Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

Synthesizing Nonblocking Distributed Supervisors based on Automaton Abstraction

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

Abstract— **Blockingness is one of the major obstacles that need to be overcome in the Ramadge-Wonham supervisory synthesis paradigm. Owing to the high computational complexity usually resulted from synchronous product, it is difficult to synthesize a nonblocking supervisor for a large scale system. In this paper we aim to overcome this difficulty by presenting a distributed synthesis approach. The core idea of this approach is to create abstractions of parts of a target system during the synthesis process to avoid any potentially large representation. Meanwhile, those abstractions allow synthesizing local supervisors which guarantee the global nonblockingness, when they are applied to the target system in a conjunctive style.**

Index Terms— **discrete-event systems, nondeterministic finite-state automata, automaton abstraction, distributed synthesis**

I. Introduction

The automaton-based Ramadge-Wonham (RW) supervisory control paradigm first appeared in the control literature in 1982, which was subsequently summarized in [1] [2]. Since then there has been a large volume of literature under the same paradigm but with different architectural setups, e.g. [3] on modular control, [4] [5] on decentralized control, [6] on hierarchical control. One of the main challenges of RW supervisor synthesis is to achieve nonblockingness when a target system consists of 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 [8] 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 [8]. 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 [9] a supervisor synthesis approach for state feedback control is proposed based on the concept of *state tree structures*. It has been shown in [9] that a system with 10^{24} states can be well handled. Nevertheless, this approach is essentially a centralized approach. Thus, it is still not sufficient to deal with systems of industrial size. Besides, it does not consider partial observation.

Rong Su and Jacobus E. Rooda are affiliated with the Systems Engineering Group in Department of Mechanical Engineering, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands. Emails: R.Su@tue.nl, J.E.Rooda@tue.nl

Jan H. van Schuppen is affiliated with Centrum voor Wiskunde en Informatica (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. E-mail: J.H.van.Schuppen@cwi.nl

In this paper we present an aggregated distributed supervisor synthesis approach based on automaton abstraction. The idea of abstraction has been known in the literature, e.g. in [10] abstraction is used in the modular and hierarchical supervisor synthesis; and in [11] for decentralized control. Nevertheless, to make their approaches work, natural projections have to possess the *observer* property [7], which may not always hold by a natural projection. Although a natural projection can always be modified to become an observer (with respect to a specific language) [13], such a modification has a potential drawback in the sense that the alphabet of the codomain of the projection may be fairly large for the sake of achieving the observer property, and the consequence is that the size of the projected image may not be small enough to allow supervisor synthesis for large-scale systems. In [16] [17] the authors propose a new automaton abstraction technique. Such an abstraction has the following property: the product of a plant model G with another model S (e.g. a supervisor) is nonblocking if (and only if) the product of the abstraction $\kappa(G)$ of G and S is nonblocking. The abstraction technique bears similarity with the one proposed in [14]. But in [14] the authors require that a reduced model is weakly bisimilar to the original one. As a contrast, the new abstraction technique does not have such a requirement. The consequence is that, we usually end up with a reduced model simpler than what the technique proposed in [14] can achieve, resulting in different ways of supervisor synthesis subsequently. Based on the new abstraction operation we propose to compute a distributed supervisor in an aggregated way, which guarantees the global nonblockingness of the closed-loop system. This approach bears some similarity to the one proposed in [18]. But in [18] abstractions are done by using conflict equivalence similar to the one in [14], which is different from ours.

This paper is organized as follows. In Section II we introduce automaton composition and abstraction over nondeterministic automata. Then we present a distributed supervisor synthesis problem based on nondeterministic automata in Section III. Conclusions are stated in Section IV.

II. Automaton Composition and Abstraction

In this section we briefly review basic concepts and properties described in [16] [17], where more details and necessary proofs can be found. We follow the notations used in [15], and assume that readers are familiar with concepts of Ramadge-Wonham supervisory control paradigm, as

978-1-4244-3124-3/08/$25.00 ©2008 IEEE

described in [15].

Since automaton abstraction proposed later in this paper will result in nondeterministic automata, to avoid unnecessary complication we simply start with them. Given an alphabet Σ, let $\phi(\Sigma)$ be the collection of all nondeterministic finite-state automata with alphabet Σ. Given an automaton $G = (X, \Sigma, \xi, x_0, X_m)$, 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. We define a map $B : \phi(\Sigma) \to 2^{\Sigma^*}$ with

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

We call $B(G)$ the *blocking set* of G. Similarly, define another map $N : \phi(\Sigma) \to 2^{\Sigma^*}$, where for any $G \in \phi(\Sigma)$,

$$N(G) := \{s \in \Sigma^* | \xi(x_0, s) \neq \varnothing \wedge \xi(x_0, s) \cap X_m \neq \varnothing\}$$

We call $N(G)$ the *nonblocking set* of G. It is possible that $B(G) \cap \overline{N(G)} \neq \varnothing$, due to nondeterminism. We now introduce the parallel composition of automata.

Given two 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. For a slight abuse of notations, from now on we use $G_1 \times G_2$ to denote its reachability part. $\xi_1 \times \xi_2$ is extended to $X_1 \times X_2 \times (\Sigma_1 \cup \Sigma_2)^* \to 2^{X_1 \times X_2}$.

Suppose $G = (X, \Sigma, \xi, x_0, X_m)$. We bring in a new event symbol τ, which is uncontrollable and unobservable. An automaton $G = (X, \Sigma \cup \{\tau\}, \xi, x_0, X_m)$ is *standardized* if

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

We can always convert an automaton into a standard automaton. From now on, unless specified explicitly, we assume that each alphabet Σ contains τ, and $\phi(\Sigma)$ is the set of all standardized finite state automata, whose alphabet is Σ.

Definition 1: Given $G = (X, \Sigma, \xi, x_0, X_m)$, a *bisimulation* relation on X is an equivalence relation $R \subseteq X \times X$ such that for each $(x, x') \in R$ and $s \in \Sigma^*$, $\xi(x, s) \neq \varnothing$ implies $\xi(x', s) \neq \varnothing$ and for any $y \in \xi(x, s)$,

$$(\exists y' \in \xi(x', s))\, (y, y') \in R \wedge [y \in X_m \iff y' \in X_m]$$

The largest bisimulation relation is called *bisimilarity* on X, written as $\sim$. □

Definition 2: Given $G = (X, \Sigma, \xi, x_0, X_m)$, let $\Sigma' \subseteq \Sigma$ and $P : \Sigma^* \to \Sigma'$ be the natural projection. A *weak bisimulation* relation on X with respect to Σ' is an equivalence relation $R \subseteq X \times X$ such that for each $(x, x') \in R$ the following condition holds: for any $s \in \Sigma^*$, $\xi(x, s) \neq \varnothing$ implies that there exists $s' \in \Sigma^*$ with $P(s) = P(s')$ and $\xi(x', s') \neq \varnothing$ such that for any $y \in \xi(x, s)$,

$$(\exists y' \in \xi(x', s'))\, (y, y') \in R \wedge [y \in X_m \iff y' \in X_m]$$

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

For each $x \in X$ let $< x >:= \{x' \in X | (x, x') \in \approx_{\Sigma'}\}$, and $X/\approx_{\Sigma'} := \{< x > | x \in X\}$.

Definition 3: An *automaton abstraction* with respect to Σ and Σ' is a map $\kappa : \phi(\Sigma) \to \phi(\Sigma')$ such that any element $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$ is mapped to $\kappa(G) = (X', \Sigma', \xi', x'_0, X'_m) \in \phi(\Sigma')$, where

1) $X' = X/\approx_{\Sigma'}$
2) $x'_0 =< x_0 > \in X'$
3) $X'_m = \{< x > \in X' | < x > \cap X_m \neq \varnothing\}$
4) $\xi' : X' \times \Sigma' \to 2^{X'}$, where for any $< x > \in X'$ and $\sigma \in \Sigma'$,
$$\xi'(< x >, \sigma) := \{< x' > \in X' | (\exists y \in < x >, \\ y' \in < x' >)(\exists u, u' \in (\Sigma - \Sigma')^*)\, y' \in \xi(y, u\sigma u')\}$$

□

What κ does on G is simply to create transitions among elements of $X/\approx_{\Sigma'}$. If we use $|X|$ to denote the size of X, then $|X'| \leq |X|$. The time complexity of computing $\kappa(G)$ is mainly due to the computation of X', which is shown in [17] to be $O((mn^2 + l)\log n)$, where $l = |X_m|$, $n = |X|$ and m is the number of transitions in G.

As an illustration, suppose a standardized automaton $G \in \phi(\Sigma)$ is depicted in Figure 1, where $\Sigma = \{\tau, a, b, c, u\}$. We

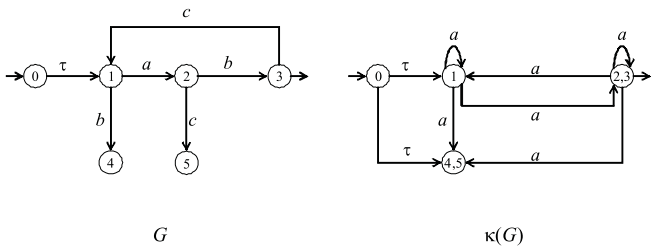

Fig. 1. A Standardized Automaton G and Abstraction $\kappa(G)$

take $\Sigma' = \{\tau, a\}$. Then we can show that

$$X' = X/\approx_{\Sigma'} = \{< 0 >= \{0\}, < 1 >= \{1\}, \\ < 2 >= \{2, 3\}, < 4 >= \{4, 5\}\}$$

The abstraction $\kappa(G)$ is depicted in Figure 1.

In the definition of $\kappa(G)$, if we do not introduce the event τ (thus, the standardized automata), it will be difficult for us to deal with the situation where there is a blocking state x reachable from the initial state x_0 via a 'silent' path $s \in (\Sigma - \Sigma')^*$, i.e. $x \in \xi(x_0, s)$, such that any path s' from x, i.e. $\xi(x, s') \neq \varnothing$, is also silent, e.g. the blocking state 5 in Figure 1. It is also difficult for us to deal with marker states that are reachable from x_0 via silent paths. With the newly introduced event τ, we can easily solve the problems, as we have done in Figure 1. Besides, since τ is uncontrollable and unobservable, introducing it will not affect the existence of a nonblocking supervisor. Next, we discuss how to perform distributed supervisor synthesis.

III. Distributed Supervisor Synthesis

We first describe how to synthesize a nonblocking supervisor based on automaton abstraction. Then we provide a distributed supervisor synthesis procedure.

A. Supervisor Synthesis over Nondeterministic Automata

In the RW paradigm a plant model $G \in \phi(\Sigma)$ and a specification $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$ are both deterministic. In this paper the plant model G may be nondeterministic. We would like to ask whether we can still synthesize a supervisor S in terms of a deterministic automaton such that controllability, observability and nonblockingness are attainable. The motivation of requiring S to be deterministic is that S cannot distinguish the nondeterminism in G from external observation sequences, thus it will apply the same control action on states in G that are reachable by the same string. Here the specification H will remain deterministic, and it is not necessarily standardized, namely it is possible that $\tau \notin \Delta$ because abstraction will never be applied to H.

To answer our question, we first redefine the concepts of controllability and observability in the automaton framework. Let $G = (X, \Sigma, \xi, x_0, X_m)$. For each $x \in X$ let

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

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

Definition 4: Given $\hat{G} = (\hat{X}, \Sigma, \hat{\xi}, \hat{x}_0, \hat{X}_m)$, we say $\hat{G}$ is *f-homomorphic* to G, denoted as $\hat{G} \rightsquigarrow_f G$, if there is a mapping $f : \hat{X} \to X$ such that the following hold:

1) $f(\hat{x}_0) = x_0$
2) $f(\hat{X}_m) \subseteq X_m$
3) For any $\hat{x}_1, \hat{x}_2 \in \hat{X}$ and $\sigma \in \Sigma$,

$$\hat{x}_2 \in \hat{\xi}(\hat{x}_1, \sigma) \Rightarrow f(\hat{x}_2) \in \xi(f(\hat{x}_1), \sigma)$$

If f is bijective and $G \rightsquigarrow_{f^{-1}} \hat{G}$, then G is called *isomorphic* to G, denoted as $\hat{G} \equiv G$. □

Definition 5: Given $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$, where $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, suppose there is another automaton $A = (W, \Sigma, \varsigma, w_0, W_m) \in \phi(\Sigma)$ such that $A \rightsquigarrow_f G$. Then A is *controllable* with respect to G, f and Σ_{uc} if for any $w \in W$, $F_A(w) \cap E_G(f(w)) \subseteq \Sigma_c$. □

Def. 5 is the state interpretation of the concept of controllability in the RW paradigm, saying that at any state w in A (which can be interpreted as representing an objective behavior), a transition σ that is disallowed by A (i.e. $\sigma \in F_A(w)$) but allowed by G (i.e. $\sigma \in E_G(f(w))$) must be controllable (i.e. $\sigma \in \Sigma_c$). In other words, no uncontrollable event can be disallowed in A, if A is controllable. We now introduce the concept of observability.

Definition 6: Given $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$, where $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, suppose there is another automaton $A = (W, \Sigma, \varsigma, w_0, W_m) \in \phi(\Sigma)$ such that $A \rightsquigarrow_f G$. Then A is *observable* with respect to G, f and the natural projection $P : \Sigma^* \to \Sigma_o^*$ if the following condition holds: for any $w, w' \in W$ if there exist $s, s' \in \Sigma^*$ with

$$w \in \varsigma(w_0, s) \wedge w' \in \varsigma(w_0, s') \wedge P(s) = P(s')$$

then we have $(F_A(w) \cap E_G(f(w))) \cap E_A(w') \cup (F_A(w') \cap E_G(f(w'))) \cap E_A(w) = \varnothing$. □

What Def. 6 says is that, if A is observable then for any two states w and w' reachable by two strings s and s' having the same projected image (i.e. $P(s) = P(s')$), there is no event allowed at w but disallowed at w' (i.e. $(F_A(w') \cap E_G(f(w'))) \cap E_A(w)$) and vice versa (i.e. $(F_A(w) \cap E_G(f(w))) \cap E_A(w')$). Notice that, if $\Sigma_o = \Sigma$, namely every event is observable, A may still not be observable, owing to nondeterminism. The state-based definitions of controllability and observability bear similarity to those defined in [12]. We now introduce the concept of nonblocking supervisors.

Definition 7: A deterministic finite-state automaton $S = (Y, \Sigma, \eta, y_0, Y_m) \in \phi(\Sigma)$ is a *nonblocking supervisor* of $G = (X, \Sigma, \xi, x_0, X_m) \in \phi(\Sigma)$ with respect to a specification $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$ under the natural projection $P_o : \Sigma^* \to \Sigma_o^*$, where $\Sigma_o \subseteq \Sigma$, if the following hold:

1) $N(G \times S) \subseteq N(G \times H)$
2) $B(G \times S) = \varnothing$
3) $G \times S$ is controllable with respect to G, f_0 and Σ_{uc}
4) $G \times S$ is observable with respect to G, f_0 and P_o

where $f_0 : X \times Y \to X : (x, y) \mapsto f_0(x, y) := x$. □

The first condition of Def. 7 says that the closed-loop behavior (CLB) satisfies the specification H and the second one says CLB must be nonblocking. The third and fourth ones are self-explanatory. We have the following result.

Proposition 1: Given $G \in \phi(\Sigma)$ and a deterministic automaton $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$, $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ and $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, there exists a nonblocking supervisor $S \in \phi(\Sigma)$ of G with respect to H if and only if there exists $A \in \phi(\Sigma)$ such that $A \rightsquigarrow_f G \times H$, A is controllable with

respect to G, f and Σ_{uc}, A is observable with respect to G, f and the natural projection $P : \Sigma^* \to \Sigma_o^*$, and $B(A) = \varnothing$.□

The proof of Prop. 1, which is not included in this paper owing to the page limit, indicates that S is simply the canonical recognizer of $N(A)$, when A is controllable, observable and $B(A) = \varnothing$. Supervisory control of nondeterministic finite-state automata has been explored recently in the literature, e.g. [19] [20]. In this paper we focus on distributed supervisor synthesis based on abstraction, which is described in the following section.

B. Aggregate Distributed Supervisor Synthesis

Definition 8: A *distributed system* with respect to the given alphabets $\{\Sigma_i | i \in I\}$ is a collection of nondeterministic finite-state automata $\mathcal{G} := \{G_i = (X_i, \Sigma_i, \xi_i, x_{i,0}, X_{i,m}) \in \phi(\Sigma_i) | i \in I\}$, where each G_i ($i \in I$) is called the i^{th} *component* of $\mathcal{G}$. Each Σ_i ($i \in I$) is partitioned into a *controllable* alphabet $\Sigma_{i,c}$ and an *uncontrollable* alphabet $\Sigma_{i,uc}$, namely $\Sigma_i = \Sigma_{i,c} \dot\cup \Sigma_{i,uc}$. It is also partitioned into an *observable* alphabet $\Sigma_{i,o}$ and an *unobservable* alphabet $\Sigma_{i,uo}$, namely $\Sigma_i = \Sigma_{i,o} \dot\cup \Sigma_{i,uo}$. □

We assume that for any $i, j \in I$,

$$i \neq j \Rightarrow \Sigma_{i,c} \cap \Sigma_{j,uc} = \varnothing \wedge \Sigma_{i,o} \cap \Sigma_{j,uo} = \varnothing$$

In other words, there is no event, which is controllable in G_i but uncontrollable in G_j ($i \neq j$). There is also no event, which is observable in G_i but unobservable in G_j ($i \neq j$). Let $\Sigma := \cup_{i\in I}\Sigma_i$, $\Sigma_c := \cup_{i\in I}\Sigma_{i,c}$, $\Sigma_{uc} := \cup_{i\in i}\Sigma_{i,uc}$, $\Sigma_o := \cup_{i\in i}\Sigma_{i,o}$ and $\Sigma_{uo} := \cup_{i\in i}\Sigma_{i,uo}$. Let J be an index set and $\{\Delta_j \subseteq \cup_{i\in I}\Sigma_i | j \in J\}$ be a collection of alphabets. Suppose we have a set of specifications $\mathcal{H} = \{H_j \in \phi(\Delta_j) | j \in J\}$, where each H_j is a deterministic automaton. We want to synthesize a collection of deterministic automata

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

where K is an index set, such that $\times_{k\in K} S_k$ is a nonblocking supervisor of $\times_{i\in I} G_i$ with respect to $\times_{j\in J} H_j$. Later we will call such an $\mathcal{S}$ a nonblocking distributed supervisor of $\mathcal{G}$ with respect to $\mathcal{H}$, and each S_i is called a local (nonblocking) supervisor.

We first consider a 2-component distributed system $\mathcal{G} = \{G_1, G_2\}$. Then we extend it to a general distributed system. Suppose there exists a specification $H \in \phi(\Delta)$, where $\Delta \subseteq \Sigma_1 \cup \Sigma_2$. Let $\Sigma' \subseteq \Sigma_1 \cup \Sigma_2$ such that $\Sigma_1 \cap (\Sigma_2 \cup \Delta) \subseteq \Sigma'$. Let $\kappa_1 : \mathcal{G}(\Sigma_1) \to \mathcal{G}(\Sigma_1 \cap \Sigma')$ be the automaton abstraction, and $A_1 := \kappa_1(G_1)$. We now use A_1 as an abstraction of G_1 with respect to G_2, and treat $A_1 \times G_2$ as the plant model. We have the following result.

Proposition 2: Suppose there exists a nonblocking supervisor $S \in \phi(\Gamma)$ of $A_1 \times G_2$ with respect to H, where $\Gamma = \Sigma_2 \cup \Sigma'$. Then S is also a nonblocking supervisor of $G = G_1 \times G_2$ with respect to H. □

Proposition 2 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 a Distributed Supervisor (ASDS)

1) Initially set $T_1 := \Sigma_1$, $Q_1 := \varnothing$ and $W_1 := G_1$.
2) For $k = 2, \cdots, n$,
   a) Set $I_k := \{k, k+1, \cdots, n\}$ and
   $$J_k := \{j \in J | \Delta_j \subseteq T_{k-1} \cup \Sigma_k\} - Q_{k-1}$$
   b) Let $\Sigma_{I_k} := \cup_{i\in I_k}\Sigma_i$ and $\Theta_k := \cup_{j\in J-Q_{k-1}}\Delta_j$.
   c) If $J_k \neq \varnothing$, define $V_k := \times_{j\in J_k} H_j$. Otherwise, set V_k as the canonical recognizer of Σ_k^* (thus, V_k imposes no constraint).
   d) Let $A_{k-1} := \kappa_k(W_{k-1})$ with the abstraction
   $$\kappa_{k-1} : \phi(T_{k-1}) \to \phi(\Sigma_{A_{k-1}})$$
   where $(\Sigma_{I_k} \cup \Theta_k) \cap T_{k-1} \subseteq \Sigma_{A_{k-1}} \subseteq T_{k-1}$.
   e) Synthesize a nonblocking supervisor $S_k \in \phi(\Gamma_k)$ of $A_{k-1} \times G_k$ with respect to V_k, where $\Gamma_k := \Sigma_{A_{k-1}} \cup \Sigma_k$. If such an S is not empty, go to Step (f). Otherwise, terminate.
   f) Let $T_k := \Sigma_{A_{k-1}} \cup \Sigma_k$, $Q_k := Q_{k-1} \cup J_k$ and $W_k := A_{k-1} \times G_k \times S_k$. □

To explain ASDS, 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 and G_2 in the sense that $\Delta_i \subseteq \Sigma_1 \cup \Sigma_2$ for $i = 1, 2, \cdots, m$, and the remaining specifications $H_{m+1}, H_{m+2}, \cdots, H_r$ 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 $j = m+1, 2, \cdots, r$. What ASDS does is as follows. First, it computes a nonblocking supervisor S_2 to make sure that $G_1 \times G_2 \times S_2$ satisfies the specification $V_2 = H_1 \times \cdots \times H_m$. When $\{H_1, \cdots, H_m\} = \varnothing$, ASDS simply sets V_2 to be the canonical recognizer of Σ_2^*. 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 is created. The alphabet Σ_{A_1} is chosen by whatever convenient reasons, as long as the condition $\Sigma_1 \cap (\Sigma_2 \cup \Sigma_3 \cup_{i=1}^r \Delta_i) \subseteq \Sigma_{A_1} \subseteq \Sigma_1$ holds. The reason of imposing this condition is that in the subsequent computation we can always use A_1 to replace G_1. 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, two 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. By the previous description we can compute a nonblocking supervisor S_2

satisfying V_2. Suppose S_2 exists, then we can create a new plant $W_2 = A_1 \times G_2 \times S_2$ which is nonblocking and satisfies $V_2 = H_1 \times \cdots \times H_m$. We want to synthesize a nonblocking supervisor S_3 such that $W_2 \times G_3 \times S_3$ satisfies $V_3 = H_{m+1} \times \cdots \times H_r$. To achieve this, we create an abstraction A_2 of W_2, and repeat the same procedure as we have done for computing S_2.

When ASDS terminates at $k = n$, the collection of local nonblocking supervisors

$$\mathcal{S} = \{S_k \in \phi(\Gamma_k) | k \in K = \{2, 3, \cdots, n\}\}$$

is called a *distributed supervisor* of (the plant)

$$\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I = \{1, 2, \cdots, n\}\}$$

with respect to the ordering $(G_1, \cdots, G_n)$ under the collection of specifications $\mathcal{H} = \{H_j \in \phi(\Gamma_j)) | j \in J\}$. Let $G := \times_{i \in I} G_i$, $S := \times_{k \in K} S_k$ and $H := \times_{j \in J} H_j$. We have the following result.

Proposition 3: Let $\mathcal{S} = \{S_k | k \in K\}$ be computed by ASDS. Then S is a nonblocking supervisor of G with respect to H. □

The proof of Prop. 3 uses induction on k with Prop. 2. Although during the above construction $\mathcal{S}$ contains $|I| - 1$ local supervisors, several may not impose any control on the system. This can be checked whenever a local supervisor S_k is computed. In that case we simply remove that local supervisor 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 ASDS. How to choose a good ordering is an interesting problem, which hopefully can be addressed in our future papers.

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 $\{S_2, \cdots, S_n\}$ 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 $r = \max\{2, k\}$. 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.

Finally, we use a concrete example to illustrate the techniques developed in the previous sections. Suppose we have three processing units, whose behaviors are depicted in Figure 2. Each robot G_i $(i = 1, 2, 3)$ has the following standard operations: (1) fetch a work piece from a source (a_i); (2) put it to a sink (b_i); (3) preprocess (c_i); (4) postprocess (d_i). To produce one piece of product, three work pieces are needed, one for each robot to go through those standard steps. When the robot G_1 preprocesses a work piece, there is a slight chance that it may get stuck (f_1). Among each alphabet Σ_i, the controllable alphabet is $\Sigma_{i,c} = \{c_i\}$, and for simplicity suppose the observable alphabet $\Sigma_{i,o}$ is Σ_i, namely every event is observable. There are two local specifications depicted in Figure 2, saying that the robot G_1 must preprocess a work piece if G_2 does; and G_2 must preprocess a work piece if G_3 does. We now start to synthesize a distributed supervisor by using ASDS.

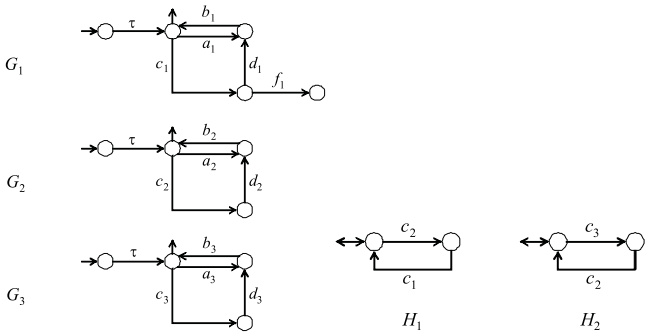

Fig. 2. Processing Units$\{G_1, G_2, G_3\}$ and Specifications $\{H_1, H_2\}$

First, we create an appropriate abstraction of G_1. Since $\Sigma_1 \cap (\Sigma_2 \cup \Sigma_3 \cup \Delta_1 \cup \Delta_2) = \{\tau, c_1\}$, we simply choose $\Sigma_{A_1} = \{\tau, c_1\}$, which contains all controllable events available to G_1 as well. In reality, we may also want to include all observable events in Σ_{A_1} so that the abstraction $\kappa_1 : \phi(T_1) \to \phi(\Sigma_{A_1})$ can capture all possible observations as well. In this example, we do not consider observations because every event is observable (except for f_1). The abstraction $A_1 := \kappa_1(G_1)$ is depicted in Figure 3. We now use $A_1 \times G_2$ as the overall model, which is depicted in Figure 3. Since $\Delta_1 \subseteq \Sigma_1 \cup \Sigma_2$, we use H_1 as the specification. Clearly, the event c_1 must be disabled at states $1, 2, 3$. Otherwise, the blocking state 5 will be reached. The event c_2 at state 1 should also be disabled. Otherwise, the specification H_1 will be violated. The nonblocking local supervisor S_2 of $A_1 \times G_2$ with respect to H_1 is depicted in Figure 3, where the alphabet of S_2 is $\Gamma_2 = \{\tau, a_2, b_2, c_1, c_2, d_2\}$. We now use $A_1 \times G_2 \times S_2$ as a new plant, and create an abstraction A_2. Since $(\Sigma_{A_1} \cup \Sigma_2) \cap (\Sigma_3 \cup \Delta_2) = \{\tau, c_2\}$, we choose the alphabet $\Sigma_{A_2} = \{\tau, c_2\}$. The abstraction $A_2 = \kappa_2(A_1 \times G_2 \times S_2)$ is depicted in Figure 4. We can see that the new plant model $A_2 \times G_3$ has the same transition structure as that of G_3. But the alphabet of

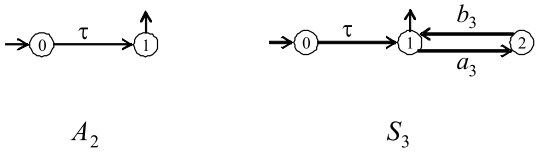

Fig. 4. A_2 and Nonblocking Local Supervisor S_3 of $A_2 \times G_3$ w.r.t. H_2

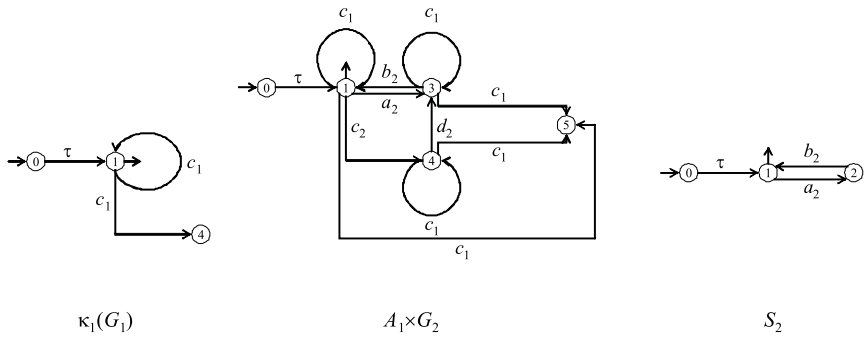

Fig. 3. Automata of A_1, $A_1 \times G_2$ and Nonblocking Local Supervisor S_2 of $A_1 \times G_2$ w.r.t. H_1

$A_2 \times G_3$ is $\Sigma_3 \cup \{c_2\}$. We now use H_2 as the specification. Clearly, the event c_3 must be disabled because, otherwise, the specification H_2 will be violated. The final nonblocking local supervisor S_3 is depicted in Figure 4, where the alphabet of S_3 is $\Gamma_3 = \Sigma_3 \cup \{c_2\} = \{\tau, a_3, b_3, c_1, c_3, d_3\}$. We can check that $\{s_1, s_2\}$ is a nonblocking distributed supervisor of $\{G_1, G_2, G_3\}$ with respect to specifications $\{H_1, H_2\}$. We can verify that the maximum number of states of any intermediate computational result in terms of automata is 5 states, which occurs when we compute $A_1 \times G_2$. Clearly, abstractions help to reduce the computational complexity in this example because otherwise we will have to face the product $G_1 \times G_2 \times G_3$ directly, which has 36 states.

IV. Conclusions

In this paper we introduce a new technique for automaton abstraction and present an aggregate approach for distributed supervisor synthesis based on abstractions of nondeterministic finite-state automata. The main advantage of this approach is that only local computation is involved, thus, high complexity incurred by synchronous product of a large number of components can be avoided. Besides, we can achieve a certain level of implementation flexibility in terms of attaining reusability of some local supervisors when the structure of a target system changes.

References

[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] W.M. Wonham and P.J. Ramadge. Modular supervisory control of discrete event systems. *Maths. of Control, Signals & Systems*, 1(1):13–30, 1988.

[4] K. Rudie and W.M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transanctions on Automatic Control*, 37(11):1692–1708, 1992.

[5] T.S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 12(3):335–377, 2002.

[6] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-eventsystems. *IEEE Trans. Automatic Control*, 35(10):1125–1134, 1990.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] S.Takai, T. Ushio, and S. Kodama. Static-state feedback control of discrete event systems under partial observations. *IEEE Trans. Automatic Control*, 40(11):1950–1954, 1994

[13] K.C. Wong and W.M. Wonham. On the computation of observers in discrete-event systems. *Discrete Event Dynamic Systems*, 14(1):55-107, 2004.

[14] 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.

[15] 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.

[16] R. Su, J.H. van Schuppen and J.E. Rooda. Supervisor Synthesis based on Abstractions of Nondeterministic Automata. In *Proc. 9th International Workshop on Discrete Event Systems (WODES08)*, pages 412–418, 2006.

[17] R. Su, J.H. van Schuppen and J.E. Rooda. *Model Abstraction of Nondeterministic Finite State Automata in Supervisor Synthesis*. SE Report No. 2008-03, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Technonology, Eindhoven, The Netherlands, 2008. ISSN 1567-1872. URL: http://se.wtb.tue.nl/sereports.

[18] 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, 2006.

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

[20] C. Zhou, R. Kumar and S. Jiang. Control of nondeterministic discrete event systems for bisimulation equivalence. *IEEE Trans. Automatic Control*, 51(5):754-765, 2006.