

Clearly, the empty language $\emptyset$ is $\mathrm{LIC}_j$ with respect to $\mathbf{\Phi}$.

For an arbitrary language $E \subseteq \Sigma^*_{IL_j}$, we define the set of all sublanguages of $E$ that are $\mathrm{LIC}_j$ with respect to $\mathbf{\Phi}$ as

$$\mathcal{C}_{L_j}(E) := \{K \subseteq E | K \text{ is } \mathrm{LIC}_j \text{ with respect to } \mathbf{\Phi}\ \}$$

*Proposition 5:* For system $\mathbf{\Phi}$, $\mathcal{C}_{L_j}(E)$ is nonempty and is closed under arbitrary unions. In particular, $\mathcal{C}_{L_j}(E)$ contains a (unique) supremal element, denoted $\sup \mathcal{C}_{L_j}(E)$.

*Proof:* See proof in [14]. ∎

For system $\mathbf{\Phi}$, if we compute $\sup \mathcal{C}_{L_j}(L_m(\boldsymbol{\mathcal{G}}_{L_j}))$ then a DES representing this sublanguage is a locally maximally permissible $j^{th}$ low-level supervisor. As the supervisor would be nonblocking, our new system would thus satisfy point 2 of Defn. 4 for this value of $j$.

To compute $\sup \mathcal{C}_{L_j}(L_m(\boldsymbol{\mathcal{G}}_{L_j}))$, we need to define a suitable predicate based fixpoint operator. For a given predicate $P \in Pred(Q_{L_j})$, we already know how to compute $\sup \mathcal{LPC}_j(L(\boldsymbol{\mathcal{G}}_{L_j}, P))$. We now need to to develop operators to handle points 5-6 of Defn. 3. As we want to intersect the resulting language with $L_m(\boldsymbol{\mathcal{G}}_{L_j})$, we can achieve this in a similar manner to what was used in Proposition 3.

*Definition 13: For system $\mathbf{\Phi}$, define the function $\Gamma_{p5_j}$ : $Pred(Q_{L_j}) \rightarrow Pred(Q_{L_j})$ for $P \in Pred(Q_{L_j})$ as:*

$\Gamma_{p5_j}(P) := P - pr(\{q \in Q_{L_j} | (\exists \rho \in \Sigma_{R_j})(\exists \alpha \in \Sigma_{A_j})$
$\xi^l_j(x, \rho\alpha)! \ \&\ \delta_{L_j}(q, \rho) \models \neg\ \mathcal{CR}(\boldsymbol{\mathcal{G}}_{L_j}, P_\alpha, \Sigma_{L_j}, P)\})$,
*where* $P_\alpha := pr(\{q' \in Q_{L_j} | \delta_{L_j}(q', \alpha) \models P\})$.

Algorithm 2 shows how to compute $\Gamma_{p5_j}(P)$ for arbitrary $P \in Pred(Q_{L_j})$.

**Algorithm 2** $\Gamma_{p5_j}(P)$

1: $P_{bad5} \leftarrow false$;
2: **for** each $\alpha \in \Sigma_{A_j}$ **do**
3: $\quad P_\alpha \leftarrow pr(\{q' \in Q_{L_j} \mid \delta_{L_j}(q', \alpha) \models P\})$;
4: $\quad P_{\mathcal{CR}_\alpha} \leftarrow \mathcal{CR}(\boldsymbol{\mathcal{G}}_{L_j}, P_\alpha, \Sigma_{L_j}, P)$;
5: $\quad$ **for** each $\rho \in \Sigma_{R_j}$ **do**
6: $\quad\quad P_{bad5} \leftarrow P_{bad5} \vee pr(\{q \in Q_{L_j} | \xi^l_j(x, \rho\alpha)!$
$\quad\quad\ \&\delta_{L_j}(q, \rho) \models \neg P_{\mathcal{CR}_\alpha}\})$;
7: $\quad$ **end for**
8: **end for**
9: return $P - P_{bad5}$;

*Definition 14: For system $\mathbf{\Phi}$, define the function $\Gamma_{p6_j}$ : $Pred(Q_{L_j}) \rightarrow Pred(Q_{L_j})$ for $P \in Pred(Q_{L_j})$ as:*

$$\Gamma_{p6_j}(P) := P - (P_{X_{j_m}} - \mathcal{CR}(\boldsymbol{\mathcal{G}}_{L_j}, P_{L_{j_m}}, \Sigma_{L_j}, P)),$$

*where* $P_{X_{j_m}} := pr(\{q \in Q_{L_j} | x \in X^l_{j_m}\})$, $P_{L_{j_m}} := pr(Q_{L_{j_m}})$, *$q = (z, y, x)$ and $X^l_{j_m}$ and $Q_{L_{j_m}}$ are the marker state sets of $\mathbf{G}^l_{I_j}$ and $\boldsymbol{\mathcal{G}}_{L_j}$ respectively.*

**The Algorithm to Compute** $\sup \mathcal{C}_{L_j}(L_m(\boldsymbol{\mathcal{G}}_{L_j}))$

We now put everything together for the $j^{th}$ Low-level and construct our algorithm.

*Definition 15: For system $\mathbf{\Phi}$, define the function $\Gamma_{L_j}$ : $Pred(Q_{L_j}) \rightarrow Pred(Q_{L_j})$ for $P \in Pred(Q_{L_j})$ as:*

$$\Gamma_{L_j}(P) := CR(\boldsymbol{\mathcal{G}}_{L_j}, \Gamma_{p6_j}(\Gamma_{p5_j}(\mathrm{PLPC}_j(P))))$$

*Theorem 4:* For system $\mathbf{\Phi}$, the following two points hold:

1) There exists $k \in \{0, 1, 2, \ldots\}$ such that $k \leq |Q_{L_j}|$ and $\Gamma^k_{L_j}(true)$ is the greatest fixpoint of the function $\Gamma_{L_j}$.
2) $\sup \mathcal{C}_{L_j}(L_m(\boldsymbol{\mathcal{G}}_{L_j})) = L_m(\boldsymbol{\mathcal{G}}_{L_j}, \Gamma^k_{L_j}(true))$

*Proof:* See proof in [14]. ∎

We can thus take our $j^{th}$ low-level supervisor to be $\mathbf{S}_{L_j}$ defined over event set $\Sigma_{IL_j}$, with $L_m(\mathbf{S}_{L_j}) = L_m(\boldsymbol{\mathcal{G}}_{L_j}, \Gamma^k_{L_j}(true))$ and $L(\mathbf{S}_{L_j}) = L(\boldsymbol{\mathcal{G}}_{L_j}, \Gamma^k_{L_j}(true))$,

where $k$ is from Theorem 4. The supervisor $\mathbf{S}_{L_j}$ can be constructed by removing states from $\boldsymbol{\mathcal{G}}_{L_j}$ that do not satisfy $\Gamma^k_{L_j}(true)$. It can then be made trim by removing states that do not satisfy $R(\boldsymbol{\mathcal{G}}_{L_j}, \Gamma^k_{L_j}(true))$. The algorithm to compute $\Gamma^k_{L_j}(true)$ is given below.

**Algorithm 3** Computing the greatest fixpoint of $\Gamma_{L_j}$

1: $P_{bad_{L_j}} \leftarrow pr(\mathrm{Bad}_{L_j})$;
2: $P_1 \leftarrow true$;
3: **repeat**
4: $\quad P_2 \leftarrow P_1$;
5: $\quad P_1 \leftarrow \neg TR(\boldsymbol{\mathcal{G}}_{L_j}, \neg P_1 \vee P_{bad_{L_j}}, \Sigma_{lu_j} \cup \Sigma_{R_j})$;
6: $\quad P_1 \leftarrow \Gamma_{p5_j}(P_1)$;
7: $\quad P_1 \leftarrow P_1 - (P_{X_{j_m}} - \mathcal{CR}(\boldsymbol{\mathcal{G}}_{L_j}, P_{L_{j_m}}, \Sigma_{L_j}, P_1))$;
8: $\quad P_1 \leftarrow CR(\boldsymbol{\mathcal{G}}_{L_j}, P_1)$;
9: **until** $P_1 = P_2$
10: return $P_1$;

Line 5 computes $\mathrm{PLPC}_j(P_1)$. Line 7 computes $\Gamma_{p6_j}(P_1)$. Line 8 calculates the coreachable states under $P_1$.

## VI. Verification of HISC

We have also developed a method to verify a $n^{th}$ degree supervisor interface system, based on the synthesis algorithms we have presented. The method treats all the supervisors as their corresponding specifications and then applies the synthesis algorithm to the system (assuming that it is HISC-valid). If there are no *reachable* states that must be removed from $\boldsymbol{\mathcal{G}}_H$, or $\boldsymbol{\mathcal{G}}_{L_j}$ ($j \in \{1, \ldots, n\}$) after the first pass, then the system is interface consistent, level-wise nonblocking, and level-wise controllable. We note that typically the verification process is faster and uses less memory than synthesis, meaning that we can usually verify larger systems than we can apply synthesis to. We refer the reader to [14] for the algorithm details.

## VII. Symbolic Computation for HISC Synthesis and Verification

The efficiency of our HISC synthesis and verification is dominated by the computation of the four predicate transformers: $R, CR, TR$ and $\mathcal{CR}$. We have developed a method to use logic formulas to represent state subsets and transitions in a system, and then used these formulas to compute the predicate transformers discussed above as well as other miscellaneous conditions needed to verify/synthesize an HISC system. We have also developed a method of using Reduced Ordered Binary Decision Diagram [2], [6] to implement the above logic formula based algorithms. The BDD software package we used is *BuDDy 2.4* developed by Jørn Lind-Nielsen. To achieve this, we drew heavily on the work of Ma [13]. Please refer to [14] for details.

## VIII. Controller Implementation

For system $\boldsymbol{\Phi}$ defined in Section V, we showed that we could synthesize locally maximally permissible supervisors for each level, namely $\mathbf{S}_H$ and $\mathbf{S}_{L_j}$ ($j \in \{1, \ldots, n\}$). However, these automata-based supervisors could easily be very large ($\mathbf{S}_H$, in the AIP example in the next section, has a state space on the order of $10^{15}$), making them difficult to implement as controllers directly. We now briefly discuss an alternate implementation method that will typically be more practical.

For the system $\boldsymbol{\Phi}$, let $P_H$ be the resulting predicate from Algorithm 1, and $P_{L_j}$ be the resulting predicate from Algorithm 3. Let $Q$ be the statespace of the synchronous product of all the DES in system $\boldsymbol{\Phi}$. This means that a state $q \in Q$ can be represented as a tuple

$$q := (z_H, y_H, z_{L_1}, \ldots, z_{L_n}, y_{L_1}, \ldots, y_{L_n}, x_1, \ldots, x_n). \quad (2)$$

From the synthesis algorithms, we know that $\mathbf{S}_H$ can be obtained by trimming off states that do not satisfy $P_H$ from the high-level $\boldsymbol{\mathcal{G}}_H$, and $\mathbf{S}_{L_j}$ can be obtained by trimming off states that do not satisfy $P_{L_j}$ from the $j^{th}$ low-level $\boldsymbol{\mathcal{G}}_{L_j}$. We show in [14] that we can express the appropriate control action for each state $q \in Q$ as a per event predicate local to a particular level.

*Definition 16: For each* $\sigma \in \Sigma_c \cap (\Sigma_H \cup \Sigma_{R_1} \cup \cdots \cup \Sigma_{R_n})$, *define the predicate* $f_{H\sigma} \in Pred(Q_H)$ *for each* $q \in Q$ *as*

$$f_{H\sigma}(z_H, y_H, x_1, \ldots, x_n) := \begin{cases} 1, & \delta_H((z_H, y_H, x_1, \ldots, x_n), \sigma) \models P_H \\ 0, & \textit{otherwise} \end{cases} \quad (3)$$

*For each* $j \in \{1, \ldots, n\}$, $\sigma \in \Sigma_c \cap (\Sigma_{L_j} \cup \Sigma_{A_j})$, *define the predicate* $f_{L_{j_\sigma}} \in Pred(Q_{L_j})$ *for each* $q \in Q$ *as*

$$f_{L_j\sigma}(z_{L_j}, y_{L_j}, x_j) := \begin{cases} 1, & \delta_{L_j}((z_{L_j}, y_{L_j}, x_j), \sigma) \models P_{L_j} \\ 0, & \textit{otherwise} \end{cases} \quad (4)$$

For instance, for $q \in Q$ and $\sigma \in \Sigma_c \cap (\Sigma_{L_j} \cup \Sigma_{A_j})$, if $f_{L_j\sigma}(z_{L_j}, y_{L_j}, x_j) = 1$, then $\sigma$ should be enabled at state $q$. Each predicate can be represented as a BDD, and typically the BDD is much smaller than the corresponding automata supervisors. To obtain the state information $q$, we could have an observer for each component of system $\boldsymbol{\Phi}$ (ie. for $\mathbf{G}^p_H$, $\mathbf{E}_H$ etc.). As each component is typically the synchronous product of other DES, the size of the observer for each DES needed is likely to be quite small. For examples, see the AIP example in [14].

Figure 2 shows the structure of our implementation, with $k_H, k_1, \ldots k_n \in \{0, 1, \ldots\}$. The top box represents our observers which provide the state information for our predicates. The enablement information is then sent to the plant.

## IX. The AIP Example

To demonstrate the utility of our method, we applied it to a large manufacturing system, the Atelier Inter-établissement de Productique (AIP) as described in [1], [3]. The AIP system includes a central loop (CL) conveyor, four external loop (EL) conveyors, 4 transport units (TU) (each moves pallets between CL and a specific EL), an assembly station (AS) at EL1, 2 and 3, and an Input/Output (I/O) station at EL4 (allows pallets to enter/leave system). We will only briefly introduce this example. Please see [14] for complete details.

In [7], [8], Leduc et al modelled the AIP as an HISC system. Using their algorithms based upon explicit state and

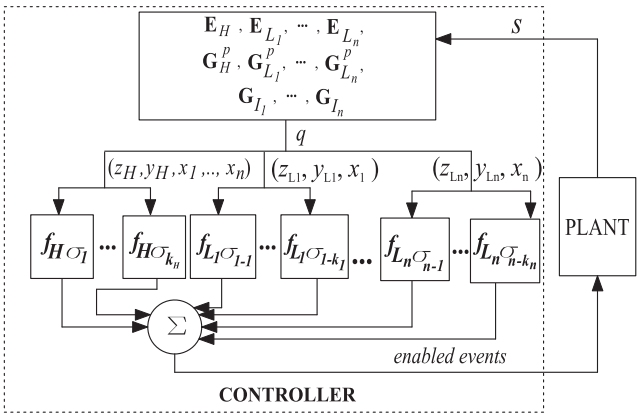


Fig. 2. Control diagram

transition listings, it took 254.7 seconds and 659MB to verify that the high level, with $3.3 \times 10^6$ states, satisfied the HISC conditions. Using our BDD based algorithms, it took us 2 seconds and an estimated 30MB.

We then extended the AIP example of [7], [8] by modelling how pallets move around the system. For example, a pallet can't reach an assembly station until it is transported from the central loop to the section of the external loop leading to the station. We also enforced capacity restrictions on each loop section as follows: maximum four pallets at a time in a given section of the CL, and five pallets at a time for a section of an EL. This was not originally modelled by Leduc et al as it made the high level too large for their software to handle.

To verify the system, we needed to add an additional supervisor that restricted the number of pallets in the system (excluding EL4) to 15, to prevent the system from blocking. The system was verified on a 2.8 GHz Pentium 4 CPU, with 512MB memory, running Fedora Core 2. It used less than 160MB of RAM, took 25.7 minutes to verify that the high-level HISC conditions were satisfied and less than 1 second to verify that the low-level HISC conditions were satisfied for each low-level. The reachable state space for the high-level was $5.16 \times 10^{13}$, and the total estimated worst case reachable statespace size was $7.04 \times 10^{28}$. A flat verification with our BDD tool quickly used up all available RAM, and had failed to complete after 24 hours.

We then removed the "15 pallets in system" supervisor, and performed a HISC synthesis. Our BDD tool used less than 160MB of RAM, took 128 minutes to synthesize a high-level supervisor, and less than 1 second to synthesize each low-level supervisor. The reachable state space for the high-level was $1.14 \times 10^{15}$, and the total estimated worst case reachable state space was $1.51 \times 10^{30}$. This is a clear improvement over previous HISC algorithms from [4], [10], [11].

## X. Conclusions

In this paper, we have developed a predicate based synthesis and verification method for systems modelled using Hierarchical Interface-based Supervisory Control. Combined with symbolic methods implemented using binary decision diagrams, we are now able to handle HISC systems with individual levels significantly larger than methods based upon explicit state and transition listings. In the AIP example investigated, we saw an increase of eight orders of magnitude. This allows us to handle much larger systems.

## References


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

[2] R. E. Bryant. Graph-Based algorithm for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, Aug. 1986.

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

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

[5] D. Harel. A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, Jun. 1987.

[6] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Nov. 1999.

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

[8] R.J. Leduc, M. Lawford, and P. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. Technical Report No. 32, SQRL, Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada, Dec. 2005. [ONLINE] Available: http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.

[9] R.J. Leduc, W.M. Wonham, and M. Lawford. Hierarchical interface-based supervisory control: Bi-level systems. Technical report No. 0103, Systems Control Group, University of Toronto, Toronto, ON, Canada, Nov. 2001.

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

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

[12] C. Ma and W. Murray Wonham. Control of state tree structures. In *Proc. 11th Mediterranean Conference on Control and Automation*, June 2003. Paper T4-005 (6pp.).

[13] Chuan Ma. *Nonblocking Supervisory Control of State Tree Structures*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., 2004.

[14] Raoguang Song. Symbolic synthesis and verification of hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2006. [ONLINE] http://www.cas.mcmaster.ca/~leduc/#studtheses.

[15] Arash Vahidi, Bengt Lennartson, and Martin Fabian. Efficient analysis of large discrete-event systems with binary decision diagrams. In *Proc. of the 44th IEEE Conference on Decision and Control and European Control Conference 2005*, pages 2751–2756, Seville, Spain, 2005.

[16] Bing Wang. Top-down design for RW supervisory control theory. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1995.

[17] W. M. Wonham. *Supervisory Control of Discrete-Event Systems*. Dept. of Electrical and Computer Engineering, University of Toronto, Jul. 2005. Monograph and TCT software can be downloaded at http://www.control.toronto.edu/DES/.

[18] Z.H. Zhang and W. Murray Wonham. STCT: an efficient algorithm for supervisory control design. In *Proc. of SCODES 2001*, pages 82–93, INRIA, Paris, July 2001.