Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2009-01

Aggregative Synthesis of Distributed Supervisors based on Automaton Abstraction

Rong Su, Jan H. van Schuppen and Jacobus E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2009-01
Eindhoven, January 2009
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

Achieving nonblockingness in supervisory control imposes a major challenge when the number of states of a target system is large, often owing to synchronous product of many relatively small local components. To overcome this difficulty, in this paper we first present a distributed supervisory control problem, then provide an aggregative synthesis approach that computes nonblocking distributed supervisors. The key to the success of this approach is a newly developed automaton abstraction technique, that removes irrelevant internal transitions at each synthesis stage so that nonblocking supervisor synthesis can be carried out on relatively small abstracted models.

1 Introduction

Since the automaton-based Ramadge-Wonham (RW) supervisory control paradigm first appeared in the control literature in 1982, which was subsequently summarized in [1] [2], there has been a large volume of literature on it. One of the main challenges of RW supervisor synthesis is to achieve nonblockingness when a target system has a large number of states, often resulted from synchronous product of many relatively small local components. To overcome this computational difficulty, many approaches have been proposed recently. For example, in [4] the authors introduce the concept of *interface invariance* in their hierarchical interface-based supervisory control approach. A very large nonblocking control problem may be solved, e.g. the system size reaches 10^{21} in the AIP example [4]. Nevertheless, designing an interface that can remain invariant during synthesis is rather difficult, which requires lots of experience and domain knowledge of the target system. In [5] a supervisor synthesis approach for state feedback control is proposed based on the concept of *state tree structures*. It has been shown in [5] that a system with 10^{24} states can be well handled. Nevertheless, this approach is essentially a centralized approach. Besides, it does not consider partial observation.

Recently attentions have been paid to modular/distributed supervisory control mainly for two reasons: potentially low synthesis complexity and high implementation flexibility, although modular/distributed control may result in less permissiveness than centralized control can achieve, e.g. [6] [8] [21] [19] [15] [20]. In this paper we first present a distributed supervisory control problem then we propose an aggregative synthesis approach to compute a supremal nonblocking state-normal supervisor of a nondeterministic distributed plant model under deterministic specifications. The key to the effectiveness of this approach is an automaton abstraction technique proposed in [10]. Such an abstraction technique does not have the drawback possessed by observers [3] used in [23] [6] [21] [7] and [19], where the alphabet of the codomain of a natural projection cannot be chosen arbitrarily - for the sake of obtaining the observer property, which in many cases results in the size of an abstraction not being small enough for subsequent supervisor synthesis. It is also different from automaton abstraction techniques proposed in [8] [22] [14] [15] and [20], where [8] requires an abstracted model weakly bisimilar to the original model, and [22] [20] are aimed for conflict equivalence, [14] for supervision equivalence and [15] for synthesis equivalence. All of these approaches require to use silent events in order to preserve appropriate equivalence relations. In our abstraction technique, no silent event is required and the construction is much simpler than using rewriting rules as used in [22] [14] [15] and [20].

We make two contributions in this paper. First, we present an algorithm to compute supremal nonblocking state-normal supervisors defined in [10]. Second, we propose an aggregative synthesis approach to compute a deterministic nonblocking distributed supervisor for a distributed system, where local components are nondeterministic and local specifications are deterministic. The algorithm for the supremal nonblocking state-normal supervisor is utilized at each stage of aggregative synthesis to compute an appropriate local supervisor, where the relevant local plant model is obtained by the proposed automaton abstraction technique. Although the idea of aggregation has been used in, e.g. [25] [21] [15] [20], their abstraction techniques are different from ours.

This paper is organized as follows. In Section II we first review relevant concepts and operations proposed in [10], then put forward a distributed supervisory control problem. After that, we present an approach for aggregative synthesis of distributed supervisors based on abstractions of nondeterministic automata in Section III. As an illustration, the proposed synthesis approach is applied to a cluster tool system in Section IV. Conclusions are stated in Section V. All long proofs are presented in the Appendix.

2 A Distributed Supervisor Synthesis Problem

In this section we first review basic concepts of languages and nondeterministic finite-state automata. Then we present a distributed supervisor synthesis problem.

2.1 Concepts of Languages and Nondeterministic Finite-State Automata

Let Σ be a finite alphabet, and Σ^* denote the Kleene closure of Σ, i.e. the collection of all finite sequences of events taken from Σ. Given two strings $s, t \in \Sigma^*$, s is called a *prefix substring* of t, written as $s \leq t$, if there exists $s' \in \Sigma^*$ such that $ss' = t$, where ss' denotes the concatenation of s and s'. We use ϵ to denote the empty string of Σ^* such that for any string $s \in \Sigma^*$, $\epsilon s = s\epsilon = s$. A subset $L \subseteq \Sigma^*$ is called a *language*. $\overline{L} = \{s \in \Sigma^* | (\exists t \in L)\, s \leq t\} \subseteq \Sigma^*$ is called the *prefix closure* of L. L is called *prefix closed* if $L = \overline{L}$. Given two languages $L, L' \subseteq \Sigma^*$, $LL' := \{ss' \in \Sigma^* | s \in L \wedge s' \in L'\}$.

Let $\Sigma' \subseteq \Sigma$. A mapping $P : \Sigma^* \to \Sigma'^*$ is called the *natural projection* with respect to (Σ, Σ'), if

1. $P(\epsilon) = \epsilon$
2. $(\forall \sigma \in \Sigma)\, P(\sigma) := \begin{cases} \sigma & \text{if } \sigma \in \Sigma' \\ \epsilon & \text{otherwise} \end{cases}$
3. $(\forall s\sigma \in \Sigma^*)\, P(s\sigma) = P(s)P(\sigma)$

Given a language $L \subseteq \Sigma^*$, $P(L) := \{P(s) \in \Sigma'^* | s \in L\}$. The inverse image mapping of P is

$$P^{-1} : 2^{\Sigma'^*} \to 2^{\Sigma^*} : L \mapsto P^{-1}(L) := \{s \in \Sigma^* | P(s) \in L\}$$

Given $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, the *synchronous product* of L_1 and L_2 is defined as:

$$L_1 || L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2) = \{s \in (\Sigma_1 \cup \Sigma_2)^* | P_1(s) \in L_1 \wedge P_2(s) \in L_2\}$$

where $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ and $P_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_2^*$ are natural projections. Clearly, $||$ is commutative and associative. Next, we introduce automaton product and abstraction.

A *nondeterministic finite-state automaton* is a 5-tuple $G = (X, \Sigma, \xi, x_0, X_m)$, where X stands for the state set, Σ for the alphabet, $\xi : X \times \Sigma \to 2^X$ for the nondeterministic transition function, x_0 for the initial state and X_m for the marker state set. As usual, we extend the domain of ξ from $X \times \Sigma$ to $X \times \Sigma^*$. If for any $x \in X$ and $\sigma \in \Sigma$, $\xi(x, \sigma)$ contains no more than one element, then G is called *deterministic*. Let

$$B(G) := \{s \in \Sigma^* | (\exists x \in \xi(x_0, s))(\forall s' \in \Sigma^*)\, \xi(x, s') \cap X_m = \varnothing\}$$

Any string $s \in B(G)$ can lead to a state x, from which no marker state is reachable, i.e. for any $s' \in \Sigma^*$, $\xi(x, s') \cap X_m = \varnothing$. We say G is *nonblocking* if $B(G) = \varnothing$. For each $x \in X$, we define another set

$$N_G(x) := \{s \in \Sigma^* | \xi(x, s) \cap X_m \neq \varnothing\}$$

and call $N_G(x_0)$ the *nonblocking set* of G, which is simply the set of all strings recognized by G. For the notation simplicity, we use $N(G)$ to denote $N_G(x_0)$. It is possible that $B(G) \cap \overline{N(G)} \neq \varnothing$, due to nondeterminism. Let $\phi(\Sigma)$ be the collection of all finite-state automata over Σ.

Given two nondeterministic automata $G_i = (X_i, \Sigma_i, \xi_i, x_{0,i}, X_{m,i}) \in \phi(\Sigma_i)$ $(i = 1, 2)$, the *product* of G_1 and G_2, written as $G_1 \times G_2$, is an automaton in $\phi(\Sigma_1 \cup \Sigma_2)$ such that

$$G_1 \times G_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \xi_1 \times \xi_2, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

where $\xi_1 \times \xi_2 : X_1 \times X_2 \times (\Sigma_1 \cup \Sigma_2) \to 2^{X_1 \times X_2}$ is defined as follows,

$$(\xi_1 \times \xi_2)((x_1, x_2), \sigma) := \begin{cases} \xi_1(x_1, \sigma) \times \{x_2\} & \text{if } \sigma \in \Sigma_1 - \Sigma_2 \\ \{x_1\} \times \xi_2(x_2, \sigma) & \text{if } \sigma \in \Sigma_2 - \Sigma_1 \\ \xi_1(x_1, \sigma) \times \xi_2(x_2, \sigma) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \end{cases}$$

Clearly, $\times$ is commutative and associative. $\xi_1 \times \xi_2$ is extended to $X_1 \times X_2 \times (\Sigma_1 \cup \Sigma_2)^* \to 2^{X_1 \times X_2}$. By a slight abuse of notations, from now on we use $G_1 \times G_2$ to denote its *reachable* part, which contains all states reachable from $(x_{1,0}, x_{2,0})$ by $\xi_1 \times \xi_2$ and transitions among these states. It is clear that $N(G_1 \times G_2) = N(G_1)||N(G_2)$. Next, we introduce automaton abstraction.

Definition 2.1. Given $G = (X, \Sigma, \xi, x_0, X_m)$, let $\Sigma' \subseteq \Sigma$ and $P : \Sigma^* \to \Sigma'^*$ be the natural projection. A *marking weak bisimulation* relation on X with respect to Σ' is an equivalence relation $R \subseteq \{(x, x') \in X \times X | x \in X_m \iff x' \in X_m\}$ such that,

$$(\forall (x, x') \in R)(\forall s \in \Sigma^*)(\forall y \in \xi(x, s))(\exists s' \in \Sigma^*)\, P(s) = P(s') \wedge (\exists y' \in \xi(x', s'))\, (y, y') \in R$$

The largest marking weak bisimulation relation on X with respect to Σ' is called *marking weak bisimilarity* on X with respect to Σ', written as $\approx_{\Sigma', G}$. □

Marking weak bisimulation relation is the same as weak bisimulation relation described in [17], except for the special treatment on marker states. From now on, when G is clear from the context, we simply use $\approx_{\Sigma'}$ to denote $\approx_{\Sigma', G}$. We now introduce abstraction.

Definition 2.2. Given $G = (X, \Sigma, \xi, x_0, X_m)$, let $\Sigma' \subseteq \Sigma$. The *automaton abstraction* of G with respect to the marking weak bisimulation $\approx_{\Sigma'}$ is an automaton $G/\approx_{\Sigma'} := (Y, \Sigma', \eta, y_0, Y_m)$ where

1. $Y := X/\approx_{\Sigma'} := \{< x > := \{x' \in X | (x, x') \in \approx_{\Sigma'}\} | x \in X\}$
2. $y_0 := < x_0 >$
3. $Y_m := \{y \in Y | y \cap X_m \neq \varnothing\}$
4. $\eta : Y \times \Sigma' \to 2^Y$, where for any $(y, \sigma) \in Y \times \Sigma'$,
$$\eta(y, \sigma) := \{y' \in Y | (\exists x \in y)(\exists u, u' \in (\Sigma - \Sigma')^*)\, \xi(x, u\sigma u') \cap y' \neq \varnothing\}$$

□

The time complexity of computing $G/\approx_{\Sigma'}$ is mainly resulted from computing $X/\approx_{\Sigma'}$, which can be done by using a state partition algorithm similar to the one presented in [27]. The complexity has been shown in [10] to be $O(\frac{1}{2}n(n-1)+mn^2\log n)$, where n is the number of states and m for the number of transitions in G. We now introduce a binary relation that will be used frequently later.

Definition 2.3. Given $G_i=(X_i,\Sigma_i,\xi_i,x_{i,0},X_{i,m})$ $(i=1,2)$, we say G_1 is *nonblocking preserving* with respect to G_2, denoted as $G_1 \sqsubseteq G_2$, if

1. $B(G_1)\subseteq B(G_2)$ and $N(G_1)=N(G_2)$

2. for any $s\in\overline{N(G_1)}$ and $x_1\in\xi_1(x_{1,0},s)$, there exists $x_2\in\xi_2(x_{2,0},s)$ such that,

$$N_{G_2}(x_2)\subseteq N_{G_1}(x_1)\wedge[x_1\in X_{1,m}\iff x_2\in X_{2,m}]$$

G_1 is *nonblocking equivalent* to G_2, denoted as $G_1\cong G_2$, if $G_1\sqsubseteq G_2$ and $G_2\sqsubseteq G_1$. □

Def. 2.3 says that, if G_1 is nonblocking preserving with respect to G_2 then their nonblocking behaviors are equal, but G_2's blocking behavior may be larger. The third condition is used to guarantee that nonblocking preserving is preserved under automaton product and abstraction. If, in addition, G_2 is nonblocking preserving with respect to G_1, then they are nonblocking equivalent. Next, we discuss synthesis of a distributed supervisor.

2.2 A Distributed Supervisor Synthesis Problem

We first provide concepts of state controllability, state observability, state normality, and nonblocking supervisor, which are introduced in [10]. Then we present a distributed supervisor synthesis problem.

Given $G=(X,\Sigma,\xi,x_0,X_m)$, for each $x\in X$ let

$$E_G:X\to 2^{\Sigma}:x\mapsto E_G(x):=\{\sigma\in\Sigma|\xi(x,\sigma)\neq\varnothing\}$$

Thus, $E_G(x)$ is simply the set of all events allowable at x in G. We now bring in the concept of *state controllability.* Let $\Sigma=\Sigma_c\cup\Sigma_{uc}$, where the disjoint subsets Σ_c and Σ_{uc} denote respectively the set of *controllable* events and the set of *uncontrollable* events. Let $L(G):=\{s\in\Sigma^*|\xi(x_0,s)\neq\varnothing\}$.

Definition 2.4. Given a nondeterministic finite-state automaton $G=(X,\Sigma,\xi,x_0,X_m)$ and $\Sigma'\subseteq\Sigma$, let $A=(Y,\Sigma',\eta,y_0,Y_m)\in\phi(\Sigma')$ and $P:\Sigma^*\to\Sigma'^*$ be the natural projection. A is *state-controllable* with respect to G and Σ_{uc} if

$$(\forall s\in L(G\times A))(\forall x\in\xi(x_0,s))(\forall y\in\eta(y_0,P(s)))\,E_G(x)\cap\Sigma_{uc}\cap\Sigma'\subseteq E_A(y)$$

□

We can check that, A is state controllable implies that $L(G \times A)\Sigma_{uc} \cap L(G) \subseteq L(G \times A)$. Thus, it is always true that state controllability implies language controllability of the product $G \times A$ described in the RW paradigm. But the converse statement is not true unless both A and G are deterministic. We now introduce the concept of *state observability*. Let $\Sigma = \Sigma_o \cup \Sigma_{uo}$, where the disjoint subsets Σ_o and Σ_{uo} denote respectively the set of *observable* events and the set of *unobservable* events. Let $P_o : \Sigma^* \to \Sigma_o^*$ be the natural projection.

Definition 2.5. Given a nondeterministic finite-state automaton $G = (X, \Sigma, \xi, x_0, X_m)$ and $\Sigma' \subseteq \Sigma$, let $A = (Y, \Sigma', \eta, y_0, Y_m) \in \phi(\Sigma')$. A is *state-observable* with respect to G and P_o if for any $s, s' \in L(G \times A)$ with $P_o(s) = P_o(s')$, we have

$$(\forall (x, y) \in \xi\times\eta((x_0, y_0), s))(\forall (x', y') \in \xi\times\eta((x_0, y_0), s'))\, E_{G\times A}(x, y)\cap E_G(x')\cap\Sigma' \subseteq E_A(y')$$

□

Def. 2.5 says that, if A is state observable then for any two states (x, y) and (x', y') in $G\times A$ reachable by two strings s and s' having the same projected image (i.e. $P(s) = P(s')$), any event σ allowed at (x, y) and x' must be allowed at y' as well. We can check that, if A is state-observable then

$$(\forall s, s' \in L(G\times A))(\forall \sigma \in \Sigma)\, P_o(s) = P_o(s') \wedge s\sigma \in L(G\times A) \wedge s'\sigma \in L(G) \Rightarrow s'\sigma \in L(G\times A)$$

Thus, state observability implies language observability of the product $G \times A$. But the converse statement is not always true unless both A and G are deterministic. Notice that, if $\Sigma_o = \Sigma$, namely every event is observable, A may still not be state-observable, owing to nondeterminism. In many applications we are interested in an even stronger observability property called *state normality* which is defined as follows.

Definition 2.6. Given a nondeterministic finite-state automaton $G = (X, \Sigma, \xi, x_0, X_m)$ and $\Sigma' \subseteq \Sigma$, let $A = (Y, \Sigma', \eta, y_0, Y_m) \in \phi(\Sigma')$ and $P : \Sigma^* \to \Sigma'^*$ be the natural projection. A is *state-normal* with respect to G and P_o if for any $s \in L(G \times A)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G \times A)$, we have that, for any $(x, y) \in \xi \times \eta((x_0, y_0), s')$ and $s'' \in \Sigma^*$,

$$P_o(s's'') = P_o(s) \;\wedge\; \xi(x, s'') \neq \varnothing \Rightarrow \eta(y, P(s'')) \neq \varnothing$$

□

We can check that, if A is state-normal with respect to G and P_o, then

$$L(G) \cap P_o^{-1}(P_o(L(G \times A))) \subseteq L(G \times A)$$

which means $L(G \times A)$ is language normal with respect to $L(G)$ and P_o. The converse statement is not true unless both A and G are deterministic. Furthermore, we can check that state normality implies state observability. But the converse statement is not true. We now introduce the concept of supervisor.

Definition 2.7. Given $G \in \phi(\Sigma)$ and $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma' \subseteq \Sigma$, an automaton $S \in \phi(\Sigma')$ is a *nonblocking supervisor* of G under H, if S is deterministic and the following conditions hold:

1. $N(G \times S) \subseteq N(G \times H)$
2. $B(G \times S) = \varnothing$

3. S is state-controllable with respect to G and Σ_{uc}
4. S is state-observable with respect to G and P_o □

The first condition of Def. 2.7 says that the closed-loop system $G \times S$ complies with the specification H in terms of language inclusion. Because of this condition we only consider H to be deterministic. The use of a nondeterministic specification is described in, e.g. [18]. Later we will use the term 'nonblocking state-normal supervisor' (NSN), when we want to emphasize that S is state-normal with respect to G and P_o. It has been shown in [10] that the set

$$\mathcal{CN}(G,H) := \{S \in \phi(\Sigma')|S \text{ is a NSN supervisor of } G \text{ w.r.t. } H \wedge L(S) \subseteq L(G)\}$$

contains a unique element $\hat{S}$ such that for any $S \in \mathcal{CN}(G,H)$, we have $N(S) \subseteq N(\hat{S})$. We call $\hat{S}$ the *supremal nonblocking state-normal supervisor* of G under H. In practice it is of our primary interest to compute such a supremal NSN supervisor, which will be discussed in the next section.

Definition 2.8. A *distributed system* with respect to given alphabets $\{\Sigma_i|i \in I\}$ is a set of nondeterministic finite-state automata $\mathcal{G} := \{G_i = (X_i, \Sigma_i, \xi_i, x_{i,0}, X_{i,m}) \in \phi(\Sigma_i)|i \in I\}$. Each G_i $(i \in I)$ is called the i^{th} *component* of $\mathcal{G}$, and $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,uc} = \Sigma_{i,o} \cup \Sigma_{i,uo}$, where disjoint subsets $\Sigma_{i,c}$ and $\Sigma_{i,uc}$ comprise respectively the *controllable* events and *uncontrollable* events, and disjoint subsets $\Sigma_{i,o}$ and $\Sigma_{i,uo}$ comprise respectively the *observable* events and *unobservable* events. □

We make the following assumption:

$$(\forall i,j \in I)\, i \neq j \Rightarrow \Sigma_{i,c} \cap \Sigma_{j,uc} = \varnothing \wedge \Sigma_{i,o} \cap \Sigma_{j,uo} = \varnothing \qquad \text{(A1)}$$

namely there is no event, which is controllable in G_i but uncontrollable in G_j $(i \neq j)$; and there is also no event, which is observable in G_i but unobservable in G_j $(i \neq j)$. For many applications this is a mild assumption and can be easily satisfied. There may exist cases in which a single event may have different controllability or observability properties in different components. Although it is still possible to deal with these cases by applying aggregative synthesis, we choose not to do that in this paper because it may create extra complications that are not helpful for conveying our main idea of aggregative synthesis.

Distributed Supervisory Control Problem: Given a distributed system $\mathcal{G} = \{G_i \in \phi(\Sigma_i)|i \in I\}$ and a set of specifications $\mathcal{H} = \{H_j \in \phi(\Delta_j)|\Delta_j \subseteq \cup_{i\in I}\Sigma_i \wedge j \in J\}$, where J is an index set and each H_j is a deterministic automaton, synthesize a collection of deterministic finite-state automata

$$\mathcal{S} = \{S_k \in \phi(\Gamma_k)|\Gamma_k \subseteq \cup_{i\in I}\Sigma_i \wedge k \in K\}$$

where K is an index set, such that the following conditions hold,

1. $N((\times_{i\in I}G_i) \times (\times_{k\in K}S_k)) \subseteq N((\times_{i\in I}G_i) \times (\times_{j\in J}H_j))$
2. $B((\times_{i\in I}G_i) \times (\times_{k\in K}S_k)) = \varnothing$
3. $\times_{k\in K}S_k$ is state-controllable with respect to $\times_{i\in I}G_i$ and $\cup_{i\in I}\Sigma_{i,uc}$

4. $\times_{k\in K} S_k$ is state-normal with respect to $\times_{i\in I} G_i$ and $P_o : (\cup_{i\in I}\Sigma_i)^* \to (\cup_{i\in I}\Sigma_{i,uo})^*$

If such a collection $\mathcal{S}$ exists, then it is called a *nonblocking distributed supervisor* of $\mathcal{G}$ under $\mathcal{H}$, where each S_k is a *local supervisor* of $\mathcal{G}$ under $\mathcal{H}$. □

Next, we present an aggregative approach to synthesize a nonblocking distributed supervisor.

3 Aggregative Synthesis of Nonblocking Distributed Supervisors

We first discuss how to compute a supremal nonblocking state-normal supervisor S of a nondeterministic plant model G under a deterministic specification H. Then we present an aggregative synthesis approach for a nonblocking distributed supervisor.

3.1 Computation of Supremal Nonblocking State-Normal Supervisor

Let $G = (X, \Sigma, \xi, x_0, X_m)$ be a nondeterministic automaton, and $H = (Z, \Delta, \delta, z_0, Z_m)$ be a deterministic automaton with $\Delta \subseteq \Sigma$. We would like to synthesize the supremal nonblocking state-normal supervisor $S = (Y, \Sigma, \eta, y_0, Y_m)$. By the previous discussion we know that such a supremal supervisor exists. Let $P_o : \Sigma^* \to \Sigma_o^*$ be the natural projection. We now present an algorithm to compute S.

Given a deterministic finite-state automaton $S = (Y, \Sigma, \eta, y_0, Y_m)$, let $\chi(S, G) = (Q, \Sigma, \varrho, q_0, Q'_m)$ be an automaton, where

$$Q := \{(x, y) \in X \times Y | (\forall s \in \Sigma_{uc}^*)\, [\xi(x, s) \neq \varnothing \Rightarrow \eta(y, s) \neq \varnothing] \\ \wedge (\exists s \in \Sigma^*)\, \xi \times \eta((x, y), s) \cap (X_m \times Y_m) \neq \varnothing\}$$

$Q_m = Q \cap (X_m \times Y_m)$ and $\varrho : Q \times \Sigma^* \to 2^Q : (q, s) \mapsto \varrho(q, s) := (\xi \times \eta(q, s)) \cap Q$. From the construction, it is possible that some states of $\chi(S, G)$ may not be reachable via ϱ. For a slight abuse of notation, we use $\chi(S, G)$ to denote its reachability part under ϱ. If $\chi(S, G)$ is the same as $G \times S$ under automaton isomorphism [24], then $G \times S$ is nonblocking and state-controllable with respect to G and Σ_{uc}. Let

$$\varphi(\chi(S, G)) := \{q \in Q | (\exists \sigma \in \Sigma)\, \varrho(q, \sigma) \subset \xi \times \eta(q, \sigma)\}$$

which contains every state q in $\chi(S, G)$ that requires event disabling in the sense that there exists at least one transition $\sigma \in \Sigma$ allowed at q in $G \times S$, but not at q in $\chi(S, G)$, namely $\xi \times \eta(q, \sigma) - \varrho(q, \sigma) \neq \varnothing$. Define $\varpi(\chi(S, G)) = (W, \Sigma, \theta, w_0, W_m)$ as an automaton, where

$$W := \{q \in Q | (\exists s \in \Sigma^*)\, \varrho(q, s) \cap \varphi(\chi(S, G)) \neq \varnothing\} \cup \{d\}$$

with $d \notin X \times Y$, $W_m := \{d\}$, $w_0 := (x_0, y_0)$ and $\theta : W \times \Sigma \to 2^W$, where,

$$(\forall w \in W)(\forall \sigma \in \Sigma)\, \theta(w,\sigma) := \begin{cases} \varrho(w,\sigma) & \text{if } w \in Q \wedge \xi \times \eta(w,\sigma) \subseteq Q \\ \varrho(w,\sigma) \cup \{d\} & \text{if } w \in Q \wedge \xi \times \eta(w,\sigma) \nsubseteq Q \\ \{d\} & \text{if } w = d \end{cases}$$

The automaton $\varpi(\chi(S,G))$ contains every string $s\sigma \in \Sigma^*$ that can 'move out of $\chi(S,G)$' in the sense that s reaches some state q in $\chi(S,G)$, where a transition σ is allowed at q in $G \times S$ but not at q in $\chi(S,G)$. Owing to nondeterminism, it is still possible that $\varrho(q,\sigma) \neq \varnothing$, when $\varrho(q,\sigma) \subset \xi \times \eta(q,\sigma)$. We can check that, if $\chi(S,G)$ is state-normal with respect to G and P_o, then no string in $N(\varpi(\chi(S,G)))$ has the same projected image as some string in $L(\chi(S,G))$ (or equivalently in $N(\chi(S,G))$ because $\chi(S,G)$ is required to be nonblocking). We now present the following algorithm to compute the supremal nonblocking state-normal supervisor.

Procedure for Supremal Nonblocking State-Normal Supervisor (PSNSNS)

1. Inputs: nondeterministic $G \in \phi(\Sigma)$ and deterministic $H \in \phi(\Delta)$, where $\Delta \subseteq \Sigma$.

2. Initialization: let S^0 be a deterministic recognizer of $N(G \times H)$.

3. For each $k = 0, 1, 2, \cdots$, if $N(\chi(S^k,G)) \cap P_o^{-1}(P_o(N(\varpi(\chi(S^k,G))))) \neq \varnothing$ then

   - Let S^{k+1} be a deterministic recognizer of $N(\chi(S^k,G)) - P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))$

   Otherwise, terminate and output S as a deterministic recognizer of $N(\chi(S^k,G))$ with $B(S) = \varnothing$. □

Proposition 3.1. PSNSNS terminates within a finite number of steps. □

The proof is presented in the Appendix. From the proof of Prop. 3.1 we can derive that, the maximum number of states at each stage of PSNSNS is $O(|X||Z|2^{|X||Z|})$, which is the maximum size of G_{NH}, where $|X|$ and $|Z|$ denote the sizes of X and Z respectively. Since H is deterministic, we can minimize the size of Z by simply using the canonical recognizer of $N(H)$. The complexity of PSNSNS is very similar to the algorithm SCOP presented in [9], which computes supremal nonblocking controllable and normal supervisors. The only difference is that, in SCOP the plant model G is deterministic - thus, the size of X can be minimized by simply using the canonical recognizer of $N(G)$. In our case we cannot do that because G is nondeterministic. We have the following result, whose proof is in the Appendix.

Theorem 3.2. When PSNSNS terminates, the nonempty output S is the supremal nonblocking state-normal supervisor of G under H. □

Theorem 3.2 shows that we can use PSNSNS to compute the supremal nonblocking state-normal supervisor, whenever a plant model G and a deterministic specification H is given. We will use this result in aggregative synthesis of nonblocking distributed supervisor, which is described next.

3.2 Aggregative Synthesis of Nonblocking Distributed Supervisors for Standardized Automata

In this section we will apply automaton abstraction in supervisor synthesis. To this end we bring in a new event symbol τ, which does not belong to any alphabet, and is always treated as uncontrollable and unobservable. We call a nondeterministic finite-state automaton $G^{\tau} = (X, \Sigma \cup \{\tau\}, \xi, x_0, X_m)$ *standardized* if

1. $x_0 \notin X_m \wedge (\forall x \in X)\,[\xi(x, \tau) \neq \varnothing \iff x = x_0] \wedge (\forall \sigma \in \Sigma)\,\xi(x_0, \sigma) = \varnothing$
2. $(\forall x \in X)(\forall \sigma \in \Sigma \cup \{\tau\})\, x_0 \notin \xi(x, \sigma)$

A standardized automaton is nothing but an automaton, in which x_0 is not marked, τ is only defined at x_0, which only has outgoing τ transitions and no incoming transition. For notation simplicity, from now on we assume that every alphabet Σ contains τ, unless specified otherwise, and we use $\phi(\Sigma)$ to denote the collection of all standardized automata over Σ. Only when we want to discuss the relationship between an automaton and its standardized version, we bring in the superscript τ. We can easily check that, abstraction of a standardized automaton is still standardized and the product of two standardized automata is also standardized. The reason to introduce standardized automata is to obtain the following results.

Proposition 3.3. *[10]* Given $G \in \phi(\Sigma)$, let $\Sigma' \subseteq \Sigma$, $P : \Sigma^* \to \Sigma'^*$ be the natural projection. Then we have $P(B(G)) \subseteq B(G/\approx_{\Sigma'})$ and $P(N(G)) = N(G/\approx_{\Sigma'})$. □

Proposition 3.4. *[10]* Given Σ_1 and Σ_2, let $G_1 \in \phi(\Sigma_1)$, $G_2 \in \phi(\Sigma_2)$ and $\Sigma' \subseteq \Sigma_1 \cup \Sigma_2$. If $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma'$, then we have $(G_1 \times G_2)/\approx_{\Sigma'} \sqsubseteq (G_1/\approx_{\Sigma_1 \cap \Sigma'}) \times (G_2/\approx_{\Sigma_2 \cap \Sigma'})$. □

If relevant automata are not standardized, then Prop. 3.3 and Prop. 3.4 may not hold, which will make the proposed aggregative synthesis based on automaton abstraction fail. We also need the following result.

Proposition 3.5. *[10]* Given Σ and Σ', let $G_1, G_2 \in \phi(\Sigma)$ and $G_3 \in \phi(\Sigma')$. If $G_1 \cong G_2$ then $G_1 \times G_3 \cong G_2 \times G_3$. □

To discuss aggregative synthesis of a nonblocking distributed supervisor, we first consider a 2-component distributed system. Then we extend it to a general distributed system. We have the following result.

Theorem 3.6. Given $G_i \in \phi(\Sigma_i)$ $(i = 1, 2)$ and two specifications $H_1 \in \phi(\Delta_1)$ with $\Delta_1 \subseteq \Sigma_1$ and $H_2 \in \phi(\Delta_2)$ with $\Delta_2 \subseteq \Sigma_1 \cup \Sigma_2$, let $\Sigma' \subseteq \Sigma_1$ and $\Sigma_1 \cap (\Sigma_2 \cup \Delta_2) \subseteq \Sigma'$. Suppose there exist a nonblocking state-normal supervisor $S_1 \in \phi(\Sigma_1)$ of G_1 under H_1, and a nonblocking state-normal supervisor $S_2 \in \phi(\Sigma_2 \cup \Sigma')$ of $((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2$ under H_2. Then $S_1 \times S_2$ is a nonblocking state-normal supervisor of $G_1 \times G_2$ under $H_1 \times H_2$.□

The proof is given in the Appendix. Theorem 3.6 allows us to synthesize a distributed supervisor in an aggregate way. Without loss of generality, suppose $I = \{1, 2, \cdots, n\}$.

We put an order on those local components, say $(G_1, G_2, \cdots, G_n)$. Let $\mathcal{H} = \{H_j | j \in J\}$ be the collection of specifications. Then we perform the following construction.

Aggregate Synthesis of Standardized Distributed Supervisor (ASSDS)

1. Inputs: standardized $\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I\}$ and $\mathcal{H} = \{H_j \in \phi(\Delta_j) | j \in J\}$.
2. Initially set $W_1 := G_1$, $J_1 := \{j \in J | \Delta_j \subseteq \Sigma_1\}$, $Q_1 := J_1$ and $T_1 := \Sigma_1$.
3. For $k = 1, \cdots, n$,
   (a) If $J_k \neq \varnothing$, let $V_k := \times_{j \in J_k} H_j$. Otherwise, set V_k as a recognizer of Σ_k^*.
   (b) Synthesize the supremal NSN supervisor S_k of W_k under V_k (by PSNSNS).
   (c) Terminate when S_k is empty or $k = n$. Otherwise, updates the following.
   (d) Set $I_{k+1} := \{i \in I | k+1 \leq i \leq n\}$, $\Sigma_{I_{k+1}} := \cup_{i \in I_{k+1}} \Sigma_i$, $\Theta_{k+1} := \cup_{j \in J - Q_k} \Delta_j$.
   (e) Choose $\Sigma_{A_k} \subseteq T_k$ with $(\Sigma_{I_{k+1}} \cup \Theta_{k+1}) \cap T_k \subseteq \Sigma_{A_k}$. Let $A_k := (W_k \times S_k)/\approx_{\Sigma_{A_k}}$.
   (f) $W_{k+1} := A_k \times G_{k+1}$.
   (g) $Q_{k+1} := \{j \in J | \Delta_j \subseteq \cup_{i=1}^{k+1} \Sigma_i\}$.
   (h) $J_{k+1} := Q_{k+1} - Q_k$
   (i) $T_{k+1} := \Sigma_{A_k} \cup \Sigma_{k+1}$.
4. When terminate upon k, output $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$. $\square$

To explain ASSDS, suppose $n = 3$ and the ordering of components is G_1, G_2, G_3. Suppose there are $r \in \mathbb{N}$ specifications: $H_1, H_2, \cdots, H_r$. Among these specifications, suppose specifications $H_1, \cdots, H_m$ $(m \leq r)$ 'touch' only G_1 in the sense that $\Delta_i \subseteq \Sigma_1$ for $i = 1, 2, \cdots, m$, and specifications $H_{m+1}, \cdots, H_k$ touch only G_1 and G_2 but not G_3, namely $\Delta_j \subseteq \Sigma_1 \cup \Sigma_2$ and $\Delta_j \cap \Sigma_3 = \varnothing$ for $j = m+1, 2, \cdots, k$, and $H_{k+1}, H_{k+2}, \cdots, H_r$ touch not only G_1 and G_2 but also G_3, namely $\Delta_j \subseteq \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ and $\Delta_j \cap \Sigma_3 \neq \varnothing$ for $j = k+1, 2, \cdots, r$. What ASSDS does is as follows. First, it computes the supremal nonblocking state-normal supervisor S_1 of $W_1 = G_1$ under the specification $V_1 = H_1 \times \cdots \times H_m$. When $\{H_1, \cdots, H_m\} = \varnothing$, ASSDS simply sets V_1 to be the canonical recognizer of Σ_1^*. In this case only nonblockingness of the closed-loop behavior is the synthesis goal. To achieve a nonblocking supervisor S_2, an abstraction A_1 of $G_1 \times S_1 = W_1 \times S_1$ is created. The alphabet Σ_{A_1} is chosen by whatever convenient reasons, as long as the condition $\Sigma_1 \cap (\Sigma_2 \cup \Sigma_3 \cup_{i=m+1}^{r} \Delta_i) \subseteq \Sigma_{A_1} \subseteq \Sigma_1$ holds. The reason of imposing this condition is that in the subsequent computation we can always use A_1 to replace $G_1 \times S_1$. If we do not want A_1 to lose too much information about controllability during abstraction, we can set $\Sigma_{1,c} \subseteq \Sigma_{A_1}$. Of course, too many events remaining in Σ_{A_1} may result in an abstraction with few states being removed from G_1. So there is a tradeoff issue that we need to deal with when we choose Σ_{A_1}, and such a tradeoff is, in our opinion, case-dependent. We now have a plant $A_1 \times G_2$ and a specification $V_2 = H_{m+1} \times \cdots \times H_k$. By the previous description we can compute the supremal nonblocking state-normal supervisor S_2 of $W_2 = A_1 \times G_2$ under V_2. Suppose S_2 exists, then we can create an abstraction A_2 of $A_1 \times G_2 \times S_2 = W_2 \times S_2$, and a new plant $W_3 = A_2 \times G_3$. We then synthesize the supremal nonblocking state-normal supervisor S_3 of W_3 under $V_3 = H_{k+1} \times \cdots \times H_r$.

Theorem 3.7. Let $\mathcal{S} = \{S_1, \cdots, S_n\}$ be computed by ASSDS, where none of S_k $(k = 1, 2, \cdots, n)$ is empty. Then $\mathcal{S}$ is a nonblocking distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. $\square$

Proof: We claim that, for any k with $1 < k \leq n$, $S_k \times \cdots \times S_n$ is a nonblocking state-normal supervisor of $A_{k-1} \times G_k \times \cdots \times G_n$ under $V_k \times \cdots \times V_n$. We show this claim by using induction on k.
(1) Base case: When $k = n$, since S_n is a nonblocking state-normal supervisor of $W_n = A_{n-1} \times G_n$ under V_n, the claim is true.
(2) Hypothesis: Suppose the claim holds for $k > 2$.
(3) Induction part: We need to show that the claim holds for $k-1$. Since $A_{k-1} = (W_{k-1} \times S_{k-1}) / \approx_{\Sigma_{A_{k-1}}} = (A_{k-2} \times G_{k-1} \times S_{k-1}) / \approx_{\Sigma_{A_{k-1}}}$ and S_{k-1} is a nonblocking state-normal supervisor of $W_{k-1} = A_{k-2} \times G_{k-1}$ under V_{k-1} and $T_{k-1} \cap (\Sigma_{I_k} \cup \Theta_k) \subseteq \Sigma_{A_{k-1}}$, by Theorem 3.6 and the induction hypothesis, we get that, $S_{k-1} \times S_k \times \cdots \times S_n$ is a nonblocking state-normal supervisor of $A_{k-2} \times G_{k-1} \times G_k \times \cdots \times G_n$ under $V_{k-1} \times \cdots \times V_n$. Thus, the induction part is true, which means the claim is true.
By the claim we get that, $S_2 \times \cdots \times S_n$ is a nonblocking state-normal supervisor of $A_1 \times G_2 \times \cdots \times G_n$ under $V_2 \times \cdots \times V_n$. Since $A_1 = (W_1 \times S_1) / \approx_{\Sigma_{A_1}} = (G_1 \times S_1) / \approx_{\Sigma_{A_1}}$ and S_1 is a nonblocking state-normal supervisor of G_1 under V_1, by Theorem 3.6 we get that $S_1 \times \cdots \times S_n$ is a nonblocking state-normal supervisor of $G_1 \times \cdots \times G_n$ under $V_1 \times \cdots \times V_n = \times_{j \in J} H_j$. Thus, the theorem follows. ■

Although during the above construction $\mathcal{S}$ contains the same number of local supervisors as that of local components, several may not impose any control on the system. This can be checked whenever a local supervisor S_k is computed, and S_k imposes no control if and only if $L(S_k) = L(W_k)$. In that case we simply remove those local supervisors from $\mathcal{S}$ during online supervisory control.

Clearly, the ordering is important not only for the computational complexity purpose but also for the existence of a distributed supervisor. Given a distributed system $\mathcal{G}$, some ordering of local components may yield empty distributed supervisory control under ASSDS. How to choose a good ordering is an interesting and important problem. Currently, we adopt a heuristic ordering procedure, which says that, *for any two components next to each other in an ordering, they must share events.* The rationality of this heuristics is that strongly coupled components (in terms of interactions through event sharing) should always be ordered close to each other. For example, suppose we have three components: a motor, a conveyor belt and a robot, where the motor drives the conveyor belt to move goods which are picked up by the robot. Given two orderings: (1) the motor, the conveyor belt and the robot; (2) the motor, the robot and the conveyor belt, it seems more reasonable for us to prefer ordering (1) to ordering (2) because there is no direct connection between the motor and the robot. This heuristics is used in the example provided in Section IV. We are still searching for other heuristic procedures that may work better than this simple one.

By imposing an ordering over local components we may also attain a limited power of reusing local supervisors when some local component is added to the target system or dropped out of it, as often encountered in system reconfiguration. For example, suppose we have a distributed supervisor $\{S_1, \cdots, S_n\}$ with respect to an ordering $(G_1, \cdots, G_n)$. If we change or remove G_k $(1 \leq k \leq n)$, we only need to redesign local supervisors $\{S_r, \cdots, S_n\}$, where $r = \max\{1, k\}$. If we add some component $\hat{G}$ after G_k and before G_{k+1}, then we only need to redesign local supervisors associated with $\{\hat{G}, G_{k+1}, \cdots, G_n\}$. Thus, a certain degree of implementation flexibility is achieved.

3.3 Synthesis of Nonblocking Distributed Supervisors of Non-Standardized Distributed Systems

In the previous subsection we present an aggregative approach to synthesize a nonblocking distributed supervisor of a distributed system $\mathcal{G}$ under a set of deterministic specifications $\mathcal{H}$. Nevertheless, all relevant automata are required to be standardized, for the sake of using automaton abstraction effectively. It is our great interest to know how to synthesize a nonblocking distributed supervisor for a distributed system modeled by non-standardized automata. To this end we present a simple procedure. But before that we first introduce the concepts of *standardization* and *de-standardization*. To avoid unnecessary confusion, we want to emphasize that, in this section we assume that τ is not contained in any alphabet, and $\phi(\Sigma)$ denotes the collection of all *non-standardized* automata, whose alphabet is Σ.

Definition 3.8. Given a nondeterministic finite-state automaton $G = (X, \Sigma, \xi, x_0, X_m)$, we say an automaton $G^\tau = (X^\tau, \Sigma \cup \{\tau\}, \xi^\tau, x_0^\tau, X_m^\tau)$ is *G-standardized* if

1. $X^\tau = X \cup \{x_0^\tau\}$, where $x_0^\tau \notin X$
2. $X_m^\tau = X_m$
3. $(\forall x \in X \cup \{x_0^\tau\})(\forall \sigma \in \Sigma \cup \{\tau\})\, \xi^\tau(x, \sigma) := \begin{cases} \xi(x,\sigma) & \text{if } x \in X \text{ and } \sigma \in \Sigma \\ x_0 & \text{if } x = x_0^\tau \text{ and } \sigma = \tau \\ \varnothing & \text{otherwise} \end{cases}$

□

The only difference between G^τ and G is that, the former contains a new state x_0^τ and a new transition from x_0^τ to x_0. As we have seen in the previous subsection, this new transition plays a crucial role in automaton abstraction, thus a crucial role in aggregative synthesis. From now on we use $\mu(G)$ to denote the G-standardized automaton G^τ. Next, we introduce the concept of *destandardization*, which is used to convert a standardized automaton into a nonstandardized one.

Definition 3.9. Let $S^\tau = (Y^\tau, \Sigma \cup \{\tau\}, \eta^\tau, y_0^\tau, Y_m^\tau)$ be a deterministic standardized automaton. We say an automaton $S = (Y, \Sigma, \eta, y_0, Y_m)$ is S^τ*-destandardized* if

1. $Y := Y^\tau - \{y_0^\tau\}$
2. $Y_m := Y_m^\tau$
3. $y_0 \in \eta^\tau(y_0^\tau, \tau)$
4. $\eta : Y \times \Sigma \to 2^Y : (x, \sigma) \mapsto \eta(x, \sigma) := \eta^\tau(x, \sigma)$ □

Since S^τ is deterministic, $\eta^\tau(y_0^\tau, \tau)$ contains only one element. Thus, S is well defined. The only difference between S^τ and its destandardized version S is that, the latter contains no transition τ. From now on we use $\nu(S^\tau)$ to denote the S^τ-destandardized automaton S. We have the following result.

Theorem 3.10. Given a distributed system $\mathcal{G} = \{G_i \in \phi(\Sigma_i)|i \in I\}$ and a collection of deterministic specifications $\mathcal{H} = \{H_j \in \phi(\Delta_j)|\Delta_j \subseteq \cup_{i\in I}\Sigma_i \wedge j \in J\}$, let $\mathcal{G}^\tau := \{\mu(G_i)|i \in I\}$ be the standardized distributed system and $\mathcal{H}^\tau := \{\mu(H_j)|j \in J\}$ for the standardized deterministic specifications. If there exists a nonblocking distributed supervisor $\mathcal{S}^\tau := \{S_k^\tau \in \phi(\Gamma_k^\tau)|\Gamma_k^\tau \subseteq \cup_{i\in I}(\Sigma_i \cup \{\tau\}) \wedge k \in K\}$ of $\mathcal{G}^\tau$ under $\mathcal{H}^\tau$, then $\mathcal{S} := \{\nu(S_k^\tau)|k \in K\}$ is a nonblocking distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. □

Proof: Let $P : (\cup_{i\in I}\Sigma_i \cup \{\tau\})^* \to (\cup_{i\in I}\Sigma_i)^*$, $P_o^\tau : (\cup_{i\in I}\Sigma_i \cup \{\tau\})^* \to (\cup_{i\in I}\Sigma_{i,o})^*$ and $P_o : (\cup_{i\in I}\Sigma_i)^* \to (\cup_{i\in I}\Sigma_{i,o})^*$ be the natural projections. By the definitions of automaton product, standardization and destandardization, we get that

$$\begin{aligned} & N(\times_{i\in I}G_i \times_{k\in K} \nu(S_k^\tau)) \\ = \; & P(N(\times_{i\in I}\mu(G_i) \times_{k\in K} S_k^\tau)) \\ \subseteq \; & P(N(\times_{i\in I}\mu(G_i) \times_{j\in J} \mu(H_j))) \text{ because } \mathcal{S}^\tau \text{ is a nonblocking supervisor of } \mathcal{G}^\tau \text{ under } \mathcal{H}^\tau \\ = \; & N(\times_{i\in I}G_i \times_{j\in J} H_j) \end{aligned}$$

and

$$B(\times_{i\in I}G_i \times_{k\in K} \nu(S_k^\tau)) = P(B(\times_{i\in I}\mu(G_i) \times_{k\in K} S_k^\tau)) = P(\varnothing) = \varnothing$$

Since $\times_{i\in I}\mu(G_i) = \mu(\times_{i\in I}G_i)$ and $\times_{k\in K}S_k^\tau = \mu(\times_{k\in K}\nu(S_k^\tau))$, where equality '=' is in the sense of DES-isomorphism [24], by the definitions of standardization and automaton product, we get that: (1) $\times_{k\in K}S_k^\tau$ is state-controllable with respect to $\times_{i\in I}G_i^\tau$ and $\cup_{i\in I}\Sigma_{i,uc} \cup \{\tau\}$ if and only if $\times_{k\in K}\nu(S_k^\tau)$ is state-controllable with respect to $\times_{i\in I}G_i$ and $\cup_{i\in I}\Sigma_{i,uc}$; (2) $\times_{k\in K}S_k^\tau$ is state-observable with respect to $\times_{i\in I}G_i^\tau$ and P_o^τ if and only if $\times_{k\in K}\nu(S_k^\tau)$ is state-observable with respect to $\times_{i\in I}G_i$ and P_o; and (3) $\times_{k\in K}S_k^\tau$ is state-normal with respect to $\times_{i\in I}G_i^\tau$ and P_o^τ if and only if $\times_{k\in K}\nu(S_k^\tau)$ is state-normal with respect to $\times_{i\in I}G_i$ and P_o. Thus, the theorem follows. ■

Theorem 3.10 allows us to use the following procedure to synthesize a nonblocking distributed supervisor of a non-standardized distributed system under deterministic specifications.

Aggregate Synthesis of Non-Standardized Distributed Supervisor (ASNSDS):

1. Inputs: $\mathcal{G} = \{G_i \in \phi(\Sigma_i)|i \in I = \{1, 2, \cdots, n\}\}$ and $\mathcal{H} = \{H_j \in \phi(\Delta_j)|j \in J\}$
2. Create $\mathcal{G}^\tau = \{\mu(G_i)|i \in I\}$ and $\mathcal{H}^\tau = \{\mu(H_j)|j \in J\}$
3. Apply ASSDS on $\mathcal{G}^\tau$ and $\mathcal{H}^\tau$ to compute $\mathcal{S}^\tau = \{S_k^\tau|k \in K = \{2, 3, \cdots, n\}\}$
4. Output $\mathcal{S} = \{\nu(S_k^\tau)|k \in K\}$ □

At this point we can see that, introducing the notion of τ and the concept of standardized automata, which are crucially important for automaton abstraction, does not impose any restriction on supervisor synthesis. Next, we use a concrete example to show the effectiveness of ASNSDS.

4 Example - A Cluster Tool

A cluster tool is an integrated manufacturing system used for wafer processing. It consists of *load locks* for wafer entering and leaving the system, *chambers*, where wafers are processed, *buffers* between different clusters in the system, and *transportation robots* for moving wafers in the system [26]. To illustrate the effectiveness of the proposed aggregative synthesis procedure, we consider the following cluster tool depicted in Figure 1, which consists of one entering load lock (L_{in}) and one exit load lock (L_{out}), nine chambers (C_{11}, C_{12}, C_{21}, C_{22}, C_{31}, C_{32}, C_{41}, C_{42}, C_{43}), three one-slot buffers (B_1, B_2, B_3), and four transportation robots (R_1, R_2, R_3 and R_4). Wafers are transported into the system from the entering load lock by the robot R_1, then moved through designated chambers for processing based on pre-specified routing sequences by relevant robots located in different clusters. Finally, processed wafers are transported out of the system through exit load lock by R_1. As an illustration, we choose the following routing sequence:

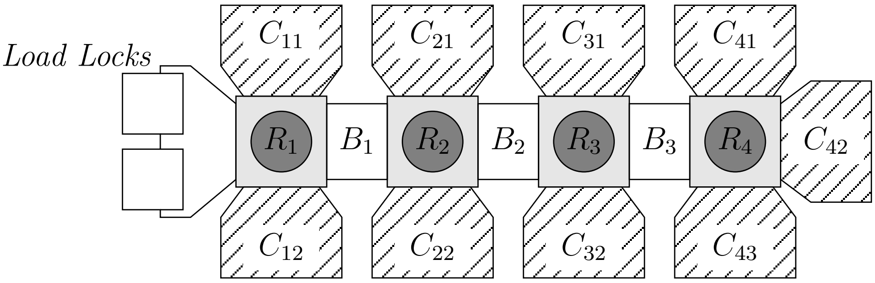

Figure 1: Structure of Cluster Tool

$$L_{in} \to C_{11} \to B_1 \to C_{21} \to B_2 \to C_{31} \to B_3 \to C_{41} \to C_{42} \to C_{43} \to B_3 \to C_{32} \to B_2 \\ \to C_{22} \to B_1 \to C_{12} \to L_{out}$$

Without supervision the system may be blocked owing to wafers competing for buffer slots. Our goal is to synthesize a distributed supervisor that can guarantee continuous wafer processing, namely blocking should never happen. To this end, we first model the system as follows.

For simplicity we assume that the entering load lock C_{in} behaves like an infinite wafer source and the exit load lock C_{out} like an infinite wafer sink. Figure 2 depicts the models of load locks. We assume that in each chamber a wafer is first dropped in by a relevant

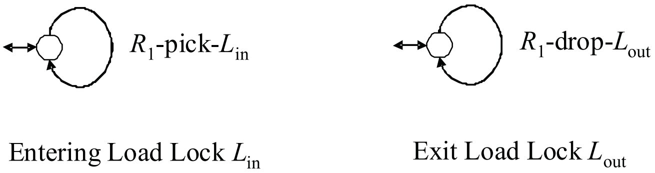

Figure 2: Load Locks

robot, then processed and finally picked up by the relevant robot. Since each chamber has the same automaton model, except for different alphabets, we only provide the model for one chamber, which is depicted in Figure 3, where, when $i = 1, 2, 3$, we have $j = 1, 2$,

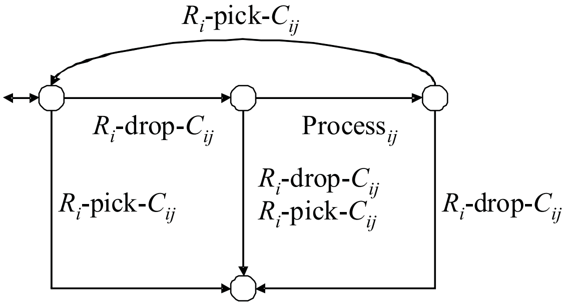

Figure 3: Model of Chamber C_{ij}

and when $i = 4$, we have $j = 1, 2, 3$. Notice that each chamber behaves like a one-slot buffer, except that it contains an internal transition Process$_{ij}$. If robot R_i tries to pick when the chamber is empty, or drop when the chamber is full, the component will become deadlock. By modeling in such a way we will force a nonblocking supervisor to prevent inappropriate pick or drop actions to happen. The models of robots are depicted in Figure 4. Finally we model each buffer B_i $(i = 1, 2, 3)$ as a component, whose model is

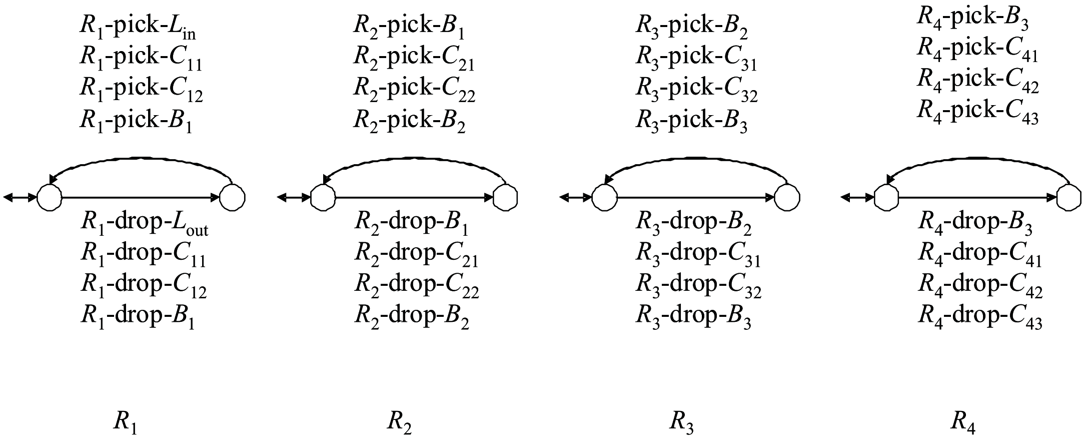

Figure 4: Models of Robots

provided in Figure 5. It says that, buffer overflow or underflow will result in deadlock. In these models we assume that all events of the robots are controllable and observable, and events

Process$_{11}$, Process$_{12}$, Process$_{21}$, Process$_{22}$, Process$_{31}$, Process$_{32}$, Process$_{41}$, Process$_{42}$, Process$_{43}$

are uncontrollable and unobservable. The local specifications are depicted in Figure 6.

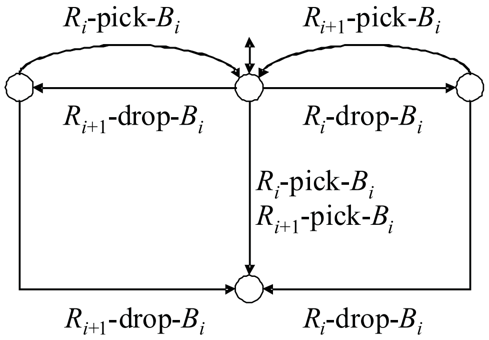

Figure 5: Model of Buffer B_i

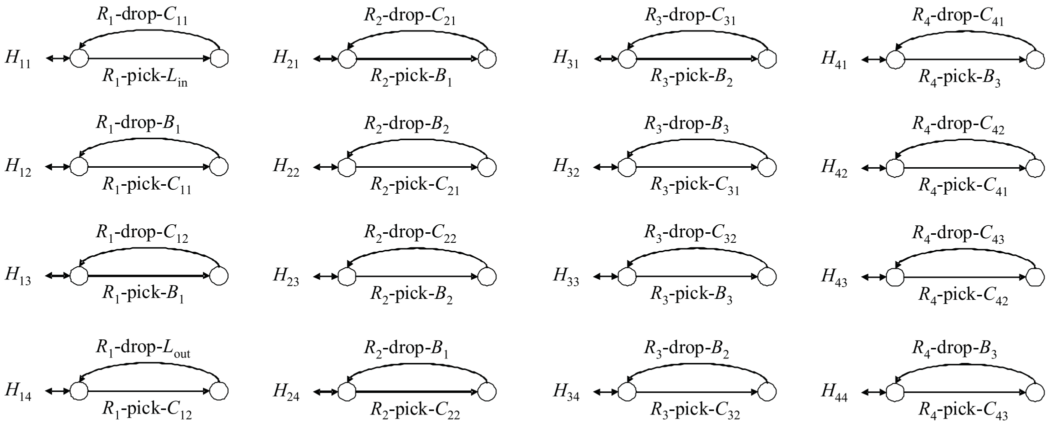

Figure 6: Models of Local Specifications

We now apply the proposed procedure ASNSDS to compute a nonblocking distributed supervisor. First, each component is standardized. Let

$$\begin{aligned}
G_1^\tau &:= \mu(C_{41}) \times \mu(C_{42}) \times \mu(C_{43}) \times \mu(R_4) \times \mu(B_3)\\
G_2^\tau &:= \mu(C_{31}) \times \mu(C_{32}) \times \mu(R_3) \times \mu(B_2)\\
G_3^\tau &:= \mu(C_{21}) \times \mu(C_{22}) \times \mu(R_2) \times \mu(B_1)\\
G_4^\tau &:= \mu(C_{11}) \times \mu(C_{12}) \times \mu(R_1) \times \mu(L_{in}) \times \mu(L_{out})
\end{aligned}$$

and

$$\begin{aligned}
H_1^\tau &:= \mu(H_{41}) \times \mu(H_{42}) \times \mu(H_{43}) \times \mu(H_{44})\\
H_2^\tau &:= \mu(H_{31}) \times \mu(H_{32}) \times \mu(H_{33}) \times \mu(H_{34})\\
H_3^\tau &:= \mu(H_{21}) \times \mu(H_{22}) \times \mu(H_{23}) \times \mu(H_{24})\\
H_4^\tau &:= \mu(H_{11}) \times \mu(H_{12}) \times \mu(H_{13}) \times \mu(H_{14})
\end{aligned}$$

Based on the structure of this system and the previously described heuristic ordering procedure, we simply order $\{G_i^\tau | i = 1, 2, 3, 4\}$ as indicated by their individual subscripts. Then we apply ASDS with the inputs $\{G_i^\tau | i = 1, 2, 3, 4\}$ and $\{H_i^\tau | i = 1, 2, 3, 4\}$. For the illustration purpose, we go through some details of ASDS. Since only H_1^τ touches G_1^τ, we synthesize the supremal nonblocking state-normal supervisor S_1^τ of G_1^τ under H_1^τ. The relevant computational results are listed as follows:

$$G_1^\tau \ (209,\ 729)\ ;\ H_1^\tau \ (17,\ 65)\ ;\ S_1^\tau \ (112,\ 222)$$

where in each tuple (x, y), x denotes the number of states and y for the number of transitions. Next, we compute the abstraction of S_1. To this end we choose the alphabet $\Sigma_{A_1} = \Sigma_{B_3}$ because, to control G_2^τ we intend to use only controllable events in B_3 and G_2^τ, although controllable events of B_3 actually describe behaviors of robots R_3 and R_4. Once Σ_{A_1} is chosen, we compute the abstraction

$$A_1 := (G_1^\tau \times S_1^\tau)/\approx_{\Sigma_{A_1}} \ (15,\ 24)$$

Since only H_2^τ touches A_1 and G_2^τ, we use $A_1 \times G_2^\tau$ as the plant and H_2^τ for the specification to synthesize S_2^τ. The results are listed as follows:

$$A_1 \times G_2^\tau \ (985,\ 4053)\ ;\ H_2^\tau \ (17,\ 65)\ ;\ S_2^\tau \ (140,\ 288)$$

Next, we choose $\Sigma_{A_2} = \Sigma_{B_2}$ and compute the abstraction

$$A_2 := (A_1 \times G_2^\tau \times S_2^\tau)/\approx_{\Sigma_{A_2}} \ (15,\ 24)$$

We use $A_2 \times G_3^\tau$ as the plant and H_3^τ for the specification to synthesize S_3^τ. The results are as follows:

$$A_2 \times G_3^\tau \ (985,\ 4053)\ ;\ H_3^\tau \ (17,\ 65)\ ;\ S_3^\tau \ (140,\ 288)$$

Then we choose $\Sigma_{A_3} = \Sigma_{B_1}$ and compute the abstraction

$$A_3 := (A_2 \times G_3^\tau \times S_3^\tau)/\approx_{\Sigma_{A_3}} \ (15,\ 24)$$

Finally, we use $A_3 \times G_4^\tau$ as the plant and H_4^τ for the specification to synthesize S_4^τ, whose results are listed as follows:

$$A_3 \times G_4^\tau \ (253,\ 913)\ ;\ H_3^\tau \ (17,\ 65)\ ;\ S_4^\tau \ (68,\ 126)$$

By Theorem 3.10 we get that $\mathcal{S} = \{\nu(S_1^\tau), \nu(S_2^\tau), \nu(S_3^\tau), \nu(S_4^\tau)\}$ is the nonblocking distributed supervisor of the cluster tool system. After using our tool for nonconflict test [11], we confirm that $\mathcal{S}$ is nonconflicting with the overall plant model, which is the product of all components (i.e. load locks, robots, chambers and buffers). We can see that, the maximum size of automata in the above computation is (985, 4053), much smaller than the size of the product of all component models. Therefore, the proposed aggregative synthesis approach is computationally much more efficient than centralized synthesis. By looking at the sizes of A_1, A_2 and A_3, which happen to be the same owing to symmetry of the system model and our choice of the routing sequence, we can see how abstraction keeps the overall complexity low during aggregative synthesis.

5 Conclusions

In this paper we first present a distributed supervisory control problem. Then after introducing an algorithm PSNSNS for computing supremal nonblocking state-normal supervisors, we provide an aggregative synthesis procedure ASDS to derive nonblocking distributed supervisors, which solves that distributed supervisory control problem. By using an automaton abstraction technique, a large number of internal transitions at each synthesis stage are removed, which can help us avoid high complexity incurred by composition of automata. Although ASDS requires all automata to be standardized, we have shown in Theorem 4 that, by using a simple conversion procedure as indicated in ASNSDS, we can apply the same aggregative synthesis approach to synthesize distributed supervisors for systems modeled by non-standardized automata. Thus, the requirement of standardized automata in ASDS does not really impose any practical constraint on applications. Besides the potential computational advantage of aggregative synthesis, we can also achieve a certain degree of implementation flexibility in terms of attaining reusability of some local supervisors when the structure of a target system changes.

Acknowledgement: We would like to thank Dr. Albert T. Hofkamp of the Systems Engineering Group at Eindhoven University of Technology for coding all algorithms mentioned in this paper. We have used his code to generate the solution of the cluster tool example of Section IV.

1. Proof of Prop. 3.1: Let $G = (X, \Sigma, \xi, x_0, X_m)$ and $H = (Z, \Delta, \delta, z_0, Z_m)$. We first construct a new automaton $G_H = (U, \Sigma, \lambda, u_0, U_m)$, where

- $U = (X \times Z) \cup X$
- $U_m = X_m \times Z_m$
- $u_0 := (x_0, z_0)$
- $\lambda : U \times \Sigma \to 2^U$, where for any $(u, \sigma) \in U \times \Sigma$,

$$\lambda(u,\sigma) := \begin{cases} \xi \times \delta(u,\sigma) & \text{if } u = (x,z) \wedge \xi \times \delta(u,\sigma) \neq \varnothing \\ \xi(x,\sigma) & \text{if } u = (x,z) \wedge \xi(x,\sigma) \neq \varnothing \wedge \delta(z,\sigma) = \varnothing \vee u = x \in X \end{cases}$$

Some states in G_H may not be reachable. For slight abuse of notation, we use G_H to denote only the reachable sub-automaton. From G_H we construct another automaton $G_{NH} = (V, \Sigma, \gamma, v_0, V_m)$, where

- $V = U \times 2^U$
- $V_m = U_m \times 2^U$
- $v_0 = (u_0, \{u \in U | (\exists s \in \Sigma^*)\, u \in \lambda(u_0, s) \wedge P_o(s) = \epsilon\})$
- $\gamma : V \times \Sigma \to 2^V$, where for any $v = (u, U_u) \in V$ and $\sigma \in \Sigma$,

$$\gamma(v,\sigma) := \{(u', \{\hat{u} \in U | (\exists u'' \in U_u)(\exists s \in \Sigma^*)\, P_o(s) = P_o(\sigma) \wedge \hat{u} \in \lambda(u'', s)\}) | u' \in \lambda(u,\sigma)\}$$

Again, for slight abuse of notation we use G_{NH} to denote the reachable sub-automaton. We now present the following procedure:

1. Let $V^0 := \{(u, U_u) \in V | u \notin X\}$

2. For $k = 1, 2, \cdots$,

   (a) $\hat{V}^k := \{v \in V^{k-1} | (\forall s \in \Sigma_{uc}^*)\, \gamma(v, s) \subseteq V^{k-1} \wedge (\exists \sigma_1, \cdots, \sigma_n \in \Sigma)(\exists v_1, \cdots, v_n \in V^{k-1})\, v_1 \in \gamma(v, \sigma_1) \,\wedge\, v_n \in V_m \,\wedge\, (\forall i \in \{2, \cdots, n\})\, v_i \in \gamma(v_{i-1}, \sigma_i)\}$

   (b) $V^k := \{(u, U_u) \in \hat{V}^k | (\forall (u', U_{u'}) \in V - \hat{V}^k)\, U_u \neq U_{u'}\}$

   (c) Termination when $V^k = V^{k-1}$

We use $G_{NH}(V^k)$ to denote a sub-automaton of G_{NH}, whose state set is V^k and $V_m^k = V^k \cap V_m$. We claim that in PSNSNS, for each k, $N(S^k) = N(G_{NH}(V^k))$. To show this claim we use induction.
By the construction of G_{NH} we have $N(S^0) = N(G \times H) = N(G_{NH}(V^0))$. We assume that, up to $j \leq k-1$, $N(S^j) = N(G_{NH}(V^j))$, We will show that $N(S^k) = N(G_{NH}(V^k))$. Suppose it is not true. Then we have two cases to consider.
Case 1: $N(S^k) - N(G_{NH}(V^k)) \neq \varnothing$. Suppose $s \in N(S^k) - N(G_{NH}(V^k))$. Since $s \in N(S^k) \subseteq N(S^{k-1}) = N(G_{NH}(V^{k-1}))$, we have two subcases to consider.
Subcase 11: $\gamma(v_0, s) \cap \hat{V}^k = \varnothing$. Since $s \in N(G_{NH}(V^{k-1}))$, we get that there exists $v = (u, U_u) \in \gamma(v_0, s) \cap V^{k-1}$ such that there exist $s' \in \Sigma_{uc}^*$ and $v' = (u', U_{u'}) \in \gamma(v, s')$ such that $v' \notin V^{k-1}$. By the construction of V^{k-1}, we get that

$$ss' \notin \overline{N(G_{NH}(V^{k-1}))} = \overline{N(S^{k-1})} = \overline{N(S^{k-1} \times G)}$$

Since $s \in N(S^{k-1}) \subseteq \overline{N(S^{k-1})} = \overline{N(S^{k-1} \times G)}$ and $s' \in \Sigma_{uc}^*$, we can derive that $s \notin \overline{N(\chi(S^{k-1}, G))}$, which means $s \in N(\varpi(\chi(S^{k-1}, G)))$. Thus,

$$s \notin N(S^k) = N(\chi(S^{k-1}, G)) - P_o^{-1}(P_o(N(\varpi(\chi(S^{k-1}, G)))))$$

which contradicts the assumption that $s \in N(S^k)$. Thus, **Subcase 11** is not true.
Subcase 12: For any $(u, U_u) \in \gamma(v_0, s) \cap \hat{V}^k$, there exists $v' = (u', U_{u'}) \in V - \hat{V}^k$ such that $U_u = U_{u'}$. Thus, there exists $s' \in \overline{N(G_{NH}(V^{k-1}))}$ such that $(u', U_{u'}) \in \gamma(v_0, s')$ and $P_o(s) = P_o(s')$. There are two possibilities: (1) there exists $s'' \in \Sigma_{uc}^*$ such that $\gamma(v', s'') \nsubseteq V^{k-1}$. Then by the proof of **Subcase 11** we get that $s' \in N(\varpi(\chi(S^{k-1}, G)))$. Since $P_o(s) = P_o(s')$, we get that $s \notin N(S^k)$ - contradiction. (2) For any $s'' \in \Sigma^*$, if $\gamma(v', s'') \cap V_m \neq \varnothing$ then $s's'' \notin N(G_{NH}(V^{k-1}))$. Suppose $u' = (x, z)$ and let

$$\theta_{x,s'} := \{s's'' \in N(G \times H) | s'' \in \Sigma^*\}$$

Then $\theta_{x,s'} \cap N(G_{NH}(V^{k-1})) = \varnothing$, which means $\theta_{x,s'} \cap N(S^{k-1} \times G) = \varnothing$. Thus, $s' \in N(\varpi(\chi(S^{k-1}, G)))$. Since $P_o(s) = P_o(s')$, we get that

$$s \notin N(S^k) = N(\chi(S^{k-1}, G)) - P_o^{-1}(P_o(N(\varpi(\chi(S^{k-1}, G)))))$$

Again, we have a contradiction. Thus, **Subcase 12** does not hold, which means **Case 1** is not true.
Case 2: $N(G_{NH}(V^k)) - N(S^k) \neq \varnothing$. Suppose $s \in N(G_{NH}(V^k)) - N(S^k)$. Since $s \in N(G_{NH}(V^k)) \subseteq N(G_{NH}(V^{k-1})) = N(S^{k-1}) = N(S^{k-1} \times G)$ but $s \notin N(S^k)$, we have

$$s \notin N(\chi(S^{k-1}, G)) - P_o^{-1}(P_o(N(\varpi(\chi(S^{k-1}, G)))))$$

Thus, we have two subcases to consider.
Subcase 21: $s \notin N(\chi(S^{k-1}, G))$. Since $s \in N(S^{k-1}) = N(S^{k-1} \times G)$, we get that, there exist $(x, y) \in \xi \times \eta^{k-1}((x_0, y_0^{k-1}), s)$ and $s' \in \Sigma_{uc}^*$ such that $\xi(x, s') \neq \varnothing$ but $\eta^{k-1}(y, s') = \varnothing$. Thus, we get that $ss' \notin \overline{N(S^{k-1})} = \overline{N(G_{NH}(V^{k-1}))}$ with $s' \in \Sigma_{uc}^*$ and $ss' \in L(G_{NH}(V^0))$. Thus, there exists $v = (u, U_u) \in \gamma(v_0, s)$ such that $v \in V^{k-1}$ but $\gamma(v, s') \nsubseteq V^{k-1}$, which means $v \notin \hat{V}^k$. Therefore, $v \notin V^k$ because $V^k \subseteq \hat{V}^k$. So $s \notin \overline{N(G_{NH}(V^k))}$ - contradicting the assumption that $s \in N(G_{NH}(V^k))$. Thus, **Subcase**

21 does not hold.
Subcase 22: $s \in N(\chi(S^{k-1}, G)) \cap P_o^{-1}(P_o(N(\varpi(\chi(S^{k-1}, G)))))$. Since

$$s \in P_o^{-1}(P_o(N(\varpi(\chi(S^{k-1}, G)))))$$

there exists $s' \in N(\varpi(\chi(S^{k-1}, G)))$ such that $P_o(s') = P_o(s)$. By the construction of $\varpi(\chi(S^{k-1}, G))$, we get that there exist $t, t' \in \Sigma^*$, $x \in X$ and $\sigma \in \Sigma$ such that $s' = t\sigma t'$ and for any $t'' \leq t$, $(\xi \times \eta^{k-1}((x_0, y_0^{k-1}), t'')) \cap Q^{k-1} \neq \varnothing$, $x \in \xi(x_0, t)$, $\xi(x, \sigma) \neq \varnothing$ but $(\xi \times \eta^{k-1}((x_0, y_0^{k-1}), t\sigma)) \notin Q^{k-1}$. There are two possibilities: (1) $t\sigma \notin \overline{N(S^{k-1})} = \overline{N(G_{NH}(V^{k-1}))}$; (2) $t\sigma \in \overline{N(S^{k-1})}$. For (1) we can derive that, there exists $v \in \gamma(v_0, t\sigma)$ such that $v \notin V^{k-1}$, which means $v \notin \hat{V}^k$. Thus, $P_o^{-1}(P_o(t\sigma)) \cap \overline{N(G_{NH}(V^k))} = \varnothing$. Since $P_o(t\sigma) \leq P_o(s') = P_o(s)$, we get that $s \notin \overline{N(G_{NH}(V^k))}$ - contradicting to the assumption that $s \in N(G_{NH}(V^k))$. For (2), there exists $(x, y) \in \xi \times \eta^{k-1}((x_0, y_0^{k-1}), t) \cap Q^{k-1}$ such that there exists $(x', y') \in \xi \times \eta^{k-1}((x, y), \sigma)$ and $(x', y') \notin Q^{k-1}$. Since $(x', y') \notin Q^{k-1}$, we have two possibilities to consider: (2.1) there exists $t'' \in \Sigma_{uc}^*$ such that $\xi(x, t'') \neq \varnothing$ but $\eta^{k-1}(y, t'') = \varnothing$, which means $t\sigma t'' \notin \overline{N(S^{k-1})} = \overline{N(G_{NH}(V^{k-1}))}$ with $t'' \in \Sigma_{uc}^*$ and $t\sigma t'' \in L(G)$. Thus, there exists $v \in \gamma(v_0, t\sigma)$ such that $\gamma(v, t'') \nsubseteq V^{k-1}$, which means $v \notin \hat{V}^k$. Thus, we can derive that $s \notin N(G_{NH}(V^k))$ - contradiction. (2.2) For any $t'' \in \Sigma^*$, $\xi \times \eta^{k-1}((x', y'), t'') \cap (X_m \times Y_m^{k-1}) = \varnothing$. Therefore, there exists $v \in \gamma(v_0, t\sigma) \cap V^{k-1}$ such that, for any $t'' = \sigma_1 \cdots \sigma_n$ and $v_1, \cdots, v_n \in V$, if $v_1 = \gamma(v, \sigma_1)$, $v_n \in V_m$ and $v_i \in \gamma(v_{i-1}, \sigma_i)$ $(i = 2, \cdots, n)$, then there exists $v_j \notin V^{k-1}$. This means $v \notin \hat{V}^k$. Again, we can derive that $s \notin N(G_{NH}(V^k))$ - contradiction. Thus, **Subcase 22** does not hold, and so is **Case 2**.
Since in either case we derive a contradiction, it must be true that $N(S^k) = N(G_{NH}(V^k))$, and the induction is complete, which means the claim is true. Since V is finite, there must exist a $k \in \mathbb{N}$ such that $V^k = V^{k-1}$. Thus, from the claim we have $N(S^k) = N(S^{k-1})$, which means PSNSNS terminates no later than k. ■

2. Proof of Theorem 3.2: Suppose PSNSNS terminates at $k \in \mathbb{N}$. We first show that S is a nonblocking state-normal supervisor in (1). Then in (2) we show that S is supremal.
(1) Let $S = (Y, \Sigma, \eta, y_0, Y_m)$. By the definition of PSNSNS, we get that $N(S) = N(\chi(S^k, G))$. Since

$$N(\chi(S^k, G)) \subseteq N(\chi(S^{k-1}, G)) \subseteq \cdots \subseteq N(S^0) \subseteq N(G \times H)$$

we have $N(G \times S) = N(G) \cap N(S) \subseteq N(G) \cap N(G \times H) \subseteq N(G \times H)$.
To show that S is state-normal with respect to G and P_o, by Def. 2.6 we need to show that for any $s \in L(G \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G \times S)$, we have

$$(\forall (x, y) \in \xi \times \eta((x_0, y_0), s'))(\forall s'' \in \Sigma^*)\, P_o(s's'') = P_o(s) \Rightarrow [\xi(x, s'') \neq \varnothing \Rightarrow \eta(y, s'') \neq \varnothing]$$

Suppose it is not true. Then we get that, there exist $s \in L(G \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G \times S)$ such that

$$(\exists (x, y) \in \xi \times \eta((x_0, y_0), s'))(\exists s'' \in \Sigma^*)\, P_o(s's'') = P_o(s) \wedge \xi(x, s'') \neq \varnothing \wedge \eta(y, s'') = \varnothing$$

Let $s'' = t\sigma t'$, where $\eta(y, t) \neq \varnothing$ but $\eta(y, t\sigma) = \varnothing$ (such a σ must exist). Since S is a recognizer of $N(\chi(S^k, G))$ and $B(S) = \varnothing$, we have $s't \in L(\chi(S^k, G))$ but $s't\sigma \notin L(\chi(S^k, G))$. Because $s's'' \in L(G)$ we have $s't\sigma \in L(G)$.Thus, we have $s't \in L(S^k)$. Let $S^k = (Y^k, \Sigma, \eta^k, y_0^k, Y_m^k)$ and $\chi(S^k, G) = (Q^k, \Sigma, \varrho^k, q_0^k, Q_m^k)$. Suppose $y^k \in \eta^k(y_0^k, s't)$ and $x' \in \xi(x, t)$. There are two cases to consider. Case 1: $(x', y^k) \in Q^k$. Then since $\xi(x', \sigma) \neq \varnothing$ but $\varrho^k((x', y^k), \sigma) = \varnothing$, we get that $(x', y^k) \in \varphi(\chi(S^k, G))$. In $\varpi(\chi(S^k, G))$ we have $d \in \theta((x', y^k), \sigma)$. Thus, $s't\sigma \in N(\varpi(\chi(S, G)))$, and $s't\sigma t' \in N(\varpi(\chi(S, G)))$ as well. Since $P(s) = P(s's'') = P(s't\sigma t')$, we get that $s \in N(P_o^{-1}(P_o(N(\varpi(\chi(S^k, G))))))$. On the other hand, since $s \in L(G \times S)$, we get that $s \in N(\chi(S^k, G))$. Thus,

$$N(\chi(S^k, G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k, G)))))) \neq \varnothing$$

But this contradicts the assumption that $N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) = \varnothing$, because the procedure terminates at k. Case 2: $(x',y^k) \notin Q^k$. Clearly, $(x_0,y_0^k) \in Q^k$. Thus, there must exist $(x'',y''^k) \in \xi \times \eta^k((x_0,y_0^k),t'') \cap Q^k$ with $t''\sigma' \leq s't$ such that $\xi(x'',\sigma') \neq \varnothing$ but $\varrho^k((x'',y''^k),\sigma') = \varnothing$. Since $t''\sigma \leq s't \leq s''$, we can use the same argument as in case 1 to derive that

$$N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) \neq \varnothing$$

which contradicts our assumption that

$$N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) = \varnothing$$

Thus, in either case we get that, S must be state-normal with respect to G and P_o.
To show that S is state-controllable with respect to G and Σ_{uc}, by Def. 2.4 we need to show that

$$(\forall s \in L(G \times S))(\forall x \in \xi(x_0,s))(\forall y \in \eta(y_0,s))\, E_G(x) \cap \Sigma_{uc} \subseteq E_S(y)$$

Suppose it is not true. Then

$$(\exists s \in L(G \times S))(\exists x \in \xi(x_0,s))(\exists y \in \eta(y_0,s))\, E_G(x) \cap \Sigma_{uc} \nsubseteq E_S(y)$$

Since S is a recognizer of $\chi(S^k,G)$ and $B(S) = \varnothing$, we get that

$$s \in L(\chi(S^k,G)) \subseteq L(S^k)$$

Suppose $\eta^k(y_0^k,s) = \{y^k\}$ (because η is deterministic). Then we have two cases to consider. Case 1: $(x,y^k) \in Q^k$. Then we have $E_G(x) \cap \Sigma_{uc} \nsubseteq E_{S^k}(y^k)$, contradicting the definition of $\chi(S^k,G)$. Case 2: $(x,y^k) \notin Q^k$. Then there exist $s',s'' \in \Sigma^*$ and $\sigma \in \Sigma$ such that $s'\sigma s'' = s$, $x' \in \xi(x_0,s'\sigma)$, $x \in \xi(x',s'')$, $(x',q'^k) \in \varrho^k(q_0^k,s')$ but $\varrho((x',q'^k),\sigma) = \varnothing$. Since $s \in L(\chi(S^k,G))$, by using the argument as in proving state-normality, we can derive that

$$N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) \neq \varnothing$$

which contradicts the assumption that $N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) = \varnothing$. Thus, in either case, S is state-controllable with respect to G and Σ_{uc}.
Finally, we need to show that $B(G \times S) = \varnothing$. Suppose it is not true. Then

$$(\exists s \in \Sigma^*)(\exists (x,y) \in \xi \times \eta((x_0,y_0),s))(\forall s' \in \Sigma^*)\, \xi \times \eta((x,y),s') \cap (X_m \times Y_m) = \varnothing$$

Since S is a recognizer of $\chi(S^k,G)$ and $B(S) = \varnothing$, we get that $s \in L(\chi(S^k,G)) \subseteq L(S^k)$. Thus, there exists $y^k \in \eta^k(y_0^k,s)$. There are two cases to consider.
Case 1: $(x,y^k) \in Q^k$. from the fact that

$$(\forall s' \in \Sigma^*)\, \xi \times \eta((x,y),s') \cap (X_m \times Y_m) = \varnothing$$

and the fact that S is a recognizer of $\chi(S^k,G)$, we can derive that, for any $s' \in \Sigma^*$, $\varrho((x,y^k),s') \cap Q_m^k = \varnothing$. But this contradicts the definition of $\chi(S^k,G)$.
Case 2: $(x,y^k) \notin Q^k$. Then by using the same argument as proving Case 2 for state-controllability, we can derive that

$$N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) \neq \varnothing$$

which contradicts the assumption that $N(\chi(S^k,G)) \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^k,G)))))) = \varnothing$. Thus, in either case we have $B(G \times S) = \varnothing$.
(2) To show that S is the supremal nonblocking state-normal supervisor of G under H, suppose there is another supremal supervisor called S' with $N(S') \subseteq N(G)$. we will use induction to show that $N(S') \subseteq N(S^j)$ for each $j \in \mathbb{N}$. clearly, $N(S') \subseteq N(S^0)$. Suppose it is true that $N(S') \subseteq N(S^j)$, we need to show that $N(S') \subseteq N(S^{j+1})$. To this end we first make two claims
Claim 1: $N(S') \subseteq N(\chi(S^j,G))$. To show Claim 1, suppose it is not true. Then there exists $s \in N(S')$ but $s \notin N(\chi(S^j,G))$. Clearly, $s \in N(S^j)$ because $N(S') \subseteq N(S^j)$. Since S' is state-controllable with respect to G and Σ_{uc}, we get that

$$(\forall x \in \xi(x_0,s))(\forall y' \in \eta'(y_0',s))\, E_G(x) \cap \Sigma_{uc} \subseteq E_{S'}(y')$$

Since $N(S') \subseteq N(S^j)$ and both S' and S^j are deterministic, we have, for any $y^j \in \eta^j(y_0^j, s)$, $E_{S'}(y') \subseteq E_{S^j}(y^j)$. Thus, $E_G(x) \cap \Sigma_{uc} \subseteq E_{S^j}(y^j)$, which means $(x, y^j) \in Q^j$. Thus, $s \in N(\chi(S^j, G))$, contradicting the assumption that $s \notin N(\chi(S^j, G))$. Therefore, the claim is true.
Claim 2: $N(S') \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S^j, G)))))) = \varnothing$. To show such a claim, notice that $N(S') \subseteq N(S^j)$ and both S' and S^j are deterministic. Thus,

$$N(\varpi(\chi(S^j, G))) \subseteq N(\varpi(\chi(S', G)))$$

Since S' is state-normal with respect to G and Σ_{uo}, we have

$$N(S') \cap N(P_o^{-1}(P_o(N(\varpi(\chi(S', G)))))) = \varnothing$$

Therefore, the claim is true.
From Claims 1 and 2 we have

$$N(s') \subseteq N(\chi(S^j, G)) - N(P_o^{-1}(P_o(N(\varpi(\chi(S^j, G)))))) = N(S^{j+1})$$

In particular, $N(S') \subseteq N(\chi(S^k, G)) = N(S)$. Since S has been shown to be a nonblocking state-normal supervisor of G under H, we get that S is actually the supremal nonblocking state-normal supervisor of G under H. ■

3. Proof of Theorem 3.6: We first show that $N(G_1 \times G_2 \times S_1 \times S_2) \subseteq N(G_1 \times G_2 \times H_1 \times H_2)$. To this end let $P_1' : \Sigma_1^* \to \Sigma'^*$ be the natural projection. We have

$$\begin{aligned}
& N(((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \times S_2) \subseteq N(((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \times H_2) \\
& \quad \text{because } S_2 \text{ is a nonblocking supervisor of } ((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \text{ under } H_2 \\
\Rightarrow\ & N((G_1 \times S_1)/\approx_{\Sigma'})||N(G_2 \times S_2) \subseteq N((G_1 \times S_1)/\approx_{\Sigma'})||N(G_2 \times H_2) \\
\Rightarrow\ & P_1'(N(G_1 \times S_1))||N(G_2 \times S_2)) \subseteq P_1'(N(G_1 \times S_1))||N(G_2 \times H_2) \text{ by Prop. 3.3} \\
\Rightarrow\ & N(G_1 \times S_1)||P_1'(N(G_1 \times S_1))||N(G_2 \times S_2)) \subseteq N(G_1 \times S_1)||P_1'(N(G_1 \times S_1))||N(G_2 \times H_2) \\
\Rightarrow\ & N(G_1 \times S_1)||N(G_2 \times S_2)) \subseteq N(G_1 \times H_1)||N(G_2 \times H_2)) \\
& \quad \text{because } N(G_1 \times S_1) = N(G_1 \times S_1)||P_1'(N(G_1 \times S_1)) \text{ and } N(G_1 \times S_1) \subseteq N(G_1 \times H_1) \\
\Rightarrow\ & N(G_1 \times G_2 \times S_1 \times S_2) \subseteq N(G_1 \times G_2 \times H_1 \times H_2)
\end{aligned}$$

Next, we show that $B(G_1 \times G_2 \times S_1 \times S_2) = \varnothing$. To this end let $P' : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma'^*$ be the natural projection. We have

$$\begin{aligned}
& B(((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \times S_2) = \varnothing \\
& \quad \text{Since } S_2 \text{ is a nonblocking supervisor of } ((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \text{ under } H_2 \\
\Rightarrow\ & B(((G_1 \times S_1)/\approx_{\Sigma'}) \times ((G_2 \times S_2)/\approx_{\Sigma_2 \cup \Sigma'})) = \varnothing \\
& \quad \text{because } G_2 \times S_2 \cong (G_2 \times S_2)/\approx_{\Sigma_2 \cup \Sigma'} \text{ and by Prop. 3.5} \\
\Rightarrow\ & B((G_1 \times S_1 \times G_2 \times S_2)/\approx_{\Sigma_2 \cup \Sigma'}) = \varnothing \text{ because } (\Sigma_2 \cup \Sigma') \cap \Sigma_1 = \Sigma' \text{ and Prop. 3.4} \\
\Rightarrow\ & P'(B(G_1 \times G_2 \times S_1 \times S_2)) = \varnothing \text{ by Prop. 3.3} \\
\Rightarrow\ & B(G_1 \times G_2 \times S_1 \times S_2) = \varnothing \text{ by the property of natural projection } P'
\end{aligned}$$

We now show that $S_1 \times S_2$ is state-controllable with respect to $G_1 \times G_2$ and $\Sigma_{1,uc} \cup \Sigma_{2,uc}$. Suppose it is not true. Then by Def. 2.4, there exist $s \in L(G_1 \times G_2 \times S_1 \times S_2)$, $(x_1, x_2) \in \xi_1 \times \xi_2((x_{1,0}, x_{2,0}), s)$ and $(y_1, y_2) \in \eta_1 \times \eta_2((y_{1,0}, y_{2,0}), s)$ such that

$$E_{G_1 \times G_2}(x_1, x_2) \cap (\Sigma_{1,uc} \cup \Sigma_{2,uc}) \not\subseteq E_{S_1 \times S_2}(y_1, y_2)$$

which means

$$(\exists \sigma \in \Sigma_{1,uc} \cup \Sigma_{2,uc})\, \xi_1 \times \xi_2((x_1, x_2), \sigma) \neq \varnothing \wedge \eta_1 \times \eta_2((y_1, y_2), \sigma) = \varnothing$$

There are two cases to consider.
Case 1: $\sigma \in \Sigma_{1,uc} - (\Sigma' \cup \Sigma_{2,uc})$. By the assumption (A1) we get that $\sigma \in \Sigma_1 - (\Sigma_2 \cup \Sigma')$. Since $\eta_1 \times \eta_2((y_1, y_2), \sigma) = \varnothing$, we get that $\eta_1(y_1, \sigma) = \varnothing$. Clearly, $\xi_1(x_1, \sigma) \neq \varnothing$. Thus, we get $E_{G_1}(x_1) \cap \Sigma_{1,uc} \not\subseteq E_{S_1}(y_1)$, contradicting the assumption that S_1 is state-controllable

with respect to G_1 and $\Sigma_{1,uc}$.
Case 2: $\sigma \in (\Sigma_{1,uc} \cap \Sigma') \cup \Sigma_{2,uc}$. Clearly, $\eta_1(y_1, \sigma) \neq \varnothing$ because, otherwise, as proved in Case 1, S_1 is not state-controllable with respect to G_1 and $\Sigma_{1,uc}$. Since $\eta_1 \times \eta_2((y_1, y_2), \sigma) = \varnothing$, we get that $\eta_2(y_2, \sigma) = \varnothing$. Clearly, $\xi_1 \times \eta_1((x_1, y_1), \sigma) \neq \varnothing$. Let $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ be the natural projection, and ξ' be the transition map of $(G_1 \times S_1)/\approx_{\Sigma'}$. Since both G_1 and S_1 are standardized, by Def. 2.2 we get that

$$< x_1, y_1 > \in \xi'(< x_{1,0}, y_{1,0} >, P_1(s)) \ \wedge \ \xi'(< x_1, y_1 >, \sigma) \neq \varnothing$$

Thus, we have

$$(< x_1, y_1 >, x_2) \in \xi' \times \xi_2((< x_{1,0}, y_{1,0} >, x_{2,0}), s) \ \wedge \ \xi' \times \xi_2((< x_1, y_1 >, x_2), \sigma) \neq \varnothing$$

from which we get that

$$E_{((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2}(< x_1, y_1 >, x_2) \cap ((\Sigma_{1,uc} \cup \Sigma_{2,uc}) \cap \Sigma') \nsubseteq E_{S_2}(y_2)$$

But this contradicts that S_2 is state-controllable with respect to $((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2$ and $(\Sigma_{1,uc} \cup \Sigma_{2,uc}) \cap \Sigma'$. Thus, in either case we have $S_1 \times S_2$ is state-controllable with respect to $G_1 \times G_2$ and $\Sigma_{1,uc} \cup \Sigma_{2,uc}$.
Finally, we show that $S_1 \times S_2$ is state-normal with respect to $G_1 \times G_2$ and the natural projection $P_o : (\Sigma_1 \cup \Sigma_2)^* \to (\Sigma_{1,o} \cup \Sigma_{2,o})^*$. Suppose it is not true. Then by Def. 2.6 we get that there exist $s \in L(G_1 \times G_2 \times S_1 \times S_2)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G_1 \times G_2 \times S_1 \times S_2)$ such that there exist $(x_1, x_2, y_1, y_2) \in \xi_1 \times \xi_2 \times \eta_1 \times \eta_2((x_{1,0}, x_{2,0}, y_{1,0}, y_{2,0}), s')$ and $s'' \in (\Sigma_1 \cup \Sigma_2)^*$ with

$$P_o(s's'') = P_o(s) \ \wedge \ \xi_1 \times \xi_2((x_1, x_2), s'') \neq \varnothing \ \wedge \ \eta_1 \times \eta_2((y_1, y_2), s'') = \varnothing$$

Clearly, there exist $t, t' \in (\Sigma_1 \cup \Sigma_2)^*$ and $\sigma \in \Sigma_1 \cup \Sigma_2$ such that $s'' = t\sigma t'$ and

$$\eta_1 \times \eta_2((y_1, y_2), t) \neq \varnothing \ \wedge \ \eta_1 \times \eta_2((y_1, y_2), t\sigma) = \varnothing$$

There are two cases to consider.
Case 1: $\sigma \in \Sigma_1 - (\Sigma_2 \cup \Sigma')$. From $\eta_1 \times \eta_2((y_1, y_2), t\sigma) = \varnothing$ we get that $\eta_1(y_1, t\sigma) = \varnothing$. Let $P_{1,o} : \Sigma_1^* \to \Sigma_{1,o}^*$ be the natural projection. By the assumption (A1), we have $P_1 \circ P_o = P_{1,o} \circ P_1$. Thus, we get $P_1(s) \in L(G_1)$ and $P_1(s') \in \overline{P_{1,o}^{-1}(P_{1,o}(P_1(s)))} \cap L(G_1)$ such that $(x_1, y_1) \in \xi_1 \times \eta_1((x_{1,0}, y_{1,0}), P_1(s'))$ and $P_1(s'') \in \Sigma_1^*$ with

$$P_{1,o}(P_1(s's'')) = P_{1,o}(P_1(s)) \ \wedge \ \xi_1(x_1, P_1(s'')) \neq \varnothing \ \wedge \ \eta_1(y_1, P_1(s'')) = \varnothing$$

which contradicts the assumption that S_1 is state-normal with respect to G_1 and $P_{1,o}$.
Case 2: $\sigma \in \Sigma_2 \cup \Sigma'$. Clearly, $\eta_1(y_1, t\sigma) \neq \varnothing$ because, otherwise, as proved in Case 1, S_1 is not state-normal with respect to G_1 and $P_{1,o}$. So from $\eta_1 \times \eta_2((y_1, y_2), t\sigma) = \varnothing$ we have $\eta_2(y_2, t\sigma) = \varnothing$. Let

$$P_2 : (\Sigma_1 \cup \Sigma_2)^* \to (\Sigma_2 \cup \Sigma')^* \text{ and } P_{2,o} : (\Sigma_2 \cup \Sigma')^* \to ((\Sigma_2 \cup \Sigma') \cap (\Sigma_{1,o} \cup \Sigma_{2,o}))^*$$

be the natural projection. By the assumption (A1), we have $P_2 \circ P_o = P_{2,o} \circ P_2$. Since all automata are standardized and $s \in L(G_1 \times G_2 \times S_1 \times S_2)$, we get that

$$P_2(s) \in L(((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \times S_2)$$

Use a similar way we can derive that

$$P_2(s') \in \overline{P_{2,o}^{-1}(P_{2,o}(P_2(s)))} \cap L(((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2 \times S_2)$$

and $(< x_1, y_1 >, x_2, y_2) \in \xi' \times \xi_2 \times \eta_2((< x_{1,0}, y_{1,0} >, x_{2,0}, y_{2,0}), P_2(s'))$ and $P_2(s'') \in (\Sigma_2 \cup \Sigma')^*$ with

$$P_{2,o}(s's'') = P_{2,o}(s) \ \wedge \ \xi' \times \xi_2((< x_1, y_1 >, x_2), P_2(s'')) \neq \varnothing \ \wedge \ \eta_2(y_2, P_2(s'')) = \varnothing$$

But this contradicts that S_2 is state-normal with respect to $((G_1 \times S_1)/\approx_{\Sigma'}) \times G_2$ and $P_{2,o}$. Thus, in either case we have $S_1 \times S_2$ is state-normal with respect to $G_1 \times G_2$ and P_o.■

Bibliography

[1] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event systems. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.

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

[3] K.C. Wong and W.M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.

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

[5] C. Ma and W.M. Wonham. Nonblocking supervisory control of state tree structures. *IEEE Trans. Automatic Control*, 51(5):782–793, 2006.

[6] L. Feng and W.M. Wonham. Computationally efficient supervisor design: modularity and abstraction. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 3–8, 2006.

[7] K. Schmidt, H. Marchand and B. Gaudin. Modular and decentralized supervisory control of concurrent discrete event systems using reduced system models. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 149–154, 2006.

[8] R. Su and J.G. Thistle. A distributed supervisor synthesis approach based on weak bisimulation. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 64–69, 2006.

[9] W. M. Wonham. *Supervisory Control of Discrete-Event Systems.* Systems Control Group, Dept. of ECE, University of Toronto. URL: www.control.utoronto.ca/DES, July 1, 2007.

[10] R. Su, J.H. van Schuppen and J.E. Rooda. *Model abstraction of nondeterministic finite state automata in supervisor synthesis.* Submitted to IEEE Trans. Automatic Control, November, 2008. It also appears in SE Technical Report No. 2008-3, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Techonology, Eindhoven, The Netherlands, 2008. ISSN 1567-1872. URL: http://se.wtb.tue.nl/sereports.

[11] R. Su, J.H. van Schuppen, J.E. Rooda and A.T. Hofkamp. *Nonconflict check by using sequential automaton abstractions.* Submitted to Automatica, November, 2008. It also appears in SE Technical Report No. 2008-10, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Techonology, Eindhoven, The Netherlands, 2008. ISSN 1567-1872. URL: http://se.wtb.tue.nl/sereports.

[12] M. Fabian and B. Lennartson. On non-deterministic supervisory control. In *Proc. 35th IEEE Conference on Decision and Control*, pages 2213–2218, 1996.

[13] H. Flordal and R. Malik. Modular nonblocking verification using conflict equivalence. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 100–106, 2006.

[14] H. Flordal, R. Malik, M. Fabian and K. Akesson. Compositional synthesis of maximally permissive supervisors using supervisor equivalence. In *Discrete Event Dynamic Systems*, 17(4):475-504, 2007.

[15] R. Malik and H. Flordal. Yet another approach to compositional synthesis of discrete event systems. In *Proc. 9th International Workshop on Discrete Event Systems (WODES08)*, pages 16–21, 2008.

[16] R.C. Hill, D.M. Tilbury and S. Lafortune. Modular supervisory control with equivalence-based conflict resolution. In *Proc. 2008 American Control Conference (ACC08)*, pages 491–498, 2008.

[17] R. Milner. Operational and algebraic semantics of concurrent processes. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pp. 1201-1242, MIT Press, 1990

[18] A. Overkamp. Supervisory control using failure semantics and partial specifications. *IEEE Trans. Automatic Control*, 42(4):498-510, 1997.

[19] K. Schmidt and C. Breindl. On maximal permissiveness of hierarchical and modular supervisory control approaches for discrete event systems. In *Proc. 9th International Workshop on Discrete Event Systems (WODES08)*, pages 462–467, 2006.

[20] R.C. Hill, D.M. Tilbury and S. Lafortune. Modular supervisory control with equivalence-based conflict resolution. In *Proc. 2008 American Control Conference (ACC08)*, pages 491–498, 2008.

[21] R. Hill and D. Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 399–406, 2006.

[22] H. Flordal and R. Malik. Modular nonblocking verification using conflict equivalence. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 100–106, 2006.

[23] P.N. Pena, J.E.R. Cury and S. Lafortune. Testing modularity of local supervisors: an approach based on abstractions. In *Proc. 8th International Workshop on Discrete Event Systems (WODES06)*, pages 107–112, 2006.

[24] R. Su and W.M. Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14(1):31-53, 2004

[25] R. Su and W.M. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Trans. Automatic Control*, 50(12):1923-1935, 2005

[26] J. Yi, S. Ding, M.T. Zhang, M.T. and P. van der Meulen. Throughput analysis of linear cluster tools. In *proc. 3rd IEEE International Conference on Automation Science and Engineering (CASE2007)*, pages 1063-1068, 2007.

[27] J.C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2-3): 219-236, 1990