Coordinated distributed supervisory control

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. 2009-02

Coordinated Distributed Supervisory Control

Rong Su, Jan H. van Schuppen and Jacobus E. Rooda[1] [2]

ISSN: 1872-1567

SE Report: Nr. 2009-02
Eindhoven, March 2009
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

In supervisor synthesis achieving nonblockingness is a major computational challenge when a target system consists of a large number of local components. To overcome this difficulty we propose a coordinated distributed supervisor synthesis approach, where specifications are enforced by local supervisors. To avoid conflicting among local supervisors, coordinators are created based on automaton abstraction.

1 Introduction

In the Ramadge/Wonham supervisory control paradigm [1] [2] one of the main challenges of 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 difficulty, many approaches have been proposed recently, e.g. state-feedback control based on state-tree structures [5], hierarchical interface-based control [4] and modular/distributed control [6] [8] [20] [18] [15] [19].

The modular/distributed approaches are particular interesting for two reasons: potentially low synthesis complexity and high implementation flexibility. The low complexity is achieved through local synthesis by using appropriate abstraction, and implementation flexibility refers that, a structural change of the target system may result in only a small number of relevant local controllers to be updated. Currently there are two major types of abstraction techniques: language-based abstraction, e.g. [6] [20] [7] [18], and automaton-based abstraction, e.g. [8] [21] [14] [15] [19] [11]. The language-based abstraction techniques rely on a special type of natural projections called *observers* [3], which is crucial for achieving nonblockingness. The shortcoming of using observers is that, the codomain of a natural projection needs to be sufficiently large so that the observer property can be obtained. The consequence is that, an abstracted model may not be small enough for subsequent modular/distributed synthesis. The automaton-based abstraction techniques do not have any special requirement on the codomain of abstraction maps. But in general they create nondeterministic abstracted models, even when the original models are deterministic. This forces a user to deal with supervisor synthesis for nondeterministic systems. Fortunately, when specifications are deterministic, such synthesis can be easily performed [11]. Among existent automaton abstraction techniques, [8] requires an abstracted model weakly bisimilar to the original model. [21] [19] are aimed for conflict equivalence, [14] for supervision equivalence and [15] for synthesis equivalence. All of these approaches require to use heuristic rewriting rules and silent events in order to preserve appropriate equivalence relations.

In [10] a new automaton abstraction technique is introduced, which is applied in aggregative synthesis proposed in [11]. The advantage of this new technique is that, no silent event is required and the construction is much simpler than using heuristic rewriting rules. An extension is made in [12] to guarantee that abstraction will not create extra blocking behaviors that may happen by the original technique in [10] [11]. In this paper our main contribution is to develop a procedure to synthesize nonblocking coordinated distributed supervisors, based on the abstraction technique proposed in [12]. The coordination strategy is similar to those mentioned in [21] [18] [19], except that we use a different abstraction technique. The distributed supervisory control problem setup is close to the one used in [11], except that the latter one is aimed for aggregative synthesis in the sense that, a new local supervisor is constructed based on relevant local components and previously constructed local supervisors. As a contrast, in this paper all local supervisors are constructed at the same time, then relevant coordinators are bult to solve potential conflicts among local supervisors. The advantage of this approach is that, when the system's architecture is changed, only a few relevant local supervisors and coordinators are required to be updated. But in aggregative synthesis proposed in [11], all local components are ordered in a list, and a change of a local component, say G_i, will force all local supervisors associated with local components after G_i in the list to be updated. When G_i is at the beginning of the list, such a change will be equivalent to recomputing the entire distributed supervisor. Thus, the coordinated distributed synthesis proposed in this paper enjoys computational and implementation advantages over the aggregative synthesis

proposed in [11]. Nevertheless, the aggregative synthesis may generate a distributed supervisor more permissive than the one generated by the coordinated distributed synthesis.

This paper is organized as follows. In Section II we first review relevant concepts and automaton operations. Then in Section III we put forward a distributed supervisory control problem, and present a coordinated distributed synthesis approach based on abstractions of nondeterministic automata. As an illustration, the proposed synthesis approach is applied to a cable TV service network in Section IV. Conclusions are stated in Section V. All long proofs are presented in the Appendix.

2 Preliminaries on Languages and Nondeterministic Finite-state Automata

In this section we first review basic concepts of languages and nondeterministic finite-state automata. Then we present a few results that will be used in synthesis.

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 [9],

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 $s' \in \Sigma^*$, $\xi(x, s') \cap X_m = \varnothing$. Such a state x is called a *blocking state* of G, and we call $B(G)$ the *blocking set*. A state that is not a blocking state is called a *nonblocking state*. 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 $N(G_1 \times G_2) = N(G_1) || N(G_2)$, due to the marked states of $G_1 \times G_2$ being $X_{m,1} \times X_{m,2}$. 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 $R \subseteq X \times X$ such that, $R \subseteq \{(x, x') \in X \times X | x \in X_m \iff x' \in X_m\}$ and

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

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 [16], 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 [23]. 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) $B(G_1) \subseteq B(G_2)$, (2) $N(G_1) = N(G_2)$, and (3) for all $s \in \overline{N(G_1)}$, $x_1 \in \xi_1(x_{1,0}, s)$, 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.

To use the proposed automaton abstraction properly, we need to introduce the concept of *standardized automata*, which is defined as follows.

We bring in two new symbols $\tau, \mu \notin \Sigma$, and call $G^{\tau,\mu} = (X, \Sigma \cup \{\tau, \mu\}, \xi, x_0, X_m)$ *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. $(\forall x \in X)(\forall \sigma \in \Sigma \cup \{\tau\})\, x_0 \notin \xi(x, \sigma)$
3. $(\forall x \in X)\, x \in X_m \Rightarrow x \in \xi(x, \mu)$

A standardized automaton is nothing but an automaton, in which x_0 is not marked, τ is only defined at x_0, which has only outgoing τ transitions and no incoming transitions, and μ is selflooped at every marker state. We can consider μ as a marking event, which marks every marker state. The importance of introducing the notion of standardized automaton is explain in details in [12]. Briefly speaking, it guarantees that the nonblockingness of an abstraction implies the nonblockingness of the original automaton, and vice versa.

If an automaton is not standardized, such a property may not hold when we applied the proposed automaton abstraction. It has been shown in [12] that, the abstraction of a standardized automaton is a standardized one, and the product of two standardized automata is also a standardized one. Although it looks like we are restricting ourselves to a special type of automata, it has been explained in [11] that, the notion τ does not put any constraint on supervisor synthesis based on abstractions and we will also explain later in this paper that the notion μ does not impose any restriction as well. From now on, unless specified explicitly, we assume that each alphabet Σ contains τ and μ, and $\phi(\Sigma)$ is the collection of all standardized finite-state automata, whose alphabet is Σ. By a slight abuse of notation, we use G to denote a standardized automaton $G^{\tau,\mu}$. We have the following result, which is useful in distributed synthesis.

Proposition 2.4. Suppose we have a collection of alphabets $\{\Sigma_i | i \in I\}$ for some index I, and a collection of components $\{G_i \in \phi(\Sigma_i) | i \in I\}$. Let $\Sigma' \subseteq \cup_{i\in I}\Sigma_i$ such that $\cup_{i,j\in I:i\neq j}\Sigma_i \cap \Sigma_j \subseteq \Sigma'$. Then $(\times_{i\in I}G_i)/\approx_{\Sigma'} \cong \times_{i\in I}(G_i/\approx_{\Sigma_i\cap\Sigma'})$ □

Proof: We use induction on the size of I. When $|I| = 2$, by Prop. 5 in [12], the result holds. Suppose it holds for $|I| = n$. We show that it also holds for $|I| = n+1$ as follows:

$$
\begin{aligned}
& (\times_{i\in I}G_i)/\approx_{\Sigma'} \\
= \;& (\times_{i\in I-\{j\}}G_i \times G_j)/\approx_{\Sigma'} \\
\cong \;& ((\times_{i\in I-\{j\}}G_i)/\approx_{(\cup_{i\in I-\{j\}}\Sigma_i)\cap\Sigma'}) \times (G_j/\approx_{\Sigma_j\cap\Sigma'}) \\
& \text{since } \Sigma_j \cap (\cup_{i\in I-\{j\}}\Sigma_i) \subseteq \Sigma' \text{ and by Prop. 5 in [12]} \\
\cong \;& \times_{i\in I-\{j\}}(G_i/\approx_{\Sigma_i\cap\Sigma'}) \times (G_j/\approx_{\Sigma_j\cap\Sigma'}) \\
& \text{because } |I-\{j\}| = n \text{ and by the induction hypothesis and Prop. 2 in [12]} \\
= \;& \times_{i\in I}(G_i/\approx_{\Sigma_i\cap\Sigma'})
\end{aligned}
$$

Thus, the proposition is true. ■

In control engineering examples G usually consists of a large number of small automata, namely $G = G_1 \times \cdots \times G_n$ for some very large number $n \in \mathbb{N}$, where $G_i \in \phi(\Sigma_i)$ for each $i = 1, 2, \cdots, n$. How to compute $G/\approx_{\Sigma'}$ imposes a great computational difficulty. To overcome it, we propose the following algorithm. Let $I = \{1, \cdots, n\}$ for some $n \in \mathbb{N}$. For any $J \subseteq I$, let $\Sigma_J := \cup_{j\in J}\Sigma_j$.

Sequential Abstraction over Product: (SAP)
(1) Inputs of SAP: a collection $\{G_i \in \phi(\Sigma_i) | i \in I\}$ and an alphabet $\Sigma' \subseteq \cup_{i\in I}\Sigma_i$ with $\tau, \mu \in \Sigma'$.
(2) For $k = 1, 2, \cdots, n$, we perform the following computation.

- Set $J_k := \{1, 2, \cdots, k\}$, $T_k := \Sigma_{J_k} \cap (\Sigma_{I-J_k} \cup \Sigma')$.
- If $k = 1$ then $W_1 := G_1/\approx_{T_1}$
- If $k > 1$ then $W_k := (W_{k-1} \times G_k)/\approx_{T_k}$

(3) Output of SAP: W_n □

Proposition 2.5. *[12]* Suppose W_n is computed by SAP. Then $(\times_{i\in I}G_i)/\approx_{\Sigma'} \cong W_n$. □

SAP allows us to obtain an abstraction of $G = \times_{i\in I} G_i$ in a sequential way. Thus, we can avoid computing G explicitly, which may be prohibitively large for systems of industrial size. Next, we discuss how to perform distributed supervisor synthesis.

3 Synthesis of Coordinated Distributed Supervisors

3.1 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. In particular, $\tau \in \Sigma_{uc}$ and $\mu \in \Sigma_c$. Let $L(G) := \{s \in \Sigma^* | \xi(x_0, s) \neq \varnothing\}$.

Definition 3.1. Given $G = (X, \Sigma, \xi, x_0, X_m)$ 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 reverse 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. In particular, $\tau, \mu \in \Sigma_{uo}$. Let $P_o : \Sigma^* \to \Sigma_o^*$ be the natural projection.

Definition 3.2. Given $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$ 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. 3.2 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. $P_o(s) = P_o(s')$), 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 reverse 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 3.3. Given $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$ 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

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

□

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 reverse statement is not true unless both A and G are deterministic. Furthermore, we can check that state normality implies state observability. But the reverse statement is not true. We now introduce the concept of supervisor.

Definition 3.4. 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. 3.4 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. [17]. 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. From Prop. 4 in [10] we get 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 NSCSN supervisor. A computational procedure for supremal NSCSN is provided in [11]. We now present the concept of distributed systems.

Definition 3.5. A *distributed system* with respect to given alphabets $\{\Sigma_i | i \in I\}$ is a collection 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. The *compositional behavior* of $\mathcal{G}$ is specified by $\times_{i\in I} G_i$. □

The product of local components is the system of interest. Interaction among local components is modeled by event sharing among local components. 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 distributed 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 synthesizing coordinated distributed supervisors. We now present a statement of a control problem.

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 w.r.t. $\times_{i\in I} G_i$ and $\cup_{i\in I}\Sigma_{i,uc}$
4. $\times_{k\in K} S_k$ is state-normal w.r.t. $\times_{i\in I} G_i$ 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 $\mathcal{H}$. There are many ways to compute a nonblocking distributed supervisor. For example, in [11] an aggregative synthesis approach is proposed. In this paper we will present a synthesis approach that computes in parallel a set of local supervisors to take care of local specifications, then compute one or several coordinators to solve potential conflict among local supervisors. We call such a supervisor as a *coordinated distributed supervisor*. Next, we discuss how to synthesize nonblocking coordinated distributed supervisors.

3.2 Synthesis of Coordinated Distributed Supervisors

Given a distributed system $\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I = \{1, 2, \cdots, n\} \ \wedge \ n \in \mathbb{N}\}$, suppose each local component G_i $(i \in I)$ has its deterministic local specification $H_i \in \phi(\Delta_i)$,

where $\Delta_i \subseteq \Sigma_i$. Furthermore, there is one deterministic specification $H \in \phi(\Delta)$, where $\Delta \subseteq \cup_{i\in I}\Sigma_i$. We would like to synthesize a nonblocking distributed supervisor $\mathcal{S}$ of $\mathcal{G}$ under $\{H, H_i | i \in I\}$. To this end, we need the following result.

Proposition 3.6. Let $G_1, G_2 \in \phi(\Sigma)$ and $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$. Suppose $G_1 \sqsubseteq G_2$. Then a nonblocking state-observable (or state-normal) supervisor $S \in \phi(\Sigma)$ of G_2 under H is also a nonblocking state-observable (or state-normal) supervisor of G_1 under H. □

The proof of Prop. 3.6 is provided in the Appendix. The proposition says that, if a plant G_1 is nonblocking preserving with respect to G_2, then a nonblocking supervisor for G_2 is also a nonblocking supervisor for G_1. In many cases it may be easier to obtain G_2 than G_1. For example, it is easier to use SAP to compute an abstraction, than simply compute the product first then perform the abstraction operation on the product. Prop. 3.6 is used in the following main result.

Theorem 3.7. Suppose for each G_i we have a nonblocking state-observable (or state-normal) supervisor $S_i \in \phi(\Sigma_i)$ under H_i. Let $\Sigma' \subseteq \cup_{i\in I}\Sigma_i$ such that $\cup_{i,j:i\neq j}\Sigma_i \cap \Sigma_j \subseteq \Sigma'$ and $\Delta \subseteq \Sigma'$. For each $i \in I$ suppose we have $W_i \in \phi(\Sigma_i \cap \Sigma')$ such that $(G_i \times S_i)/\approx_{\Sigma_i\cap\Sigma'} \sqsubseteq W_i$. Let $S = (Y, \Sigma', \eta, y_0, Y_m) \in \phi(\Sigma')$ be a nonblocking state-observable (or state-normal) supervisor of $\times_{i\in I} W_i$ under H. Then $S \times_{i\in I} S_i$ is a nonblocking state-observable (or state-normal) supervisor of $\times_{i\in I} G_i$ under $H \times_{i\in I} H_i$. □

The proof of Theorem 3.7 is provided in the Appendix. What Theorem 3.7 says is that, we can synthesize a local supervisor S_i for each component G_i so that the local specification H_i can be enforced. Then we compute an abstraction so that we can synthesize a local supervisor to take care of H. In practical applications sometimes a specification, say H_i, may cover several local components, say $\{G_{il} \in \phi(\Sigma_{il}) | l = 1, \cdots, r\}$, in the sense that, $\Delta_i \subseteq \cup_{l=1}^{r}\Sigma_{ij}$. In this case, we can compute $G_i := \times_{l=1}^{r} G_{il}$ and treat it as a local component so that H_i is defined for G_i. Thus, the setup in Theorem 3.7 is general enough. The reason that we bring in W_i in Theorem 3.7 is because, when G_i actually consists of many small components, e.g. $\{G_{il} \in \phi(\Sigma_{il}) | l = 1, \cdots, r\}$, computing $(G_i \times S_i)/\approx_{\Sigma_i\cap\Sigma'}$ may be feasible only through a sequential procedure, e.g. using the SAP. In that case, the outcome of that procedure may not be exactly equal to $(G_i \times S_i)/\approx_{\Sigma_i\cap\Sigma'}$. The theorem says that, as long as $(G_i \times S_i)/\approx_{\Sigma_i\cap\Sigma'}$ is nonblocking preserving with respect to W_i, which is computed by an appropriate procedure, e.g. the SAP, then synthesizing a local supervisor based on $\{W_i | i \in I\}$ will result in a nonblocking supervisor for the original local components. In Theorem 3.7 we call each S_i a *local supervisor* of $\mathcal{G}$ and S a *coordinator* of $\mathcal{G}$, which is mainly used to coordinate local supervisors $\{S_i | i \in I\}$ to avoid conflict. The existence of S gives rise to the term *coordinated distributed supervisor*. Of course, S itself is a supervisor, which enforces the specification H. Theorem 3.7 allows us to synthesize a multiple-level multiple-coordinator distributed supervisor. For example, the system in Theorem 3.7 may be only a single module of a large system. Thus, after obtaining $\{S_i | i \in I\} \cup \{S\}$, we can compute an appropriate abstraction of $\times_{i\in I}(G_i \times S_i) \times S$ (by using the proposed SAP) so that high level local supervisors and/or coordinators can be synthesized. This will be illustrated in the example of Supervisory Control of Cable TV Service Network.

3.3 Coordinated Distributed Supervisors with Nonstandardized automata

So far we have only considered standardized automata. It is of primary interest for us to know whether it is possible to apply the proposed technique to nonstandardized automata. The answer is yes and our general strategy is that, we first convert nonstandardized automata into standardized ones, then apply the synthesis approach proposed in the previous section, and finally we convert the standardized distributed supervisor into a nonstandardized one. To show that such a strategy works, we need to introduce a few concepts and results first, which are described as follows.

Given $G \in \phi(\Sigma)$ and $S = (Y, \Sigma, \eta, y_0, Y_m) \in \phi(\Sigma)$, we propose the following computational procedure, which is denoted as PODS, standing for Procedure for Observation Driven Supervisor:

1. Let $S' = (Y', \Sigma_o \cup \{\tau\}, \eta', y_0', Y_m') \in \phi(\Sigma_o \cup \{\tau\})$ be the deterministic canonical recognizer of $P_\tau(N(G \times S))$ with $B(S') = \varnothing$, where $P_\tau : \Sigma^* \to (\Sigma_o \cup \{\tau\})^*$ is the natural projection. For any state $y' \in Y'$, an event $\sigma \in \Sigma_{uo} - \{\tau\}$ is called *relevant* at y' with respect to G, denoted as $\sigma \pitchfork_{G \times S} y'$, if
$$(\exists s \in \Sigma_o^*)\, \eta'(y_0', s) = y' \wedge P_\tau^{-1}(s)\sigma \cap L(G \times S) \neq \varnothing$$
2. Output $S'' = (Y'', \Sigma, \eta'', y_0'', Y_m'') \in \phi(\Sigma)$, where $Y'' = Y'$, $Y_m'' = Y_m'$, $y_0'' = y_0'$ and the transition map η'' is defined as follows:
$$(\forall y'' \in Y'')(\forall \sigma \in \Sigma)\, \eta''(y'', \sigma) := \begin{cases} \eta'(y'', \sigma) & \text{if } \sigma \in \Sigma_o \cup \{\tau\} \\ y'' & \text{if } \sigma \in \Sigma_{uo} \text{ and } \sigma \pitchfork_{G \times S} y'' \end{cases}$$

□

What this procedure does is that: first we create a canonical recognizer S' of $P_\tau(N(G \times S))$, then at each state y' of S' we selfloop all unobservable events (other than τ) that are relevant at y' with respect to G. Clearly, S'' is still standardized. We have the following result.

Proposition 3.8. Given $G \in \phi(\Sigma)$ and $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$, let $S \in \phi(\Sigma)$ be a nonblocking state-observable (or state-normal) supervisor of G under H. Suppose S'' is obtained by PODS. Then S'' is a nonblocking state-observable (or state-normal) supervisor of G under H. □

The proof of Prop. 3.8 is presented in the Appendix. We call such an S'' a standardized *implementable* nonblocking supervisor of G with respect to H, in the sense that, except for the τ transition, S'' moves from one state to a different state only through observable transitions. For notation simplicity, we will use $\rho(S, G)$ to denote S'' computed by PODS. When G is clear from the context or not specified explicitly, we use $\rho(S)$ to denote S''. The proof of Prop. 3.8 indicates that, every nonblocking supervisor S can be converted into a standardized implementable nonblocking supervisor S'' such that $N(G \times S) = N(G \times S'')$ and $L(G \times S) = L(G \times S'')$. We will use this fact to convert a nonblocking supervisor modeled by a standardized automaton into a nonblocking supervisor modeled by a nonstandardized automaton. To this end, we introduce the concepts of *standardization* and *destandardization*. To avoid unnecessary confusion, here we emphasize that, from now on in this section we assume that τ and μ are not contained in any alphabet, and $\varphi(\Sigma)$

denotes the collection of all *nonstandardized* automata, whose alphabet is Σ.

Definition 3.9. Given $G = (X, \Sigma, \xi, x_0, X_m)$, we say an automaton $G^{\uparrow} = (X^{\uparrow}, \Sigma \cup \{\tau, \mu\}, \xi^{\uparrow}, x_0^{\uparrow}, X_m^{\uparrow})$ is *G-standardized* if

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

□

The only difference between $G^{\uparrow}$ and G is that, the former contains a new state $x_0^{\uparrow}$, a new transition τ from $x_0^{\uparrow}$ to x_0, and selflooping μ at each marker state. From now on we use $\theta(G)$ to denote the G-standardized automaton $G^{\uparrow}$. Next, we introduce the concept of *destandardization*, which is used to convert a standardized automaton into a nonstandardized one.

Definition 3.10. A standardized automaton $G^{\uparrow} = (X^{\uparrow}, \Sigma \cup \{\tau, \mu\}, \xi^{\uparrow}, x_0^{\uparrow}, X_m^{\uparrow})$ is *μ-selflooping* if for any $x, x' \in X^{\uparrow}$, we have that $x' \in \xi^{\uparrow}(x, \mu)$ implies $x' = x$. □

Definition 3.11. Let $S^{\uparrow} = (Y^{\uparrow}, \Sigma \cup \{\tau, \mu\}, \eta^{\uparrow}, y_0^{\uparrow}, Y_m^{\uparrow})$ be a deterministic μ-selflooping standardized automaton. We say an automaton $S = (Y, \Sigma, \eta, y_0, Y_m)$ is *$S^{\uparrow}$-destandardized* if

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

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

We have the following result.

Theorem 3.12. Given a distributed system $\mathcal{G} = \{G_i \in \varphi(\Sigma_i) | i \in I\}$ and a collection of deterministic specifications $\mathcal{H} = \{H_j \in \varphi(\Delta_j) | \Delta_j \subseteq \cup_{i \in I} \Sigma_i \wedge j \in J\}$, let $\mathcal{G}^{\uparrow} := \{\theta(G_i) | i \in$

$I\}$ be the standardized distributed system and $\mathcal{H}^{\uparrow} := \{\theta(H_j)|j \in J\}$ for the standardized deterministic specifications. If there exists a nonblocking distributed supervisor

$$\mathcal{S}^{\uparrow} := \{S_k^{\uparrow} \in \phi(\Gamma_k^{\uparrow})|\Gamma_k^{\uparrow} \subseteq \cup_{i\in I}\Sigma_i \cup \{\tau, \mu\} \wedge S_k^{\uparrow} \text{ is } \mu\text{-selflooping} \wedge k \in K\}$$

of $\mathcal{G}^{\tau}$ under $\mathcal{H}^{\tau}$, then $\mathcal{S} := \{\nu(S_k^{\uparrow})|k \in K\}$ is a nonblocking distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. □

Proof: Let $G := \times_{i\in I}G_i$, $H := \times_{j\in J}H_j$ and $S := \times_{k\in K}\nu(S_k^{\uparrow})$. By Def. 3.9 we get that $G^{\uparrow} := \times_{i\in I}\theta(G_i)$ is a standardized μ-selflooping automaton, and so is $H^{\uparrow} := \times_{j\in J}\theta(H_j)$. Since each $S_k^{\uparrow}$ $(k \in K)$ is standardized μ-selflooping, we can derive that $S^{\uparrow} := \times_{k\in K}S_k^{\uparrow}$ is a standardized μ-selflooping automaton. Thus, $G^{\uparrow} \times S^{\uparrow}$ is a standardized μ-selflooping automaton. Furthermore, we can show that, $\nu(G^{\uparrow} \times S^{\uparrow})$ is DES-isomorphic to $G \times S$ and $\nu(G^{\uparrow} \times H^{\uparrow})$ is DES-isomorphic to $G \times H$, where *DES-isomorphism* is defined in [22], which simply says that, two automata are essentially identical, except for their state labels, which are mapped bijectively between two state sets. Thus, it is straightforward to show that, if $\mathcal{S}^{\uparrow}$ is a nonblocking state-observable (or state-normal) distributed supervisor of $\mathcal{G}^{\uparrow}$ under $\mathcal{H}^{\uparrow}$, then $\mathcal{S}$ is a nonblocking state-observable (or state-normal) distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. ■

We now present the following Procedure for Synthesis of Distributed Supervisors with Coordinators modeled by Nonstandardized Automata (PSDSCNA).

1. Inputs:

   $$\mathcal{G} = \{G_i \in \varphi(\Sigma_i)|i \in I = \{1, 2, \cdots, n\}\},\ \mathcal{H} = \{H_j \in \varphi(\Delta_j)|j \in J\} \cup \{H \in \phi(\Delta)\}$$

2. Create $\mathcal{G}^{\uparrow} = \{\theta(G_i)|i \in I\}$ and $\mathcal{H}^{\uparrow} = \{\theta(H_j)|j \in J\} \cup \{\theta(H)\}$

3. Compute the collection $\hat{\mathcal{S}}^{\uparrow} = \{\rho(S_j^{\uparrow})|j \in J\} \cup \{\rho(S^{\uparrow})\}$ as follows:

   (a) For each $j \in J$, let $I_j \subseteq I$ and $\Sigma_{I_j} := \cup_{i\in I_j}\Sigma_i$ such that $\Delta_j \subseteq \Sigma_{I_j}$

   (b) Use the procedure PSNSNS in [11] to compute the supremal nonblocking state-normal supervisor $S_j^{\uparrow} \in \phi(\Sigma_{I_j} \cup \{\tau, \mu\})$ of $\times_{i\in I_j}\theta(G_i)$ under $\theta(H_j)$

   (c) Choose $\Sigma' \subseteq \cup_{i\in I}\Sigma_i$ such that $\Delta \subseteq \Sigma'$

   (d) Compute abstraction $G^{\uparrow} := (\times_{i\in I}\theta(G_i) \times_{j\in J} \rho(S_j^{\uparrow}))/\approx_{\Sigma'\cup\{\tau,\mu\}}$

   (e) Compute the supremal nonblocking state-normal supervisor $S^{\uparrow}$ of $G^{\uparrow}$ under $\theta(H)$

4. Output $\mathcal{S} = \{\nu(\rho(S_j^{\uparrow}))|j \in J\} \cup \{\nu(\rho(S^{\uparrow}))\}$ □

Corollary 3.13. $\mathcal{S}$ computed in PSDSCNA is a nonblocking distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. □

Proof: By Prop. 3.8 and Theorem 3.7 we can derive that $\hat{\mathcal{S}}^{\uparrow} := \{\rho(S_j^{\uparrow})|j \in J\} \cup \{\rho(S^{\uparrow})\}$ is a nonblocking distributed supervisor of $\mathcal{G}^{\uparrow}$ under $\mathcal{H}^{\uparrow}$. By the definition of ρ, each automaton in $\hat{\mathcal{S}}^{\uparrow}$ is μ-selflooping. Thus, by Theorem 3.12 we get that $\mathcal{S}$ is a nonblocking distributed supervisor of $\mathcal{G}$ under $\mathcal{H}$. ■

At this point we can see that, introducing events τ, μ and the concept of standardized automata does not impose any significant constraint on synthesis of distributed supervisors. They are used only for the purpose of applying automaton abstraction in synthesis. Next, we will use an example to illustrate concepts and computational procedures introduced in the previous sections.

4 Example - Cable TV Service Network

Suppose a cable TV company wants to build a TV service network in a city. For the illustration purpose, suppose the city consists of 3 communities C_1, C_2 and C_3, and each community C_i $(i = 1, 2, 3)$ has 3 families F_1^i, F_2^i and F_3^i. The company wants to sell cable TV service to each family. They offer two types of packages: the basic package β and the advanced package α. To offer a package, a certain procedure needs to follow. Figure 1 depicts the procedure for offering the basic package β to family F_j^i in Community C_i $(i = 1, 2, 3$ and $j = 1, 2, 3)$, where the alphabet $\Sigma_{\beta,j}^i$ is the collection of all events

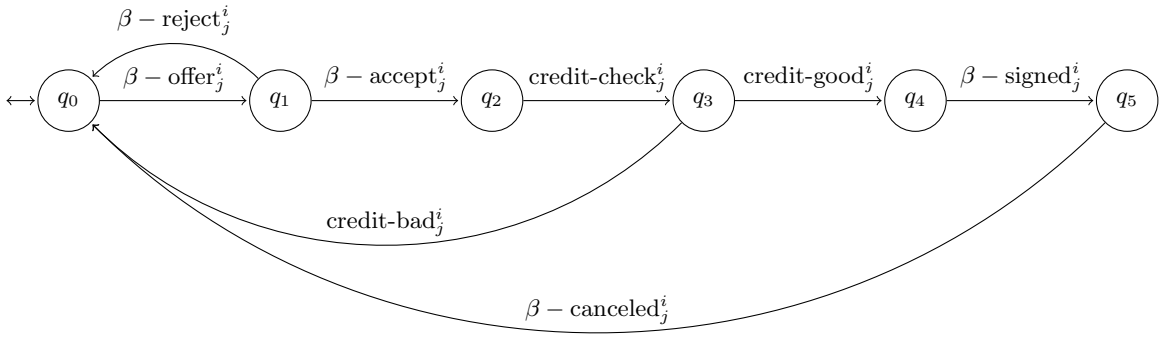

Figure 1: Automaton Model $G_{\beta,j}^i \in \phi(\Sigma_{\beta,j}^i)$

appearing in Figure 1. The controllable alphabet is $\Sigma_{\beta,j,c}^i = \{\beta - \text{offer}_j^i, \text{credit-check}_j^i, \beta - \text{canceled}_j^i\}$, and $\Sigma_{\beta,j,o}^i = \Sigma_{\beta,j}^i$, namely every event is observable for the sake of simplicity. Similarly, Figure 2 depicts the procedure of offering the advanced package α to family F_j^i in Community C_i, where the controllable alphabet is $\Sigma_{\alpha,j,c}^i = \{\alpha - \text{offer}_j^i\}$. The reason that $\beta - \text{canceled}_i^j$ is controllable but $\alpha - \text{canceled}_i^j$ is uncontrollable is because a user can cancel the advanced package any time he/she wants, but to cancel the basic package, he/she needs to clear all existent account balances and cancel α package first if applicable - in other words, a user cannot cancel the basic package at will. The specification $H_{F_j^i} \in \phi(\Delta_{F_j^i})$ that describes how package β and package α are offered together to family F_j^i is depicted in Figure 3, which says that, the advanced package α can be offered only after the basic package β is signed. The alphabet $\Delta_{F_j^i}$ is the collection of all events appearing in Figure 3. Each community has a restriction on the total number of signed basic packages, owing to the bandwidth limit. For the illustration purpose, suppose the maximum number of signed β packages for each community is 2. Such a specification H_{C_i} for community C_i $(j = 1, 2, 3)$ is depicted in Figure 4, where Σ_{signed}^i

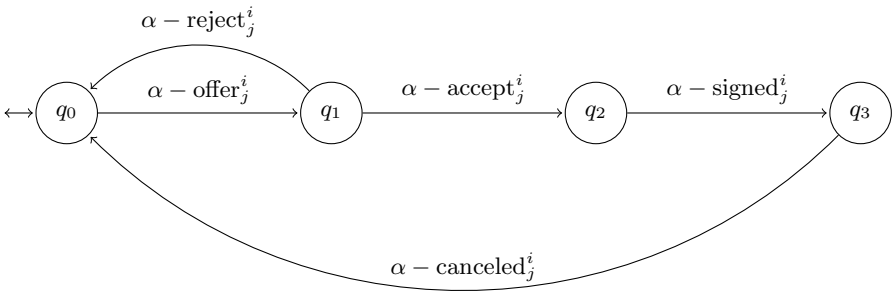

Figure 2: Automaton Model $G^i_{\alpha,j} \in \phi(\Sigma^i_{\alpha,j})$

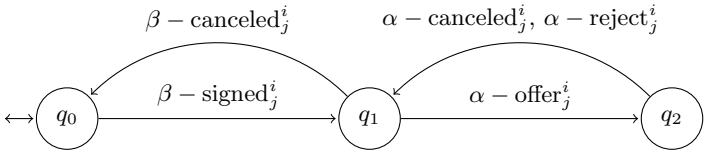

Figure 3: Family Specification $H_{F^i_j} \in \phi(\Delta_{F^i_j})$

denotes the collection of events $\{\beta - \text{signed}^i_j | i = 1, 2, 3\}$ and $\Sigma^i_{\text{canceled}}$ denotes the collection of events $\{\beta - \text{canceled}^i_j | i = 1, 2, 3\}$. The alphabet Δ_{C_i} is the collection of all events appearing in Figure 4. Finally, at the city level the total number of advanced packages is also restricted, owing to the bandwidth limit and city laws. For the illustration purpose, suppose the maximum total number of signed α packages in communities C_1 and C_2 is 3. The specification H is depicted in Figure 5. where Σ_{signed} denotes the collection of events $\{\alpha - \text{signed}^i_j | i = 1, 2, 3 \wedge j = 1, 2\}$ and Σ_{canceled} denotes the collection of events $\{\alpha - \text{canceled}^i_j | i = 1, 2, 3 \wedge j = 1, 2\}$. We now apply the proposed coordinated synthesis approach to synthesize a nonblocking distributed supervisor.

We first standardize every component model and specification. Then for each family F^i_j we compute the product $G^i_j := G^i_{\alpha,j} \times G^i_{\beta,j} \in \phi(\Sigma_{F^i_j})$ with $\Sigma_{F^i_j} := \Sigma^i_{\alpha,j} \cup \Sigma^i_{\beta,j}$, which is treated as the local plant model with the local specification $H_{F^i_j} \in \phi(\Delta_{F^i_j})$. By using a procedure presented in [11] we can compute the supremal nonblocking state-normal supervisor $S_{F^i_j} \in \phi(\Sigma_{F^i_j})$ of G^i_j under $H_{F^i_j}$. The relevant computational results are listed below:

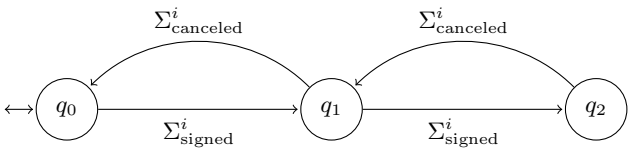

Figure 4: Community Specification $H_{C_i} \in \phi(\Delta_{C_i})$

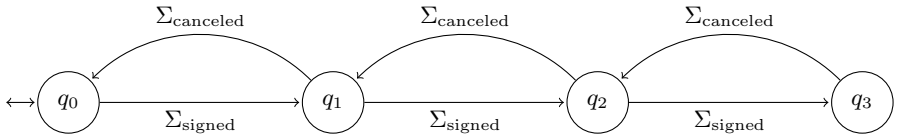

Figure 5: City Specification H

$$G_j^i \text{ (25, 63) ; } H_{F_j^i} \text{ (4, 7) ; } S_{F_j^i} \text{ (10, 14)}$$

where each tuple (x, y) denotes x states and y transitions. After we obtain local supervisors $\{S_{F_j^i} | j = 1, 2, 3\}$, we compute a coordinator S_{C_i} that enforces the community-level specification H_{C_i}. To this end, by using SAP we compute an abstraction

$$G_{C_i} := (\times_{i=1}^{3}(G_j^i \times S_{F_j^i}))/ \approx_{\Sigma'^i}$$

where $\Sigma'^i \subseteq \cup_{j=1}^{3}\Sigma_j^i$ and $\Delta_{C_i} \subseteq \Sigma'^i$. To make sure that the abstracted model G_{C_i} contains sufficient control means, we define $\Sigma'^i := \Delta_{C_i} \cup \{\beta - \text{offer}_j^i | i = 1, 2, 3\}$. After that we compute the supremal nonblocking state-normal supervisor $S_{C_i} \in \phi(\Sigma'^i)$ of G_{C_i} under H_{C_i}. The computational results are listed as follows:

$$G_{C_i} \text{ (65, 685) ; } H_{C_i} \text{ (4, 14) ; } S_{C_i} \text{ (21, 64)}$$

Finally, we compute one more coordinator to take care of the specification H. To this end we first compute an abstraction

$$G := (\times_{i=1}^{3}((\times_{j=1}^{3}(G_j^i \times S_{F_j^i})) \times S_{C_i}))/ \approx_{\Sigma'}$$

where $\Sigma' \subseteq \cup_{i=1}^{3}\cup_{j=1}^{3}\Sigma_j^i$ and $\Delta \subseteq \Sigma'$. To make sure that the abstracted model G contains sufficient control means, we define $\Sigma' := \Delta \cup \{\alpha - \text{offer}_j^i | i = 1, 2 \wedge j = 1, 2, 3\}$. After that, we compute the supremal nonblocking state-normal supervisor S of G under H. The computational results are listed as follows:

$$G \text{ (1408, 49005) ; } H \text{ (5, 38) ; } S \text{ (462, 2995)}$$

By using the nonconflict-checking procedure provided in [12] we confirm that, the coordinated distributed supervisor $\times_{i=1}^{3}((\times_{j=1}^{3}S_{F_j^i}) \times S_{C_i}) \times S$ is nonconflicting with $\times_{i=1}^{3}\times_{j=1}^{3}$ $(G_{\alpha,j}^i \times G_{\beta,j}^i)$.

Clearly, the centralized synthesis will not work well for this example because the size of the product of all local components is $25^6 = 244140625$. The language-based abstraction is also computationally inefficient for this example because, to make sure each involved natural projection is an observer, the projected images are not small enough. For example, to synthesize the coordinator S_{C_1}, if we use natural projections to compute abstractions and we choose the alphabet $\Sigma'^1 = \{\beta-\text{offer}_j^1, \beta-\text{signed}_j^1, \beta-\text{canceled}_j^1 | i = 1, 2, 3\}$, then in order to make the relevant natural projections to be observers we need to extend Σ'^1 to the set $\{\beta-\text{offer}_j^1, \beta-\text{signed}_j^1, \beta-\text{canceled}_j^1, \beta-\text{reject}_j^1, \text{credit-good}_j^1, \text{credit-bad}_j^1 | i = 1, 2, 3\}$, which results in an abstraction with 76 states, in contrast to 65 states obtained by our automaton abstraction approach. The difference becomes significant when more families in each community are involved - the ratio of the size of G_{C_1} obtained by our abstraction

approach and that of the language-based approach is roughly $(1.25)^j$, where j is the number of families in a community. Thus, our proposed approach has clear computational advantage over centralized synthesis approaches and language-based modular synthesis approaches.

5 Conclusions

In this paper we introduce a coordinated distributed supervisor synthesis approach based on abstractions of nondeterministic finite-state automata. The main advantage of this approach is its simplicity and potentially low computational complexity in contrast to existant distributed synthesis approaches based on observers. When a module contains a large number of components, we can apply the proposed SAP procedure to obtain an abstraction, which may significantly reduce the computational complexity. Because supervisor synthesis is done in a local fashion, high complexity incurred by synchronous product of a large number of components may be avoided. Besides, a certain degree of implementation flexibility can be achieved in terms of reusing 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 example of Section IV.

1. Proof of Prop. 3.6: Let $G_i = (X_i, \Sigma, \xi_i, x_{i,0}, X_{i,m})$ $(i = 1, 2)$ and $S = (Y, \Sigma, \eta, y_0, Y_m)$.
(1) First, we have

$$\begin{aligned} N(G_1 \times S) &= N(G_1)||N(S) \\ &= N(G_2)||N(S) \text{ because } G_1 \sqsubseteq G_2 \\ &= N(G_2 \times S) \\ &\subseteq N(G_2 \times H) \text{ because } S \text{ is a nonblocking supervisor of } G_2 \text{ under } H \\ &= N(G_2)||N(H) \\ &= N(G_1)||N(H) \\ &= N(G_1 \times H) \end{aligned}$$

Therefore, we have $N(G_1 \times S) \subseteq N(G_1 \times H)$.
(2) Since $G_1 \sqsubseteq G_2$, by Prop. 2 in [12] we have $G_1 \times S \sqsubseteq G_2 \times S$, which means $B(G_1 \times S) \subseteq B(G_2 \times S)$. Since S is a nonblocking supervisor of G_2 under H, we have $B(G_2 \times S) = \varnothing$. Thus $B(G_1 \times S) = \varnothing$.
(3) We now show S is state-controllable with respect to G_1 and Σ_{uc}. By Def. 3.1 we need to show that

$$(\forall s \in L(G_1 \times S))(\forall x_1 \in \xi_1(x_{1,0}, s))(\forall y \in \eta(y_0, P(s)))\, E_{G_1}(x_1) \cap \Sigma_{uc} \subseteq E_S(y)$$

To this end, let $s \in L(G_1 \times S)$. Since we have shown that $B(G_1 \times S) = \varnothing$, we have

$$L(G_1 \times S) = \overline{N(G_1 \times S)} = \overline{N(G_2 \times S)} = L(G_2 \times S)$$

Clearly, $E_{G_1}(x_1) \subseteq \cup_{x_2 \in \xi_2(x_{2,0}, s)} E_{G_2}(x_2)$ because $G_1 \sqsubseteq G_2$ implies that $L(G_1) \subseteq L(G_2)$. Since S is deterministic and state-controllable with respect to G_2 and Σ_{uc}, we have

$$\cup_{x_2 \in \xi_2(x_{2,0}, s)} E_{G_2}(x_2) \cap \Sigma_{uc} \subseteq E_S(y)$$

which means

$$E_{G_1}(x_1) \cap \Sigma_{uc} \subseteq E_S(y)$$

Thus, S is state-controllable with respect to G_1 and Σ_{uc}.
(4) Suppose S is state-observable with respect to G_2 and P_o. We need to show that S is state-observable with respect to G_1 and P_o. By Def. 3.2 we need to show that, for any $s, s' \in L(G_1 \times S)$ with $P_o(s) = P_o(s')$, we have

$$(\forall (x_1, y) \in \xi_1 \times \eta((x_{1,0}, y_0), s))(\forall (x_1', y') \in \xi_1 \times \eta((x_{1,0}, y_0), s'))\, E_{G_1 \times S}(x_1, y) \cap E_{G_1}(x') \subseteq E_S(y')$$

To this end, let $s, s' \in L(G_1 \times S)$ with $P_o(s) = P_o(s')$. Since $L(G_1 \times S) = L(G_2 \times S)$, we have $s, s' \in L(G_2 \times S)$, and

$$E_{G_1 \times S}(x_1, y) \subseteq \cup_{(x_2, y) \in \xi_2 \times \eta((x_{2,0}, y_0), s)} E_{G_2 \times S}(x_2, y)$$

Since $L(G_1) \subseteq L(G_2)$, we have

$$E_{G_1}(x_1') \subseteq \cup_{x_2' \in \xi_2(x_{2,0}, s')} E_{G_2}(x_2')$$

Since S is deterministic and state-observable with respect to G_2 and P_o, we have

$$(\cup_{(x_2, y) \in \xi_2 \times \eta((x_{2,0}, y_0), s)} E_{G_2 \times S}(x_2, y)) \cap (\cup_{x_2' \in \xi_2(x_{2,0}, s')} E_{G_2}(x_2')) \subseteq E_S(y')$$

Thus,

$$E_{G_1 \times S}(x_1, y) \cap E_{G_1}(x') \subseteq E_S(y')$$

which means S is state-observable with respect to G_1 and P_o.
(5) Finally, suppose S is state-normal with respect to G_2 and P_o. We need to show that S is state-normal with respect to G_1 and P_o. By Def. 3.3 we need to show that, for any $s \in L(G_1 \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G_1 \times S)$, we have

$$(\forall (x_1, y) \in \xi_1 \times \eta((x_{1,0}, y_0), s'))(\forall s'' \in \Sigma^*)\, P_o(s's'') = P_o(s) \Rightarrow [\xi_1(x_1, s'') \neq \varnothing \Rightarrow \eta(y, s'') \neq \varnothing]$$

To this end, let $s \in L(G_1 \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G_1 \times S)$. Since $L(G_1 \times S) = L(G_2 \times S)$, we have $s \in L(G_2 \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G_2 \times S)$. For any $s'' \in \Sigma^*$, if $P_o(s's'') = P_o(s)$ and $\xi_1(x_1, s'') \neq \varnothing$, we get that $s's'' \in L(G_1) \subseteq L(G_2)$. Thus, there exists $(x_2, y) \in \xi_2 \times \eta((x_{2,0}, y_0), s')$ such that

$$P_o(s's'') = P_o(s) \wedge \xi_2(x_2, s'') \neq \varnothing$$

Since S is deterministic and state-normal with respect to G_2 and P_o, we have $\eta(y, s'') \neq \varnothing$. Thus, S is state-normal with respect to G_1 and P_o.
From (1)-(5) we get that, S is a nonblocking state-observable (or state-normal) supervisor of G_2 under H implies that S is a nonblocking state-observable (or state-normal) supervisor of G_1 under H. ■

2. Proof of Theorem 3.7: Let $G_i = (X_i, \Sigma_i, \xi_i, x_{i,0}, X_{i,m})$ and $S_i = (Y_i, \Sigma_i, \eta_i, y_{i,0}, Y_{i,m})$ for each $i \in I$, and $S = (Y, \Sigma', \eta, y_0, Y_m)$. By Prop. 2.4 we get that $(\times_{i \in I}(G_i \times S_i))/\approx_{\Sigma'} \cong \times_{i \in I}((G_i \times S_i)/\approx_{\Sigma_i \cap \Sigma'})$. Since $(G_i \times S_i)/\approx_{\Sigma_i \cap \Sigma'} \sqsubseteq W_i$, by Prop. 2 in [12] we get that

$$(\times_{i \in I}(G_i \times S_i))/\approx_{\Sigma'} \sqsubseteq \times_{i \in I}((G_i \times S_i)/\approx_{\Sigma_i \cap \Sigma'}) \sqsubseteq \times_{i \in I} W_i$$

Since S is a nonblocking state-observable (or state-normal) supervisor of $\times_{i \in I} W_i$ under H, by Prop. 3.6 we get that, S is a nonblocking state-observable (or state-normal) supervisor of $(\times_{i \in I}(G_i \times S_i))/\approx_{\Sigma'}$ under H. By Theorem 3 in [10] we get that, S is a nonblocking state-observable (or state-normal) supervisor of $\times_{i \in I}(G_i \times S_i)$ under H, which means

$$N(\times_{i \in I} G_i \times S \times_{j \in I} S_j) = N(\times_{i \in I}(G_i \times S_i) \times S) \subseteq N(\times_{i \in I}(G_i \times S_i) \times H)$$

Since S_i is a nonblocking supervisor of G_i under H_i, we have $N(G_i \times S_i) \subseteq N(G_i \times H_i)$. Thus,

$$N(\times_{i \in I} G_i \times S \times_{j \in I} S_j) \subseteq N(\times_{i \in I}(G_i \times H_i) \times H) = N(\times_{i \in I} G_i \times H \times_{j \in I} H_j)$$

Furthermore, we have $B(\times_{i \in I} G_i \times S \times_{j \in I} S_j) = B(\times_{i \in I}(G_i \times S_i) \times S) = \varnothing$.
Next, we show that $S \times_{i \in I} S_i$ is state-controllable with respect to $\times_{i \in I} G_i$ and $\cup_{i \in I} \Sigma_{i,uc}$.

For notational brevity, let $\hat{S} = S \times_{i\in I} S_i$, $\hat{G} = \times_{i\in I} G_i$, $\hat{\xi} = \times_{i\in I}\xi_i$, $\hat{\eta} = \eta \times_{i\in I} \eta_i$ and $\Sigma_{uc} = \cup_{i\in I}\Sigma_{i,uc}$. By Def. 3.1 we need to show that

$$(\forall s \in L(\hat{G}\times\hat{S}))(\forall \hat{x} \in \hat{\xi}(\hat{x}_0, s))(\forall \hat{y} \in \hat{\eta}(\hat{y}_0, s))\, E_{\hat{G}}(\hat{x}) \cap \Sigma_{uc} \subseteq E_{\hat{S}}(\hat{y})$$

To this end, let $s \in L(\hat{G}\times\hat{S})$, $\hat{x} = (x_1, x_2, \cdots, x_n)$ and $\hat{y} = (y, y_1, y_2, \cdots, y_n)$. For each $i \in I$, let $P_i : (\cup_{j\in I}\Sigma_j)^* \to \Sigma_i^*$ be the natural projection. For each $\sigma \in E_{\hat{G}}(\hat{x}) \cap \Sigma_{uc}$, if $\sigma \in \Sigma_i$, then by the assumption (A1) we have $\sigma \in \Sigma_{i,uc}$. Furthermore, we get that $\sigma \in E_{G_i}(x_i) \cap \Sigma_{i,uc}$. Since S_i is deterministic and state-controllable with respect to G_i and $\Sigma_{i,uc}$, we get that $\eta_i(y_i, \sigma) \neq \varnothing$. Thus, $\sigma \in E_{\times_{i\in I}(G_i\times S_i)}(x_1, y_1, \cdots, x_n, y_n)$. Since S is state-controllable with respect to $\times_{i\in I}(G_i \times S_i)$ and Σ_{uc}, if $\sigma \in \Sigma'$, we get that $\eta(y, \sigma) \neq \varnothing$. Thus, $\hat{\eta}(\hat{y}, \sigma) \neq \varnothing$, which means $\sigma \in E_{\hat{S}}(\hat{y})$. Therefore, $E_{\hat{G}}(\hat{x}) \cap \Sigma_{uc} \subseteq E_{\hat{S}}(\hat{y})$.
Next, assume that S_i is state-observable with respect to G_i and $P_{i,o} : \Sigma_i^* \to \Sigma_{i,o}^*$, and S is state-observable with respect to $\times_{i\in I}(G_i \times S_i)$ and $P_o : (\cup_{i\in I}\Sigma_i)^* \to (\cup_{i\in I}\Sigma_{i,o})^*$. We need to show that $\hat{S}$ is state-observable with respect to $\hat{G}$ and P_o. By Def. 3.2 we need to show that, for any $s, s' \in L(\hat{G}\times\hat{S})$ with $P_o(s) = P_o(s')$, we have

$$(\forall(\hat{x}, \hat{y}) \in \hat{\xi} \times \hat{\eta}((\hat{x}_0, \hat{y}_0), s))(\forall(\hat{x}', \hat{y}') \in \hat{\xi} \times \hat{\eta}((\hat{x}_0, \hat{y}_0), s'))\, E_{\hat{G}\times\hat{S}}(\hat{x}, \hat{y}) \cap E_{\hat{G}}(\hat{x}') \subseteq E_{\hat{S}}(\hat{y}')$$

To this end, let $s, s' \in L(\hat{G}\times\hat{S})$ with $P_o(s) = P_o(s')$, $\hat{x} = (x_1, \cdots, x_n)$, $\hat{x}' = (x_1', \cdots, x_n')$, $\hat{y} = (y, y_1, \cdots, y_n)$ and $\hat{y}' = (y', y_1', \cdots, y_n')$. For each $\sigma \in E_{\hat{G}\times\hat{S}}(\hat{x}, \hat{y}) \cap E_{\hat{G}}(\hat{x}')$, if $\sigma \in \Sigma_i$, then we get that $\sigma \in E_{G_i\times S_i}(x_i) \cap E_{G_i}(x_i')$. Since S_i is deterministic and state-observable with respect to G_i and $P_{i,o}$, by the assumption (A1) we can derive that $\eta_i(y_i', \sigma) \neq \varnothing$. Thus, $\sigma \in E_{\times_{i\in I}(G_i\times S_i)}(x_1', y_1', \cdots, x_n', y_n')$. Since S is state-observable with respect to $\times_{i\in I}(G_i \times S_i)$ and P_o, if $\sigma \in \Sigma'$, we get that $\eta(y', \sigma) \neq \varnothing$. Thus, $\hat{\eta}(\hat{y}', \sigma) \neq \varnothing$, which means $\sigma \in E_{\hat{S}}(\hat{y}')$. Therefore, $E_{\hat{G}\times\hat{S}}(\hat{x}, \hat{y}) \cap E_{\hat{G}}(\hat{x}') \subseteq E_{\hat{S}}(\hat{y}')$.
Finally, assume that S_i is state-normal with respect to G_i and $P_{i,o}$, and S is state-normal with respect to $\times_{i\in I}(G_i \times S_i)$ and P_o. We need to show that $\hat{S}$ is state-normal with respect to $\hat{G}$ and P_o. By Def. 3.3 we need to show that, for any $s \in L(\hat{G}\times\hat{S})$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(\hat{G}\times\hat{S})$, we have

$$(\forall(\hat{x}, \hat{y}) \in \hat{\xi}\times\hat{\eta}((\hat{x}_0, \hat{y}_0), s'))(\forall s'' \in \Sigma^*)\, P_o(s's'') = P_o(s) \Rightarrow [\hat{\xi}(\hat{x}, s'') \neq \varnothing \Rightarrow \hat{\eta}(\hat{y}, s'') \neq \varnothing]$$

To this end, let $s \in L(\hat{G}\times\hat{S})$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(\hat{G}\times\hat{S})$. Suppose $P_o(s's'') = P_o(s)$ and $\hat{\xi}(\hat{x}, s'') \neq \varnothing$. We need to show that $\hat{\eta}(\hat{y}, s'') \neq \varnothing$. Let $\hat{x} = (x_1, \cdots, x_n)$, $\hat{y} = (y, y_1, \cdots, y_n)$, and $P_i : (\cup_{j\in I}\Sigma_j)^* \to \Sigma_i^*$, $P' : (\cup_{j\in I}\Sigma_j)^* \to \Sigma'^*$ be the natural projection. Then we have $P_i(s) \in L(G_i \times S_i)$, $P_i(s') \in \overline{P_o^{-1}(P_{i,o}(P_i(s)))} \cap L(G_i \times S_i)$. Furthermore, by the assumption (A1) we have $P_{i,o}(P_i(s's'')) = P_{i,o}(P_i(s))$ and $\xi_i(x_i, P_i(s'')) \neq \varnothing$. Since S_i is deterministic and state-normal with respect to G_i and $P_{i,o}$, we get that $\eta_i(y_i, P_i(s'')) \neq \varnothing$. Thus, $\times_{i\in I}\xi_i\times\eta_i((x_1, y_1, \cdots, x_n, y_n), s'') \neq \varnothing$. Since S is state-normal with respect to $\times_{i\in I}(G_i\times S_i)$ and P_o, we get that $\eta(y, P'(s'')) \neq \varnothing$. Thus, $\hat{\eta}(\hat{y}, s'') \neq \varnothing$.■

3. Proof of Prop. 3.8: We first show that $L(G \times S'') = L(G \times S)$. By the construction of S'' we have $L(G \times S) \subseteq L(G \times S'')$. So we only need to show $L(G \times S'') \subseteq L(G \times S)$. Suppose it is not true. Then there exists $s \in L(G \times S'')$ but $s \notin L(G \times S)$. Since $\epsilon \in L(G \times S) \cap L(S \times S'')$, there must exist $s'\sigma \leq s$ with $\sigma \in \Sigma$ such that $s' \in L(G \times S) \cap L(S \times S'')$ and $s'\sigma \in L(G \times S'')$ but $s'\sigma \notin L(G \times S)$. Clearly, $s'\sigma \in L(G)$. Furthermore, since $s'\sigma \in L(G \times S'')$, by the construction of S'', there exists $s''\sigma \in L(G \times S)$ such that $P_o(s') = P_o(s'')$. But this means S is not state-observable with respect to G and P_o, which contradicts the fact that S is a nonblocking supervisor of G under H. Thus, $L(G\times S'') \subseteq L(G\times S)$, which means $L(G\times S'') = L(G\times S)$. By using a similar argument we can show that, $N(G \times S'') = N(G \times S)$.
Since $N(G \times S) \subseteq N(G \times H)$, we have $N(G \times S'') \subseteq N(G \times H)$.
We now show that $B(G \times S'') = \varnothing$. Let $s \in L(G \times S'')$ and $(x, y'') \in \xi \times \eta''((x_0, y_0''), s)$.

Since $L(G \times S'') = L(G \times S)$, we have $s \in L(G \times S)$. Thus, there exists $y \in Y$ such that $(x, y) \in \xi \times \eta((x_0, y_0), s)$. Since S is a nonblocking supervisor of G, we have $B(G \times S) = \varnothing$, which means there exists $s' \in \Sigma^*$ such that $\xi \times \eta((x, y), s') \cap (X_m \times Y_m) \neq \varnothing$. Clearly, $ss' \in N(G \times S) = N(G \times S'')$. Since S'' is deterministic, we get that $\eta''(y'', s') \subseteq Y''_m$. Thus, $\xi \times \eta''((x, y''), s') \cap (X_m \times Y''_m) \neq \varnothing$. Thus, $B(G \times S'') = \varnothing$.
To show S'' is state-controllable with respect to G and Σ_{uc}, by Def. 3.1 we need to show that

$$(\forall s \in L(G \times S''))(\forall x \in \xi(x_0, s))(\forall y'' \in \eta''(y''_0, s))\, E_G(x) \cap \Sigma_{uc} \subseteq E_{S''}(y'')$$

Since $L(G \times S'') = L(G \times S)$ and S is state-controllable with respect to G and Σ_{uc}, we have

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

Since S and S'' are deterministic and by the construction of S'' we have $E_S(y) \subseteq E_{S''}(y'')$. Thus, we have $E_G(x) \cap \Sigma_{uc} \subseteq E_{S''}(y'')$, which means S'' is state-controllable with respect to G and Σ_{uc}.
Suppose S is state-observable with respect to G and P_o. To show S'' is state-observable with respect to G and P_o, by Def. 3.2 we need to show that, for any $s, s' \in L(G \times S'')$ with $P_o(s) = P_o(s')$, we have

$$(\forall (x, y'') \in \xi \times \eta''((x_0, y''_0), s))(\forall (\hat{x}, \hat{y}'') \in \xi \times \eta''((x_0, y''_0), s'))\, E_{G \times S''}(x, y'') \cap E_G(\hat{x}) \subseteq E_{S''}(\hat{y}'')$$

Since $L(G \times S'') = L(G \times S)$ and S and S'' are deterministic, we get that, there exist $y \in \eta(y_0, s)$ and $\hat{y} \in \eta(y_0, s')$ such that, $E_{G \times S}(x, y) = E_{G \times S''}(x, y'')$ and $E_S(\hat{y}) \subseteq E_{S''}(\hat{y}'')$. Since S is state-observable with respect to G and P_o, we have $E_{G \times S}(x, y) \cap E_G(\hat{x}) \subseteq E_S(\hat{y})$, from which we get $E_{G \times S''}(x, y'') \cap E_G(\hat{x}) \subseteq E_{S''}(\hat{y}'')$. Thus, S'' is state-observable with respect to G and P_o.
Suppose S is state-normal with respect to G and P_o. To show S'' is state-normal with respect to G and P_o, by Def. 3.3 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

$$(\forall (x, y'') \in \xi \times \eta''((x_0, y''_0), s'))(\forall s'' \in \Sigma^*)\, P_o(s's'') = P_o(s) \wedge \xi(x, s'') \neq \varnothing \Rightarrow \eta''(y'', s'') \neq \varnothing$$

Since $L(G \times S'') = L(G \times S)$, we have $s \in L(G \times S)$ and $s' \in \overline{P_o^{-1}(P_o(s))} \cap L(G \times S)$. Thus, there exists $y \in \eta(y_0, s')$. Since S is state-normal with respect to G and P_o, we have

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

Since both $\xi(x, s'') \neq \varnothing$ and $\eta(y, s'') \neq \varnothing$, we have $\xi \times \eta((x, y), s'') \neq \varnothing$, which means $s's'' \in L(G \times S) = L(G \times S'')$. Since S'' is deterministic, we get $\eta''(y'', s'') \neq \varnothing$. Thus, S'' is state-normal with respect to G and P_o.
Therefore, S'' is a nonblocking supervisor of G under H. ■

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*. IEEE Trans. Automatic Control (accepted with minor modifications), March, 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 and J.E. Rooda. *Aggregative synthesis of distributed supervisors based on automaton abstraction*. Submitted to IEEE Trans. Automatic Control, January, 2009. It also appears in SE Technical Report No. 2009-1, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Techonology, Eindhoven, The Netherlands, 2009. ISSN 1567-1872. URL: http://se.wtb.tue.nl/sereports.

[12] 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.

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

[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. 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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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