Proceedings of the 9th International
Workshop on Discrete Event Systems
Göteborg, Sweden, May 28-30, 2008

Supervisor Synthesis based on Abstractions of Nondeterministic Automata

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, especially for systems of industrial size. Owing to the high computational complexity, synthesizing a nonblocking supervisor for a large scale system is never easy, if still possible. In this paper we attempt to solve this synthesis problem by using abstractions. We first introduce a new abstraction operation, which preserves the nonblocking property. Then we describe how to synthesize supervisors by using abstractions of relevant automata so that the high computational complexity associated with the synchronous product described in the Ramadge-Wonham supervisory control theory (SCT) may be avoided.**

***Index Terms*— discrete-event systems, nondeterministic finite-state automata, automaton abstraction, supervisor 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 the well known journal papers [1], [2]. Since then there has been a large volume of literature under the same paradigm but with different architectural setups, e.g. [3], [4] on modular control, [5], [6] on decentralized control, [7], [8] on hierarchical control, and [12] on timed discrete event systems. In the RW paradigm the main difficulty of supervisor synthesis for a system of industrial size is to achieve nonblockingness. The reason is that the size of a plant model increases quickly when the number of local components increases, due to the synchronous product which incurs cartesian product over automata. To overcome this difficulty, some authors attempt to introduce sufficient conditions, which allow local supervisor synthesis. For example, in [3] the authors propose the concept of *modularity*. When local supervisors are modular, a globally nonblocking supervisory control is achieved. Nevertheless, testing modularity itself usually imposes prohibitive computational complexity. Another notable work is presented in [9], where, by imposing *interface consistency* and *level-wise controllability* among subsystems and local supervisors in a hierarchical setup, a very large nonblocking control problem may be solved, e.g. the system size reaches 10^{21} in the AIP example [9]. But those imposed consistency properties seem to us rather peculiar and strong, and the approach does not tell how to deliberately and systematically design interfaces that allow synthesis of local supervisors that satisfy those properties, as mentioned in [10]. In [11] the authors present an interesting approach, which is aimed to synthesize a state-based supervisor. By introducing the concept of *state tree structures*, the authors propose to represent states in an efficient way, upon which the power of symbolic computation (as manifested by the manipulation of binary decision diagrams (BDDs)) is fully utilized. It has been shown in [11] that a system with 10^{24} states can be well handled. Nevertheless, this approach is essentially a centralized approach and only full observation is under consideration.

In this paper we will discuss how to synthesize a supervisor by using an appropriate abstraction of a target system. The idea of abstraction has been known in the literature, e.g. in [13] abstraction is used in the modular and hierarchical supervisor synthesis, and in [14] for decentralized control. Nevertheless, to make their approaches work, natural projections have to possess the *observer* property [8], which may not always hold by a natural projection. Although a natural projection can always be converted to an observer (with respect to a specific language) [15], such a modification may force many other originally unobservable events to be observable in order to achieve the observer property, resulting in a projected image not small enough to allow supervisor synthesis for a large-scale system. In our approach abstraction is made by applying a newly-defined operation, which preserves the nonblocking property in the sense that 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. In [16] the authors propose a reduction based on *weak bisimilarity*, which serves a similar purpose - to preserve the nonblocking property. But it turns out that weak bisimilarity is too strong to allow us to generate a sufficiently small projected image, which is crucial for the success of supervisor synthesis for a system of industrial size. Our newly-defined abstraction in this paper is tailored for supervisory control, thus can yield a much smaller result than what the reduction based on weak bisimilarity can achieve. Based on this abstraction we will show that supervisor synthesis can be achieved in a local fashion in the sense that a nonblocking supervisor for an abstraction $\kappa(G)$ is also a nonblocking supervisor for G.

This paper is organized as follows. In Section II we introduce automaton composition and abstraction over nondeterministic automata. Then we present a supervisor synthesis

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

978-1-4244-2593-8/08/$25.00 ©2008 IEEE

problem based on nondeterministic automata in Section III. After an illustration example in Section IV, conclusions are stated in Section V.

II. Automaton Composition and Nonblocking-Preserving Abstraction

In the following sections we follow the notations used in [17]. The main goal of this section is to introduce several operations that will be extensively used in supervisor synthesis, and provide a few properties associated with those operations. Given an alphabet Σ, let $\phi(\Sigma)$ be the collection of all nondeterministic finite-state automata. 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 can see that any string $s \in B(G)$ can lead to a state x, from which no marker state is reachable. Therefore, we call $B(G)$ the *blocking set* of G. Similarly, we can 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 the reachability part of the product of G_1 and G_2. The transition map $\xi_1 \times \xi_2$ is extended to $X_1 \times X_2 \times (\Sigma_1 \cup \Sigma_2)^* \to 2^{X_1 \times X_2}$.

Given an automaton $G \in \phi(\Sigma)$ which serves as a plant model, we usually want to synthesize a supervisor $S \in \phi(\Sigma')$ with $\Sigma' \subseteq \Sigma$ such that the overall closed-loop system $G \times S$ satisfies a given specification $H \in \phi(\Sigma'')$ with $\Sigma'' \subseteq \Sigma$. When the plant model G is too big to be handled directly, we usually hope to create a simplified model $G' \in \phi(\Sigma''')$ of G with $\Sigma' \cup \Sigma'' \subseteq \Sigma''' \subseteq \Sigma$, which is called an *abstraction* of G, such that a nonblocking supervisor S computed based on such an abstraction will also be a nonblocking supervisor of G itself. To achieve this purpose, we want to make sure that, $G \times S$ is nonblocking if and only if $G' \times S$ is nonblocking. We now propose the following procedure to create such an abstraction G'.

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

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

For an ordinary automaton $G = (X, \Sigma, \xi, x_0, X_m)$ we can convert it into a standardized automaton by simply: (1) extend the alphabet to $\Sigma \cup \{\tau\}$; (2) add a new state x_0'; (3) define a new transition map ξ' such that $\xi'(x_0', \tau) = \{x_0\}$ and for any $(x, \sigma) \in X \times \Sigma$ we have $\xi'(x, \sigma) = \xi(x, \sigma)$. The resulting automaton $G' = (X \cup \{x_0'\}, \Sigma \cup \{\tau\}, \xi', x_0', X_m)$ is a standardized automaton. From now on we assume that every alphabet Σ contains τ, and $\phi(\Sigma)$ is the collection of all standardized finite state nondeterministic automata, whose alphabet is Σ. The role of τ will be explained shortly.

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 an element $\kappa(G) = (X_2, \Sigma', \xi_2, x_0, X_{2,m}) \in \phi(\Sigma')$ as follows:

(1) Let $G_1 = (X_1, \Sigma', \xi_1, x_0, X_{1,m}) \in \phi(\Sigma')$, where

- If there are $u \in (\Sigma - \Sigma')^*$ and $x \in X$ such that there exists $x'' \in \xi(x, u)$ with the following property,
  $$(\forall s \in \Sigma^*)\, \xi(x'', s) \neq \varnothing \Rightarrow \\ s \in (\Sigma - \Sigma')^* \wedge \xi(x'', s) \cap X_m = \varnothing \ (*)$$
  then create a new state $z \notin X$ and set $X_1 := X \cup \{z\}$. Otherwise set $X_1 := X$. Let $X_{1,m} := X_m$.
- Let $\xi_1 : X_1 \times \Sigma' \to 2^{X_1}$ be the transition map where for any $x \in X$ and $\sigma \in \Sigma'$, $x' \in \xi_1(x, \sigma)$ if there exist $u, u' \in (\Sigma - \Sigma')^*$ such that one of the following holds:
  (a) $x' \in \xi(x, u\sigma u') \wedge x' \in X_m$
  (b) $x' \in \xi(x, u\sigma u') \wedge (\exists \sigma' \in \Sigma')\, \xi(x', \sigma') \neq \varnothing$
  (c) $x' = z \wedge (\exists x'' \in \xi(x, u\sigma u'))$ the property (*) holds.

(2) Let $\kappa(G) = (X_2, \Sigma', \xi_2, x_0, X_{2,m}) \in \phi(\Sigma')$ with

- $X_2 := \{x \in X_1 | (\exists s \in \Sigma'^*)\, x \in \xi_1(x_0, s)\}$
- $X_{2,m} := \{x \in X_{1,m} | (\exists s \in \Sigma'^*)\, x \in \xi_1(x_0, s)\}$
- $\xi_2 : X_2 \times \Sigma' \to 2^{X_2}$, where for each $(x, \sigma) \in X_2 \times \Sigma'$,
  $$\xi_2(x, \sigma) := \{x' \in X_2 | x' \in \xi_1(x, \sigma)\}$$

$\kappa(G)$ is actually a reachable sub-automaton of G_1. □

Suppose the size of a set A is denoted as $|A|$. Then we have $|X_1| \leq |X| + 1$ because we create at most one new state z. Furthermore, $|X_2| \leq |X_1|$. Thus, $|X_2| \leq |X| + 1$.

As an illustration, suppose a standardized automaton $G \in \phi(\Sigma)$ is depicted in Figure 1, where $\Sigma = \{\tau, a, b, c, u\}$. We take $\Sigma' = \{\tau, b\}$. Then based on the above procedure, $G_2 = \kappa(G)$ is depicted in Figure 2, where the state numbers correspond to those in Figure 1. Notice that there is a new state z in G_2, which is created because we can take $x'' = 6$ and the property (*) holds. In this example we can see that, whenever there is a string s in G, there is a string $P(s)$

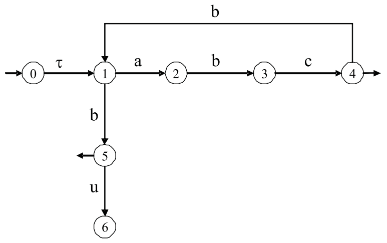

Fig. 1. A Standardized Automaton G

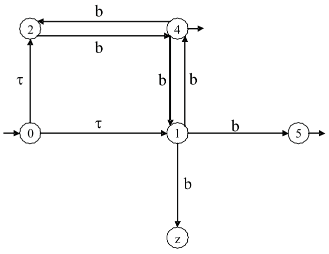

Fig. 2. The Automaton $G_2 = \kappa(G)$

in $\kappa(G)$, where $P : \Sigma^* \to \Sigma'^*$ is the natural projection. Furthermore, if s reaches a nonblocking state in G, then $P(s)$ can reach a nonblocking state in $\kappa(G)$ as well; if s reaches a blocking state in G, so can $P(s)$ in $\kappa(G)$. For example, in G the string $s = \tau abc$ reaches the marker state 4, and in $\kappa(G)$ the string $t = \tau b$ can also reach the marker state 4; the string $s = \tau abcbbu$ reaches the blocking state 6, and in $\kappa(G)$ the string $t = \tau bbb$ can reach a blocking state z as well. Similarly, for any string t in $\kappa(G)$, there is a string $s \in P^{-1}(t)$ such that, if t reaches a nonblocking (or blocking) state in $\kappa(G)$ then s can reach a nonblocking (or blocking) state in G as well.

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 6 in Figure 1. It is also difficult for us to deal with marker states that are reachable from x_0 via silent paths, e.g. the state 5 in Figure 1. With the newly introduced event τ, we can easily solve the problems, as we have done in Figure 2. Besides, since τ is uncontrollable and unobservable, introducing it will not affect the existence of a nonblocking supervisor.

Next, we want to answer the question whether $G \times S$ for some automaton S is nonblocking if and only if $\kappa(G) \times S$ is nonblocking. To this end we need the following concepts.

Given an automaton $G = (X, \Sigma, \xi, x_0, X_m)$, let $h(G) = (X_h, \Sigma_h, \xi_h, x_0, X_m)$ be an automaton such that

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

(b) $\Sigma_h := \{\sigma \in \Sigma | (\exists x, x' \in X_h)\, x' \in \xi(x, \sigma)\}$

(c) $\xi_h : X_h \times \Sigma_h \to 2^{\Sigma_h} : (x, \sigma) \mapsto \xi_h(x, \sigma) := \xi(x, \sigma) \cap X_h$

The sub-automaton $h(G)$ is simply the largest reachable and coreachable part of G.

Definition 1: 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) $f(\hat{X}_m) = X_m$
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 injective then $\hat{G}$ is called *f-monomorphic* to G, denoted as $\hat{G} \rightsquigarrow_f^m G$. 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 2: A *bisimulation* relation on X of $G = (X, \Sigma, \xi, x_0, X_m)$ is an equivalence relation $R \subseteq X \times X$ such that for each $(x, x') \in R$ and $s \in \Sigma^*$, if $\xi(x, s) \neq \varnothing$ then $\xi(x', s) \neq \varnothing$ and for any $y \in \xi(x, s)$, there exists $y' \in \xi(x', s)$ such that

$$(y, y') \in R \wedge [y \in X_m \iff y' \in X_m]$$

The largest bisimulation relation is called *bisimilarity*, written as $\sim_R$. □

Definition 3: The *quotient* of G with respect to $\sim_R$ is another automaton $G/\sim_R := (Y, \Sigma, \eta, y_0, Y_m)$ where

1) $Y := X/\sim_R$, the quotient set of X w.r.t. $\sim_R$
2) $y_0 := [x_0] \in Y$, the coset of x_0 w.r.t. $\sim_R$
3) $Y_m := \{y \in Y | y \cap X_m \neq \varnothing\}$
4) $\eta : Y \times \Sigma \to 2^Y$, where for any $(y, \sigma) \in Y \times \Sigma$,

$$\eta(y, \sigma) := \{y' \in Y | (\exists x \in y)\, \xi(x, \sigma) \cap y' \neq \varnothing\}$$

□

With Definitions 1-3 we introduce the following concept, which will be extensively used in the reminder of this paper.

Definition 4: Given two automata $G_1, G_2 \in \phi(\Sigma)$, we say G_1 is *nonblocking preserving* with respect to G_2, denoted as $G_1 \sqsubseteq_h G_2$, if $B(G_1) \subseteq B(G_2)$ and $h(G_1)/\sim_R \equiv h(G_2)/\sim_R$. G_1 is *nonblocking equivalent* to G_2, denoted as $G_1 \cong_h G_2$, if $G_1 \sqsubseteq_h G_2$ and $G_2 \sqsubseteq_h G_1$.□

Definition 5: An automaton $G = (X, \Sigma, \xi, x_0, X_m)$ is *marking aware* with respect to $\Sigma' \subseteq \Sigma$, if for any $x \in X$, where $\xi(x, \sigma) \neq \varnothing$ for some $\sigma \in \Sigma'$, the following holds:

$$(\forall s \in \Sigma^*)\, \xi(x, s) \cap X_m \neq \varnothing \Rightarrow P(s) \neq \epsilon$$

where $P : \Sigma^* \to \Sigma'^*$ is the natural projection. □

Proposition 1: Given $G_1, G_2 \in \phi(\Sigma), G_3 \in \phi(\Sigma')$, if $G_1 \sqsubseteq_h G_2$ then $G_1 \times G_3 \sqsubseteq_h G_2 \times G_3$. □

Prop. 1 says that, nonblocking preserving (or equivalence) is invariant with respect to automaton product.

Proposition 2: Let $G \in \phi(\Sigma)$, $\Sigma' \subseteq \Sigma$, $P : \Sigma^* \to \Sigma'^*$ be the natural projection and $\kappa : \phi(\Sigma) \to \phi(\Sigma')$ the abstraction. Then $P(B(G)) = B(\kappa(G))$ and $P(N(G)) = N(\kappa(G))$. □

What Prop. 2 says is illustrated by the commutative diagram in Figure 3, where the natural projection P is extended to $P : 2^{\Sigma^*} \to 2^{\Sigma'^*} : A \mapsto P(A) := \{P(s) | s \in A\}$. Recall that during automaton projection $\kappa : \phi(\Sigma) \to \phi(\Sigma')$ with $\Sigma' \subseteq \Sigma$, a new state z may be created. Thus, given two automata $G_1 \in \phi(\Sigma_1)$, $G_2 \in \phi(\Sigma_2)$ with $\Sigma_1 \cup \Sigma_2 = \Sigma$, we usually do not have $\kappa(G_1 \times G_2) \equiv \kappa_1(G_1) \times \kappa_2(G_2)$, where $\kappa_1 : \phi(\Sigma_1) \to \phi(\Sigma_1 \cap \Sigma')$ and $\kappa_2 : \phi(\Sigma_2) \to \phi(\Sigma_2 \cap \Sigma')$ are two automaton abstractions. But we do have the following.

$$\begin{tikzcd}
\theta(\Sigma) \arrow[r, "\kappa"] \arrow[d, "B/N"'] & \theta(\Sigma') \arrow[d, "B/N"] \\
2^{\Sigma^*} \arrow[r, "P"] & 2^{\Sigma'^*}
\end{tikzcd}$$

Fig. 3. The Commutative Diagram for Proposition 2

Theorem 1: If $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma'$, then we have $\kappa(G_1 \times G_2) \sqsubseteq_h \kappa_1(G_1) \times \kappa_2(G_2)$. If additionally G_i $(i = 1, 2)$ is marking aware with respect to $\Sigma_i \cap \Sigma'$, then $\kappa(G_1 \times G_2) \cong_h \kappa_1(G_1) \times \kappa_2(G_2)$. □

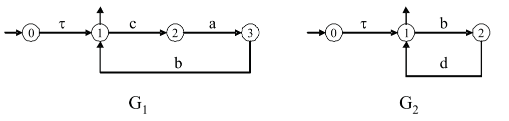

Fig. 4. Example: G_1 and G_2

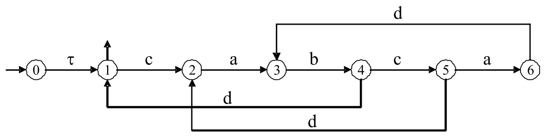

Fig. 5. Example: $G_1 \times G_2$

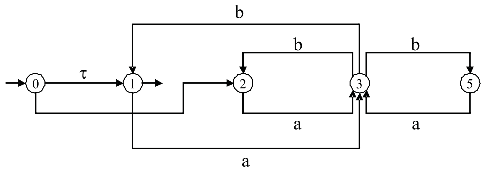

Fig. 6. Example: $\kappa(G_1 \times G_2)$

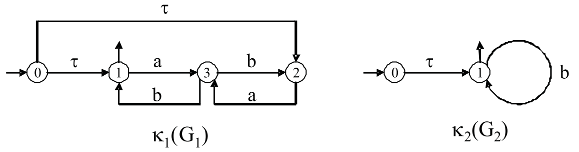

Fig. 7. Example: $\kappa_1(G_1)$ and $\kappa_2(G_2)$

Theorem 1 is about the distribution of automaton abstraction over automaton product. As an illustration we present a simple example. Suppose we have two simple components $G_1 \in \phi(\Sigma_1)$ and $G_2 \in \phi(\Sigma_2)$ depicted in Figure 4, where $\Sigma_1 = \{\tau, a, b, c\}$ and $\Sigma_2 = \{\tau, b, d\}$. The automaton product $G_1 \times G_2$ is depicted in Figure 5. Suppose $\Sigma' = \{\tau, a, b\}$. Then the result of automaton abstraction $\kappa(G_1 \times G_2)$ is depicted in Figure 6, where $\kappa : \phi(\Sigma) \to \phi(\Sigma')$ and the state numbers correspond to those in Figure 5. On the other hand we can compute $\kappa_1(G_1)$ and $\kappa_2(G_2)$ which are depicted in Figure 7. We can see that $\kappa_1(G_1) \times \kappa_2(G_2) \equiv \kappa_1(G_1)$. Since $\Sigma_1 \cap \Sigma_2 = \{\tau, b\} \subseteq \Sigma'$ and G_1, G_2 are marking aware with respect to relevant alphabets, by Theorem 1, we should have $\kappa(G_1 \times G_2) \cong_h \kappa_1(G_1) \times \kappa_2(G_2)$. If we look at Figure 6 carefully, we can see that states 2 and 5 are bisimilar to each other. If we merge them together, we have $\kappa(G_1 \times G_2)/\sim_R \equiv (\kappa_1(G_1) \times \kappa_2(G_2))/\sim_R$. Thus, indeed $\kappa(G_1 \times G_2) \cong_h \kappa_1(G_1) \times \kappa_2(G_2)$, as we expect.

Suppose we want to construct a nonblocking supervisor $S \in \phi(\Sigma')$ with $\Sigma' \subseteq \Sigma$. Let $P : \Sigma^* \to \Sigma'^*$ be the natural projection and $\kappa : \phi(\Sigma) \to \phi(\Sigma')$ the abstraction. Then

$$B(G \times S) = \varnothing \iff P(B(G \times S)) = \varnothing \qquad (1)$$

By Prop. 2 we get

$$P(B(G \times S)) = \varnothing \iff B(\kappa(G \times S)) = \varnothing \qquad (2)$$

Because $S \in \phi(\Sigma')$, we get $\kappa(S) = S$. By Theorem 1,

$$\kappa(G \times S) \sqsubseteq_h \kappa(G) \times \kappa(S) = \kappa(G) \times S$$

Thus,

$$B(\kappa(G) \times S) = \varnothing \Rightarrow B(\kappa(G \times S)) = \varnothing \qquad (3)$$

From (1)-(3) we get that $G \times S$ is nonblocking (i.e. $B(G \times S) = \varnothing$) if $\kappa(G) \times S$ is nonblocking, i.e.

$$B(\kappa(G) \times S) = \varnothing \Rightarrow B(G \times S) = \varnothing \qquad (4)$$

Since S is marking aware w.r.t. Σ', if G is also marking aware w.r.t. Σ', then by Theorem 1, Equation (3) becomes

$$B(\kappa(G) \times S) = \varnothing \iff B(\kappa(G \times S)) = \varnothing$$

and $G \times S$ is nonblocking if and only if $\kappa(G) \times S$ is nonblocking. In any case we can construct a supervisor S based on an abstraction $\kappa(G)$ of G. Of course, it is the user's choice to decide how to pick Σ'. As for how to synthesize S, it will be addressed shortly. Here we face an immediate difficulty. If G is very large, e.g. $G = G_1 \times \cdots \times G_n$ for some very large number $n \in \mathbb{N}$, how to compute $\kappa(G)$? To overcome this difficulty, we propose the following algorithm.

Suppose we have a collection of alphabets $\{\Sigma_i | i \in I\}$, where I is a finite index set. Suppose $I = \{1, 2, \cdots, n\}$ for some $n \in \mathbb{N}$. For any $J \subseteq I$, let $\Sigma_J := \cup_{j\in J}\Sigma_j$. Let $G_i \in \phi(\Sigma_i)$ for each $i \in I$, and $\Sigma' \subseteq \cup_{i\in I}\Sigma_i$. We want to compute $\kappa(\times_{i\in I}G_i)$ efficiently, where $\kappa : \phi(\cup_{i\in I}\Sigma_i) \to \phi(\Sigma')$ is the automaton abstraction.

Sequential Abstraction over Product: (SAP)
For $k = 1, 2, \cdots, n$,

- Set $J_k := \{1, 2, \cdots, k\}$, $T_k := \Sigma_{J_k} \cap \Sigma_{I-J_k} \cup \Sigma'$.
- If $k = 1$ then $W_1 := \kappa_{\Sigma_1,T_1}(G_1)$
- If $k > 1$ then $W_k := \kappa_{T_{k-1}\cup\Sigma_k,T_k}(W_{k-1} \times G_k)$

where $\kappa_{A,B} : \phi(A) \to \phi(A \cap B)$ is the abstraction. □

Proposition 3: Suppose W_n is computed by SAP. Then $\kappa(\times_{i\in I}G_i) \sqsubseteq_h W_n$. □

SAP allows us to obtain an abstraction of the entire system $G = \times_{i\in I}G_i$ in a sequential way. Thus, we can avoid computing G explicitly, which may be prohibitively large for systems of industrial size. By Theorem 1 we have

$$\kappa(\times_{i\in I}G_i \times S) \sqsubseteq_h \kappa(\times_{i\in I}G_i) \times S$$

From Prop. 3, $\kappa(\times_{i\in I}G_i) \sqsubseteq_h W_n$. Then by Prop. 1 we get

$$\kappa(\times_{i\in I}G_i \times S) \sqsubseteq_h W_n \times S$$

Thus, equation (4) can be replaced as

$$B(W_n \times S) = \varnothing \Rightarrow B(\times_{i\in I}G_i \times S) = \varnothing \qquad (5)$$

which means we can synthesize a supervisor $S \in \phi(\Sigma')$ for a large system $G = \times_{i\in I}G_i$ by simply looking at the sequentially attainable abstraction $\kappa(\times_{i\in I}G_i) \sqsubseteq_h W_n$. Next, we discuss how to synthesize this S such that $B(G\times S) = \varnothing$.

III. Supervisor Synthesis over Nondeterministic Finite-State Automata

In the RW paradigm, only deterministic automata are encountered. Thus, given a plant model $G \in \phi(\Sigma)$ and a specification $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$, we can always synthesize a supervisor $S \in \phi(\Sigma)$, which may be empty in some cases, such that the closed loop behavior $G \times S$ is controllable, observable (normal) and nonblocking. 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 6: 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. 6 is the state interpretation of the concept of controllability in the RW paradigm, saying that at any state w in A 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 7: 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. 7 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.

Definition 8: 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 $\Delta \subseteq \Sigma$ and 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. 8 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 4: 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 $A \rightsquigarrow_f G \times H$ for some f-homomorphism, A is controllable with respect to G and Σ_{uc}, A is observable with respect to G and the natural projection $P : \Sigma^* \rightarrow \Sigma_o^*$, and $B(A) = \varnothing$. □

The proof of Prop. 4, 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 $B(A) = \varnothing$. It is possible that, given $G \in \phi(\Sigma)$ and $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma$ there exists more than one nonblocking supervisor. If $S_1 \in \phi(\Sigma)$ and $S_2 \in \phi(\Sigma)$ are two deterministic automata and controllable with respect to G and $\Sigma_{uc} \subseteq \Sigma$, then we can easily show that the union of S_1 and S_2, defined as the canonical recognizer of $L_m(S_1) \cup L_m(S_2)$ with the closed behavior $L(S_1) \cup L(S_2)$, is deterministic and controllable with respect to G and Σ_{uc}. Let

$$\mathcal{C}(\Sigma) := \{S \in \phi(\Sigma) | S \text{ is controllable w.r.t. } G \text{ and } \Sigma_{uc}\}$$

We say $S_1 \in \mathcal{C}$ is *more permissive than* $S_2 \in \mathcal{C}(\Sigma)$, denoted as $S_2 \leq S_1$, if $L(S_2) \subseteq L(S_1)$. From the above discussion we can show that the greatest element of $\mathcal{C}(\Sigma)$ exists, which is called the *supremal controllable* element. Nevertheless, the observability is not closed under union of automata. This situation is similar to the one described in [2].

Proposition 5: Given $G \in \phi(\Sigma)$ and a deterministic automaton $H \in \phi(\Delta)$ with $\Delta \subseteq \Sigma' \subseteq \Sigma$, $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ and $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, if there exists a nonblocking supervisor $S \in \phi(\Sigma')$ of $\kappa(G)$ with respect to H, where $\kappa : \phi(\Sigma) \rightarrow \phi(\Sigma')$, then S is also a nonblocking supervisor of G with respect to H. □

Prop. 5 confirms that we can synthesize a supervisor S based on an abstraction $\kappa(G)$ of G. Of course, κ has to be chosen carefully so that $\kappa(G)$ can capture the specification H in the sense that $\Delta \subseteq \Sigma'$.

Supervisory control of nondeterministic finite-state automata has been explored recently in the literature, e.g. [18] [19]. A result similar to Prop. 4 is provided in [19]. In this paper we focus on the supervisor synthesis based on abstraction. Thus, Prop. 5 is our main result.

IV. Example

As an illustration we present the following example. Suppose we have two machines, which are functionally identical, except for individual event labels. The system is depicted in Figure 8. Each machine G_i ($i = 1, 2$) has the following standard operations: (1) fetching a work piece (a_i); (2) preprocessing (b_i); (3) postprocessing (c_i); (4) polishing (e_i); (5) packaging (d_i). To produce one piece of product, three work pieces are needed, one for each machine to go through those standard steps (1)-(4). At Step (5) all three work pieces are packed together to form a final product. After preprocessing b_i, there are two choices: to be postprocessed directly (c_i) or to be polished first (e_i) before postprocessing. The latter gives a product with better quality. The negative aspect is that polishing may cause the machine G_1 to fail (f_1). If failure does happen, G_1 will stop automatically and wait for repair. Among each alphabet Σ_i, the controllable alphabet is $\Sigma_{i,c} = \{a_i, e_i\}$, and the observable alphabet $\Sigma_{i,o}$ is $\Sigma_i - \{f_1\}$, namely every event is observable (for the purpose of simplicity), except for the failure event f_1. There is one specification $H \in \phi(\Delta)$ with $\Delta = \{e_1, e_2\}$, depicted in Figure 9, saying that if G_1 polishes a work piece (e_1), then G_2 must polish a work piece afterwards (e_2). We now start to synthesize a nonblocking supervisor that enforces the specification H.

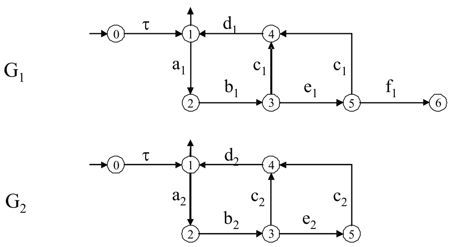

Fig. 8. A Simple Processing Unit

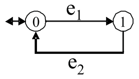

Fig. 9. The Specification $H \in \phi(\Delta)$

First, we create an appropriate abstraction of $G_1 \times G_2$. We pick $\Sigma' = \{\tau, a_1, a_2, e_1, e_2\}$. The motivation is that, since $\Delta \subseteq \Sigma'$, the abstraction $\kappa(G_1 \times G_2)$, where $\kappa : \phi(\Sigma_1 \cup \Sigma_2) \rightarrow \phi(\Sigma')$, can capture the specification H; and since all controllable events are in Σ', the abstraction $\kappa(G_1 \times G_2)$ also contains all means of control available to $G_1 \times G_2$ itself. In reality, we may also want to include all observable events in Σ' so that the abstraction can capture all possible observations as well. In this example, we do not consider observations because every event is observable (except for f_1). Since $\Sigma_1 \cap \Sigma_2 = \{\tau\} \subseteq \Sigma'$, by Theorem 1,

$$\kappa(G_1 \times G_2) \sqsubseteq_h \kappa_1(G_1) \times \kappa_2(G_2)$$

where $\kappa_i : \phi(\Sigma_i) \rightarrow \phi(\Sigma_i \cap \Sigma')$ with $i = 1, 2$ is the abstraction. The results of $\kappa_1(G_1)$ and $\kappa_2(G_2)$ are depicted in Figure 10. The abstraction $G' := \kappa_1(G_1) \times \kappa_2(G_2)$ is depicted in Figure 11, We now use G' and H to synthesize a supervisor. Clearly, e_1 must be disabled at states 2 and 5 in Figure 11. Otherwise $G_1 \times G_2$ will encounter blocking states 3 and 4. Once e_1 is disabled, e_2 must be disabled as well. Otherwise H will not hold. After removing all transitions of e_1 and e_2 from G' in Figure 11, the remaining reachable sub-automaton in Figure 11 is controllable, observable and

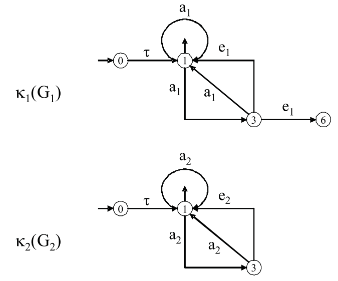

Fig. 10. The Abstractions $\kappa_1(G_1)$ and $\kappa_2(G_2)$

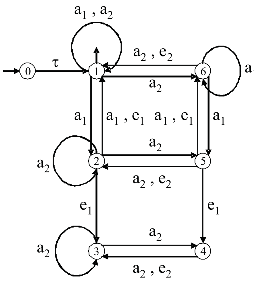

Fig. 11. The Product $G' = \kappa_1(G_1) \times \kappa_2(G_2)$

nonblocking, as depicted in Figure 12. By Prop. 4 we get

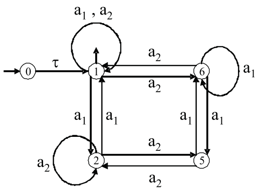

Fig. 12. Controllable, Observable, Nonblocking Sub-automaton A of G'

that, the canonical recognizer of the marked behavior $N(A)$ is the supervisor S, which is depicted in Figure 13. Clearly,

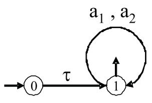

Fig. 13. The Supervisor $S \in \phi(\Sigma')$

S disables events e_1 and e_2 but allows all other events to happen. We can check that S is a nonblocking supervisor of $G_1 \times G_2$ with respect to the specification H, as predicted by Prop. 5. We can verify that the maximum number of states of any intermediate computational result in terms of automata is 7 states, which occurs when we compute $\kappa_1(G_1) \times \kappa_2(G_2)$. Clearly, abstractions help to reduce the computational complexity in this example because otherwise we will have to face the product $G_1 \times G_2$ directly, which has 31 states.

V. Conclusions

In this paper we introduce a new technique for automaton abstraction and supervisor synthesis based on nondeterministic finite-state automata. Unlike natural projections in the RW paradigm, the proposed automaton abstraction preserves the nonblocking property in the sense that the product of a target automaton G with another automaton S is nonblocking if and only if the product of the abstraction $\kappa(G)$ of G and S is nonblocking. Thus, by synthesizing a local supervisor S to achieve nonblocking supervisory control on $\kappa(G)$, which is usually smaller than the original plant G in terms of the number of states, we can guarantee that the supervisory control on the original plant G is also nonblocking. The abstraction technique has potential applications in modular and distributed supervisor synthesis, which will be addressed in our future papers. The complexity issue will also be addressed by then.

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] J. Komenda and J.H. van Schuppen. Modular control of discrete-event systems with coalgebra. *IEEE Trans. Automatic Control*, 53():447–460, 2008.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] R.J. Leduc and P. Dai. Synthesis method for hierarchical interface-based supervisory control. In *Proc. 26th American Control Conference*, pages 4260–4267, New York City, USA, July, 2007

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

[12] B. Brandin and W.M. Wonham. Supervisory control of timed discrete event systems. *IEEE Trans. Automatic Control*, 39(2):329–351, 1994.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] C. Zhou and R. Kumar. A small model theorem for bisimilarity control under partial observation. *IEEE Trans. Automation Science and Engineering*, 4(1):93-97, 2007.