2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Supervisory Control of Concurrent Discrete-Event Systems

Rong Su

Abstract— **Concurrency is a common feature in most industrial systems, where several components can execute different actions simultaneously. In this paper we first bring in a new supervisory control map for a concurrent system, and subsequently a new concept of concurrent controllability, then we show that the supremal concurrently controllable and normal sublanguages exist, which can be computed by an algorithm proposed in the paper.**

Index Terms— **discrete-event systems, concurrency, controllability, observability, normality, supervisory control**

I. Introduction

Concurrency is a property of systems in which several components may execute different actions simultaneously, and potentially interacting with each other. The standard language-based Ramadge-Wonham (RW) supervisory control paradigm [4] [9] has been shown effective to derive proper supervisors to enforce liveness and safety related logic constraints. Nevertheless, it assumes that a target system is asynchronous with respect to different events. This assumption has limited its applications to complex industrial systems, where concurrency is commonly seen either due to the nature of the system or due to some intentional setup to improve the system's performance. Facing this theoretical shortcoming, in the past several researchers proposed different extensions to the RW paradigm in order to address concurrency properly. In [2] the authors propose a new concurrency product that takes potential simultaneous executions of different events into consideration, and a sufficient and necessary condition is presented, which ensures that a well posed supervisor synthesized based on an asynchronous model can deal with possible concurrent behaviors. In some sense this work does not intend to solve concurrent synthesis as a separate paradigm but rather to restrict a system's asynchronous behaviors in such a way that the presence of concurrency will not affect the supervisor's asynchronous control capability. The authors in [7] extend the idea of concurrency product from [2] and the language-based concurrent supervisory control paradigm in [6]. By adopting the same concepts of controllability and observability as those in the RW asynchronous case, and introducing a new concept called *concurrent well-posedness* (CWP), the authors show that the supremal sublanguages usually do not exist. Thus, we need to pursue either the infimal or the maximal prefix closed sublanguages. The work in [7] suggests that such a negative result about supremal sublanguages is due to the usage of CWP and observability. Thus, even when we replace observability with normality, the negative result still holds because of CWP, as a contrast to the existence of supremal controllable and normal sublanguages in the RW asynchronous case. Although their work is valid within their formalism, we will indirectly show in this paper that the CWP is not the fundamental reason for the non-existence of supremal sublanguages when normality is used. In addition, their concurrent synchronous product is defined over asynchronous components, thus, not suitable for a system consisting of concurrent subsystems. In [5] the authors propose a new paradigm called *multiagent product systems*. In this paradigm all components must fire (possibly empty) events at the same time. Besides the different representing formats for compound events, where in [5] event vectors are used in contrast with event subsets used in [2] [7], the concurrent product in [5] is different from those in [2] and [7] in terms of how to deal with shared events. More explicitly, in [2] and [7] an event must be fired simultaneously in all components containing this event, which is not necessarily true in [5]. The work in [5] also deals with only prefix closed languages as that in [7], and the supremal (MA) controllable sublanguages do not exist in general. Partial observation is not considered in [5].

In this paper we adopt the same way of defining a compound event as a set of atomic events as used in [2] and [7]. But unlike [2] and [7], all subsequent developments in this paper such as controllability, observability, normality and supervisory control maps are all defined over the compound event alphabet. In particular, a concurrent composition operation is defined, which is conceptually similar to the synchronous product defined in [1], and a novel concept called *concurrent controllability* is introduced, which describes not only the unstoppable natural of uncontrollable events (as manifested in the standard controllability concept in the RW paradigm) but also the collateral event enabling and disabling phenomena due to the usage of compound events. It turns out that the concept of CWP in [7] is related to the phenomenon of collateral event enabling under the full concurrency circumstance, but it is not necessarily needed for cases where partial concurrency presents. We show that supremal concurrently controllable normal sublanguages exist and a concrete algorithm is provided. This result makes our paper different from previous papers, and indicates that the concurrent case can be treated similar to the RW asynchronous case in the centralized setup. In this sense we believe our work shed some light on the fundamentals of supervisory control for concurrent systems.

This paper is organized as follows. In Section II we introduce a formal setup of a concurrent system, then show in Section III the supervisor existence and supremal synthesis

Rong Su is affiliated with Division of Control and Instrumentation, School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798. Email: rsu@ntu.edu.sg

results. Conclusions are drawn in Section IV.

II. A CONCURRENT SYSTEM

Let Σ be a finite alphabet and Σ^* be the free monoid on Σ, where the unit element is the empty string ϵ and the monoid operation is the concatenation. 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'. For all string $s \in \Sigma^*$, $\epsilon s = s\epsilon = s$. A subset $L \subseteq \Sigma^*$ is called a *language*. We call $\overline{L} = \{s \in \Sigma^* | (\exists t \in L)\, s \leq t\} \subseteq \Sigma^*$ the *prefix closure* of L. Given two languages $L, L' \subseteq \Sigma^*$, let $LL' := \{ss' \in \Sigma^* | s \in L \wedge s' \in L'\}$ be the concatenation of L and L'. Given a set S, we use $|S|$ to denote its size and 2^S for its power set.

Given an alphabet Σ, each element $\sigma \in \Sigma$ is called an *atomic event* denoting a physical action, e.g., "turn on a valve", "pick up a workpiece" or "start a job" etc. We call σ *controllable* if the corresponding physical action can be disabled whenever needed; otherwise, it is *uncontrollable*. We call σ *observable* if the occurrence of the corresponding physical action can be detected by a sensor; otherwise, it is *unobservable*. Let the disjoint sets Σ_c and Σ_{uc} be the collections of controllable and uncontrollable events respectively, and the disjoint sets Σ_o and Σ_{uo} the collections of observable and unobservable events respectively. We have $\Sigma = \Sigma_c \cup \Sigma_{uc} = \Sigma_o \cup \Sigma_{uo}$.

A *compound event* with respect to Σ is a nonempty subset $\varsigma \subseteq \Sigma$, whose firing denotes the simultaneous occurrence of atomic events contained in ς. We call ς *controllable* if $\varsigma \cap \Sigma_c \neq \varnothing$, and *observable* if $\varsigma \subseteq \Sigma_o$. Let $f(\Sigma) \subseteq 2^\Sigma$ be a set of compound events. A typical compound event set can be defined as $f(\Sigma) := 2^\Sigma$, which allows full concurrency among all events, or $f(\Sigma) := 2^{\Sigma_{uc}} \cup \bigcup_{\sigma \in \Sigma_c} 2^{\Sigma_{uc} \cup \{\sigma\}}$, which allows full concurrency among only uncontrollable events. If we set $f(\Sigma) := \{\{\sigma\} | \sigma \in \Sigma\}$, then no concurrency among any atomic events is allowed, which is equivalent to the RW asynchronous case. The construction of $f(\Sigma)$ is done by a user, which must reflect the true concurrent nature of atomic events. We use $f_c(\Sigma)$ and $f_{uc}(\Sigma)$ to denote the collections of controllable and uncontrollable compound events, and $f_o(\Sigma)$ and $f_{uo}(\Sigma)$ for the collections of observable and unobservable compound events. A subset $L \subseteq f(\Sigma)^*$ is called a *concurrent language* with respect to $f(\Sigma)$.

Let $\Sigma' \subseteq \Sigma$. A map $\psi : f(\Sigma)^* \to f(\Sigma')^*$ is called the *concurrent projection* with respect to $(f(\Sigma), f(\Sigma'))$, if

1) $\psi(\epsilon) = \epsilon$
2) $(\forall \varsigma \in f(\Sigma))\, \psi(\varsigma) := \begin{cases} \{\sigma \in \Sigma'\} & \text{if } \varsigma \cap \Sigma' \neq \varnothing \\ \epsilon & \text{otherwise} \end{cases}$
3) $(\forall s\varsigma \in f(\Sigma)^*)\, \psi(s\varsigma) = \psi(s)\psi(\varsigma)$

Given a concurrent language $L \subseteq f(\Sigma)^*$, $\psi(L) := \{\psi(s) \in f(\Sigma')^* | s \in L\}$. The inverse image map of ψ is

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

Given $L_1 \subseteq f(\Sigma_1)^*$ and $L_2 \subseteq f(\Sigma_2)^*$, the *concurrent synchronous product* of L_1 and L_2 is defined as $L_1 || _C L_2 := \psi_1^{-1}(L_1) \cap \psi_2^{-1}(L_2)$, where $\psi_1 : f(\Sigma_1 \cup \Sigma_2)^* \to f(\Sigma_1)^*$ and $\psi_2 : f(\Sigma_1 \cup \Sigma_2)^* \to f(\Sigma_2)^*$ are natural projections. Synchronous product is commutative and associative.

A *finite-state concurrent automaton* (FSCA) with respect to Σ is $G = (X, f(\Sigma), \xi, x_0, X_m)$, where X is the state set, $f(\Sigma)$ the alphabet, x_0 the initial state, $X_m \subseteq X$ the marker state set, and $\xi : X \times f(\Sigma) \to X$ the (partial) transition function, which is extended to $X \times f(\Sigma)^*$. For all $x \in X$ and $\varsigma \in f(\Sigma)$, we use $\xi(x, \varsigma)!$ to denote that the transition $\xi(x, \varsigma)$ is defined. Let $L(G) := \{s \in f(\Sigma)^* | \xi(x_0, s)!\}$ be the *closed* behavior of G and $L_m(G) := \{s \in L(G) | \xi(x_0, s) \in X_m\}$ be the *marked* behavior of G. We say G is *nonblocking* if $L(G) = \overline{L_m(G)}$. Let $\phi(\Sigma)$ denote the set of all well posed FSCA, whose alphabets are $f(\Sigma)$.

The composition of two FSCA is specified by the following product operation. Given two FSCA $G_i = (X_i, f(\Sigma_i), \xi_i, x_{i,0}, X_{i,m})$ $(i = 1, 2)$, the *concurrent product* of G_1 and G_2, written as $G_1 \times G_2$, is a FSCA $G = (X, f(\Sigma), \xi, x_0, X_m)$, where $X := X_1 \times X_2$, $\Sigma := \Sigma_1 \cup \Sigma_2$, $x_0 := (x_{1,0}, x_{2,0})$, $X_m := X_{1,m} \times X_{2,m}$ and $\xi : X_1 \times X_2 \times f(\Sigma) \to X_1 \times X_2$ is defined as follows,

$$\xi(x_1, x_2, \varsigma) := \begin{cases} (\xi_1(x_1, \varsigma), x_2) & \text{if } \varsigma \subseteq \Sigma_1 - \Sigma_2 \wedge \xi_1(x_1, \varsigma)! \\ (x_1, \xi_2(x_2, \varsigma)) & \text{if } \varsigma \subseteq \Sigma_2 - \Sigma_1 \wedge \xi_2(x_2, \varsigma)! \\ (\xi_1(x_1, \varsigma_1), \xi_2(x_2, \varsigma_2)) & \text{if } \varsigma = \varsigma_1 \cup \varsigma_2 \wedge \varsigma_1 \cap \Sigma_2 \subseteq \varsigma_2 \\ & \quad \wedge \varsigma_2 \cap \Sigma_1 \subseteq \varsigma_1 \wedge \xi_1(x_1, \varsigma_1)! \\ & \quad \wedge \xi_2(x_2, \varsigma_2)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

The reason why we require $\varsigma_1 \cap \Sigma_2 \subseteq \varsigma_2$ and $\varsigma_2 \cap \Sigma_1 \subseteq \varsigma_1$ is that shared atomic events, i.e., events in $\Sigma_1 \cap \Sigma_2$, must be fired simultaneously in both G_1 and G_2. If $\Sigma_1 = \Sigma_2$ or $f(\Sigma)$ is restricted to singleton sets, i.e., $f(\Sigma) := \{\{\sigma\} | \sigma \in \Sigma\}$ then the concurrent product becomes the synchronous product in the RW asynchronous case. This matches our expectation that asynchrony is a special case of concurrency. It is worth to point it out the concept of compound events bears some similarity to the concept of synchronization vectors [1], except that we assign the attributes of controllability and observability to each compound event. The corresponding concurrent product is closely related to the synchronous product defined in [1], where each synchronization constraint is specified as a vector in which an event shared by different components must be fired in them simultaneously.

Proposition 1: The concurrent product is well defined. □

Proposition 2: Let $G_1 \in \phi(\Sigma_1)$, $G_2 \in \phi(\Sigma_2)$. Then $L_m(G_1) || _C L_m(G_2) = L_m(G_1 \times G_2)$. □

In the RW asynchronous case language synchronous product is computable by automaton synchronous product. Prop. 2 shows that a similar result holds for the concurrent case, where language concurrent synchronous product is computable by automaton concurrent product.

Definition 1: A *concurrent system* is a finite collection of FSCA $\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I\}$, whose compositional behaviors are captured by their concurrent product. □

To illustrate the aforementioned concepts and operations, suppose $\mathcal{G} = \{G_1 \in \phi(\Sigma), G_2 \in \phi(\Sigma)\}$, which is depicted in Figure 1, where $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 =$

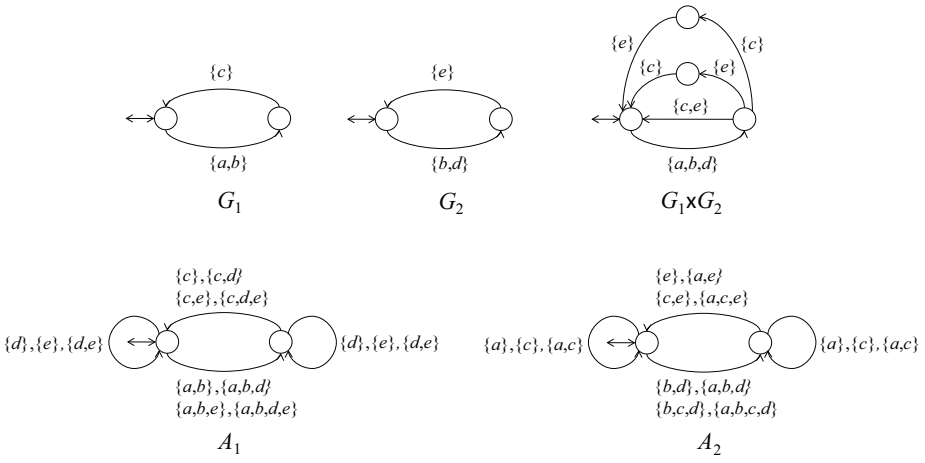

Fig. 1. Example 1

$\{b, d, e\}$. Assume that full concurrency among all atomic events are considered. Thus, $f(\Sigma_1) = 2^{\Sigma_1}$, $f(\Sigma_2) = 2^{\Sigma_2}$ and $f(\Sigma_1 \cup \Sigma_2) = 2^{\Sigma_1 \cup \Sigma_2}$. The concurrent product $G_1 \times G_2$ is depicted in Figure 1. By Prop. 2 we know that $L_m(G_1)||_C L_m(G_2) = L_m(G_1 \times G_2)$. On the other hand, to compute $L_m(G_1)||_C L_m(G_2)$ we can first compute $\psi_i^{-1}(L_m(G_i))$ $(i = 1, 2)$, where $\psi_i : f(\Sigma_1 \cup \Sigma_2)^* \to f(\Sigma_i)^*$ is the concurrent projection. The corresponding recognizer A_i, i.e., $L_m(A_i) = \psi_i^{-1}(L_m(G_i))$, is depicted in Figure 1. it is not difficult to show that $L_m(A_1) \cap L_m(A_2) = L_m(A_1 \times A_2) = L_m(G_1 \times G_2)$. Since the alphabet for A_1 is equal to the alphabet of A_2, we can check that the concurrent product $A_1 \times A_2$ is equal to the synchronous product in the RW asynchronous case.

III. Supervisory control of a concurrent system

Let $\Gamma := \{\gamma \subseteq 2^{f(\Sigma)} | (\forall \varsigma \in \gamma)\, \Sigma_{uc} \subseteq \varsigma\}$. Each $\gamma \in \Gamma$ is called a *control pattern*, which consists of several compound events all containing the uncontrollable alphabet Σ_{uc}. Given a FSCA $G \in \phi(\Sigma)$, we define a supervisory control map $V : \psi_o(L(G)) \to \Gamma$, where $\psi_o : f(\Sigma)^* \to f(\Sigma_o)^*$ is the concurrent projection. Compared with a supervisory control map for an asynchronous case in the RW paradigm, we simply replace each atomic event with a compound event. The control mechanism works in a similar way: based on the current observation $s \in \psi_o(L(G))$, the control map $V(s)$ describes all possible compound events that can be fired at the current state. An internal mechanism will (nondeterministically) choose which compound event to be fired. The *closed behavior* of G under V is defined as follows:

1) $\epsilon \in L(V/G)$,
2) $(\forall s \in L(V/G))(\forall \varsigma \in f(\Sigma))(\exists \varsigma' \in V(P_o(s)))\, s\varsigma \in L(G) \wedge \varsigma \subseteq \varsigma' \Rightarrow s\varsigma \in L(V/G)$,
3) all strings in $L(V/G)$ are obtained in Steps (1),(2).

The *marked behavior* of G under V is defined as $L_m(V/G) := L(V/G) \cap L_m(G)$. We say V is *nonblocking* if $\overline{L_m(V/G)} = L(V/G)$. The definition of the closed and marked behaviors of supervised system V/G indicates that the supervisor can disable and enable any controllable atomic events, but cannot force any enabled atomic events to fire simultaneously. The latter is the main reason for including the condition (2) in the definition, which says that any subset of enabled atomic events may occur during an actual execution. In some sense, the definition of the supervisory control map V describes the observation and control mechanisms in a discrete-event system, which determines the corresponding control theory. With such an interpretation of supervisory control in a concurrent setup we will show below that (almost) every important result for the RW asynchronous case has a counterpart for the concurrent case. The first result is about the existence of a supervisory control map for a given concurrent sublanguage.

Problem 1: Given a plant $G \in \phi(\Sigma)$, let $K \subseteq L_m(G)$. Under what conditions does there exist a control map V such that $L_m(V/G) = K$ and $L(V/G) = \overline{L_m(V/G)}$? □

Recall that to solve a similar problem in the RW asynchronous case we need two important concepts: *controllability* and *observability*. In the following we will show that we also need similar concepts to solve the concurrent version.

Definition 2: $K \subseteq L_m(G)$ is *concurrently controllable* with respect to G if

1) $\overline{K} f_{uc}(\Sigma) \cap L(G) \subseteq \overline{K}$
2) $(\forall s \in \overline{K})(\forall \varsigma \in f(\Sigma))\, s\varsigma \in \overline{K} \Rightarrow (\forall \varsigma' \in f(\Sigma) \cap 2^{\varsigma})\, [s\varsigma' \in L(G) \Rightarrow s\varsigma' \in \overline{K}]$
3) $(\forall s \in \overline{K})(\forall \varsigma \in f(\Sigma))\, s\varsigma \notin \overline{K} \Rightarrow (\forall \varsigma' \in f(\Sigma))[\varsigma \cap \Sigma_c \subseteq \varsigma' \cap \Sigma_c \Rightarrow s\varsigma' \notin \overline{K}]$ □

The concept of concurrent controllability is used to describe the existence of an event disabling mechanism that achieves K. So the fist condition is quite natural - no uncontrollable compound event can be disabled because all atomic events in it are uncontrollable. The second condition is used to capture the phenomenon of collateral even enabling, namely to enable a compound event ς all atomic events in ς are allowed, which means any compound event consisting of some atomic events of ς must be allowed as well. This condition matches the supervisory control map V proposed above. The third condition is used to capture the phenomenon of collateral event disabling, namely to disable

a compound event ς by disabling the controllable atomic event in ς we must disable every compound event, which contains all controllable atomic events in ς.

Proposition 3: Let $K_1, K_2 \subseteq L_m(G)$ be concurrently controllable with respect to G. Then $K_1 \cup K_2$ is concurrently controllable with respect to G. □

Prop. 3 indicates that concurrent controllability is closed under set union, just like controllability is closed under set union in the RW asynchronous case.

Definition 3: Let $\psi_o : f(\Sigma)^* \to f(\Sigma_o)^*$ be the concurrent projection. Then $K \subseteq L_m(G)$ is *observable* with respect to (G, ψ_o) if for all $s, s' \in \overline{K}$ and $\varsigma, \varsigma' \in f(\Sigma)$ with $\varsigma' \subseteq \varsigma$, $s\varsigma \in \overline{K} \wedge s'\varsigma' \in L(G) \wedge \psi_o(s) = \psi_o(s') \Rightarrow s'\varsigma' \in \overline{K}$ □

The concept of observability is extended from its counterpart in the RW asynchronous case [3] to cope with compound events instead of atomic events.

Theorem 1: Given $G \in \phi(\Sigma)$, let $K \subseteq L_m(G)$. Then a nonblocking supervisory control map V exists such that $L_m(V/G) = K$ iff K is concurrently controllable with respect to G, observable with respect to (G, ψ_o), and $\overline{K} \cap L_m(G) = K$. □

Theorem 1 allows us to solve the control problem, i.e., to find an appropriate supervisory control map V to realize K, by checking whether K satisfies those aforementioned conditions. In many practical applications such a concurrent sublanguage is not explicitly given. Instead, a requirement $E \subseteq f(\Sigma')^*$ with $\Sigma' \subseteq \Sigma$ is given. We need to solve the following synthesis problem.

Problem 2: Given a plant $G \in \phi(\Sigma)$ and a requirement $E \subseteq f(\Sigma')^*$ with $\Sigma' \subseteq \Sigma$, how to find a concurrent sublanguage $K \subseteq L_m(G)||_C E$ satisfying the first two conditions in Theorem 1? □

The last condition of Theorem 1 is not considered here because it is not directly related to the nonblocking control. To solve Problem 2 we need to construct a formal synthesis procedure. To this end we adopt the same strategy as used in the RW asynchronous case - that is to come up with a concept of supremal sublanguage with respect to controllability and observability. As we know, observability is not closed under set union. Therefore, we adopt the concept of *normality*.

Definition 4: $K \subseteq L_m(G)$ is *normal* with respect to (G, ψ_o) if $K = L(G) \cap \psi_o^{-1}(\psi_o(K))$. □

By using a similar argument as in the RW asynchronous case we can show that $\overline{K}$ is normal with respect to (G, ψ_o) implies that K is observable. Let

$$\mathcal{C}(G, E) :=$$
$$\{K \subseteq L_m(G)||_C E | K \text{ is concurrently controllable w.r.t. } G\}$$

and

$$\overline{\mathcal{N}}(G, E) := \{K \subseteq L_m(G)||_C E | \overline{K} \text{ is normal w.r.t. } (G, \psi_o)\},$$

we define $\mathcal{CN}(G, E) := \mathcal{C}(G, E) \cap \overline{\mathcal{N}}(G, E)$.

Proposition 4: There exists $K_* \in \mathcal{CN}(G, E)$ such that for all $K \in \mathcal{CN}(G, E)$ we have $K \subseteq K_*$. □

We call K_* the *supremal concurrently controllable and normal sublanguage of G with respect to E*, denoted as $\sup\mathcal{CN}(G, E)$. Because of the similarity between normalities in the concurrent case and in the RW asynchronous case, we can easily show that $\sup\overline{\mathcal{N}}(G, E)$ can be computed by an algorithm shown in [8]. We use the following simple procedure to compute $\sup\mathcal{C}(G, E)$.

Proc. C:

1) Input: $G = (X, f(\Sigma), \xi, x_0, X_m)$ and $A = (Y, f(\Sigma), \eta, y_0, Y_m)$ with $L_m(A) = E$.
2) Let $G^0 := G \times A = (Z^0 := X \times Y, f(\Sigma), \zeta^0 := \xi \times \eta, z_0 := (x_0, y_0), Z_m^0 := X_m \times Y_m)$.
3) Let
$$B(G^0) := \{(x, y) \in Z^0 | [(\exists \varsigma \in f_{uc}(\Sigma))\xi(x, \varsigma)! \wedge \neg\eta(y, \varsigma)!] \vee [(\forall s \in f(\Sigma)^*)\zeta^0(x, y, s) \notin Z_m^0]\}$$
4) Iterate on $k = 1, 2, \cdots$,
4.a) $h(B(G^{k-1})) := \{z \in Z^{k-1} | (\exists s \in f_{uc}(\Sigma)^*)\, \zeta^{k-1}(z, s) \in B(G^{k-1})\}$.
4.b) Let $\hat{G}^k = (\hat{Z}^k := Z^{k-1} - h(B(G^{k-1})), f(\Sigma), \hat{\zeta}^k, z_0, \hat{Z}_m^k := Z_m^{k-1} \cap Z^k)$, where
$$\hat{\zeta}^k(z, \varsigma)! \iff \hat{\zeta}^{k-1}(z, \varsigma)! \wedge (\forall \varsigma' \in f(\Sigma))[\hat{\zeta}^{k-1}(z, \varsigma') \in h(B(G^{k-1})) \Rightarrow \varsigma \cap \Sigma_c \subset \varsigma' \cap \Sigma_c]$$
4.c) $\tilde{G}^k := (\hat{Z}^k, f(\Sigma), \tilde{\zeta}^k, z_0, \hat{Z}_m^k)$, where
$$\tilde{\zeta}^k(z, \varsigma)! \iff (\forall \varsigma' \in f(\Sigma) \cap 2^\varsigma)\, \hat{\zeta}^k(z, \varsigma')!$$
4.d) $G^k := (Z^k \subseteq \hat{Z}^k, f(\Sigma), \zeta^k := \tilde{\zeta}^k|_{Z^k}, z_0, Z_m^k := Z^k \cap \hat{Z}_m^k)$ is the largest trimmed automaton of $\tilde{G}^k$, i.e., ζ^k is the restriction of $\tilde{\zeta}^k$ over Z^k, where
$$(\forall z \in \hat{Z}^k)\, z \in Z^k \iff (\exists s \in f(\Sigma)^*)\, \tilde{\zeta}(z_0, s) = z \wedge (\exists s' \in f(\Sigma)^*)\, \tilde{\zeta}^k(z, s') \in \hat{Z}_m$$
4.e) If for each $z \in Z^k$ there exists $s \in f(\Sigma)^*$ such that $\zeta^k(z, s) \in Z_m^k$, then terminate and output $L_m(G^k)$. □

Proc. C can be explained as follows. In step (2) we first construct the concurrent product G^0 of G and A, where A is a recognizer of E. Then in step (3) we construct the set $B(G^0)$ of all "bad" states, each of which, say state z, either violates the controllability property in the sense that some uncontrollable compound event is defined in G but is not allowed in E, or violates the nonblocking property in the sense that from z no marker state can be reached. In step (4) we remove bad states in an iterative way. We first in step (4.a) compute all states that can reach some bad states in $B(G^{k-1})$ via some uncontrollable paths, thus, those states become bad as well. Then in step (4.b) we separate these bad states from good ones by removing the corresponding transitions, and we also remove all collaterally disabled events (which is aimed to satisfy condition (3) of concurrent controllability). In step (4.c) we want to make sure that condition (2) of concurrent controllability holds. Finally, in step (4.d) we remove those unreachable and uncoreachable states so that the outcome is a trimmed automaton G^k. If G^k contains no bad states (as tested in step (4.e)), then the algorithm terminates and outputs the marked behavior of the last trimmed automaton G^k. Otherwise, the whole process of step (4) is repeated. Notice that in each step we remove some states and/or transitions from the last automaton, it is clear that the algorithm terminates.

Proposition 5: Proc. C terminates to $\sup\mathcal{C}(G, E)$. □

If $f(\Sigma) := \{\{\sigma\} | \sigma \in \Sigma\}$, namely all events are asynchronous, then Procedure C is reduced to an algorithm for the supremal controllable sublanguage of G with respect to E in the RW asynchronous case. We can show that their complexities are also similar. Because we have concrete procedures to compute sup$\mathcal{C}(G, E)$ and sup$\overline{\mathcal{N}}(G, E)$, we are ready to present an algorithm for sup$\mathcal{CN}(G, E)$.

Proc. CN:

1) Input: a plant $G \in \phi(\Sigma)$, a requirement $E^0 \subseteq f(\Sigma)^*$.
2) Iterate on $k = 1, 2, \cdots$,
   a) Compute $E^k := \sup\overline{\mathcal{N}}(G, \sup\mathcal{C}(G, E^{k-1}))$.
   b) If $E^k = E^{k-1}$ then terminate and output E^k. □

Proc. CN is self explained: in each round k we compute the supremal concurrently controllable sublanguage first, then the supremal normal sublanguage. If the outcome is the same as the outcome of the previous round, then a fixed point is reached, i.e., E^k is concurrently controllable and normal. Otherwise, we continue the iteration.

Theorem 2: Proc. CN terminates to sup$\mathcal{CN}(G, E)$. □

The complexity of Proc. CN is exponential and close to the complexity of computing the supremal controllable and normal sublanguage [8]. To illustrate the synthesis procedure we go through a simple example, which consists of two robots, *Robot*1 and *Robot*2, one 3-slot *Buffer* and one *Machine*. Robots can fill the buffer one slot per each action, which are denoted as $1F$ and $2F$ respectively. The machine picks an item and clear a buffer slot (denoted by P), then either outputs a good product (denoted as O) or returns a defective product back to the buffer for rework (denoted as R). The system is depicted in Figure 2. The component

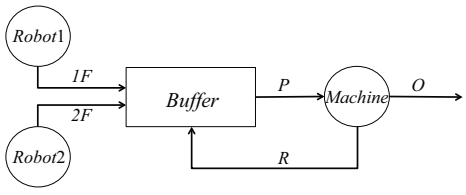

Fig. 2. Example 3: A simple manufacturing system

models are depicted in Figure 3, where $\Sigma_{Robot1} = \{1F\}$, $\Sigma_{Robot2} = \{2F\}$ and $\Sigma_{Machine} = \{P, O, R\}$. The overall alphabet Σ is the union of all component alphabets. We have $\Sigma_o = \Sigma$, namely all events are observable (for the illustration purpose), and $\Sigma_c = \{1F, 2F, P\}$. We consider full concurrency among all atomic events. The plant model G, which is the concurrent product of all component models, is also depicted in Figure 2. The requirement *SPEC* is depicted in Figure 4, which specifies the following goals: (1) the buffer cannot overflow; (2) the buffer cannot underflow; (3) the machine cannot pick when the buffer is empty, even though some robot may place an item simultaneously. The final supremal concurrently controllable and normal sublanguage $\mathcal{CN}(G, SPEC)$ is depicted in Figure 5, which reveals the following control actions:

- When *Buffer* contains one item and *Machine* has picked one item, $\{1F, 2F\}$,$\{O, 1F, 2F\}$,$\{R, 1F, 2F\}$ shall not occur, i.e., $1F$, $2F$ shouldn't be enabled simultaneously.
- When *Buffer* contains two items and *Machine* has picked one item, only $\{O\}$ and $\{R\}$ shall happen, i.e., $1F$ and $2F$ should be disabled simultaneously.
- When *Buffer* contains two items and *Machine* has not picked one item, $\{1F, 2F\}$,$\{P, 1F, 2F\}$ shall not occur, i.e., $1F$ and $2F$ shouldn't be enabled simultaneously.
- When *Buffer* contains three items and *Machine* has not picked one item yet, only $\{P\}$ shall happen, i.e., $1F$ and $2F$ should be disabled simultaneously.

If we use $M(Buffer)$ to denote the number of occupied slots, $M(Machine)$=1 or 0 to denote whether *Machine* holds an item and $T(iF)$=1 or 0 $(i = 1, 2)$ to denote whether atomic event iF is enabled, then the above control actions can be described in the following expressions:

$$M(Buffer) + M(Machine) = 2 \quad \Rightarrow \quad T(1F) + T(2F) \leq 1$$
$$M(Buffer) + M(Machine) = 3 \quad \Rightarrow \quad T(1F) + T(2F) = 0$$

At this point we can see that there is clearly a one-to-one correspondence between concepts for the concurrent case and concepts for the well known RW asynchronous case. The deep impact of replacing atomic events with compound events is reflected in the concept of concurrent controllability, which describes not only the well known unstoppable nature of uncontrollable events but also the collateral event enabling and disabling phenomena that only appear when compound events are used. Bearing such a correspondence in mind we have shown in Theorem 1 and Theorem 2 that concurrency does not affect the centralized supervisor existence and supervisory synthesis results.

IV. Conclusions

In this paper we have introduced the concept of compound events to address the possibly unavoidable concurrent behaviors among atomic events. Then we have shown that relevant concepts such as controllability, observability (normality) and the supervisory control map can all be extended to the concurrent case, and similar results about existence of a supervisory control map and feasibility of synthesizing supremal concurrently controllable and normal sublanguages as those for the RW asynchronous case can be derived. A concrete computational procedure has been provided to compute such a supremal sublanguage.

References

[1] A. Arnold. Finite transition systems: semantics of communicating systems. Univ. Bordeaux-I, France, 1994.

[2] Y. Li, W. M. Wonham. On supervisory control of real-time discrete-event systems. *Information Sciences*, 46(3):159-183, 1988.

[3] F. Lin, W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173-198, 1988.

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

[5] I. Romanovski, P.E. Caines. On the supervisory control of multiagent product systems. *IEEE Trans. Autom. Control*, 51(5):794-799, 2006.

[6] S. Takai, T. Ushio. Supervisory control of a class of concurrent discrete event systems. *IEICE Trans. Fundam.*, E87-A(3):850-855, 2004.

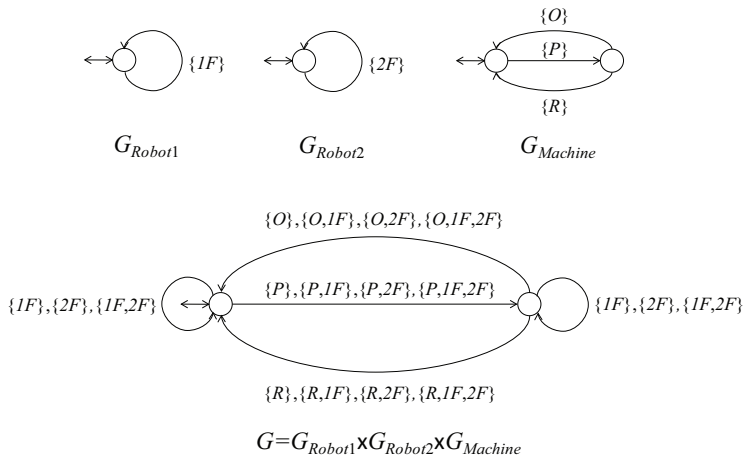

Fig. 3. Example 3: Component models

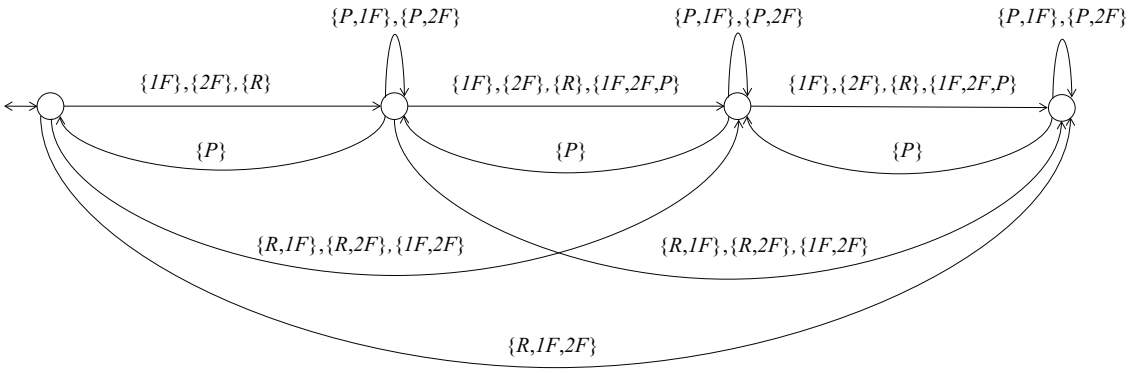

Fig. 4. Example 3: Requirement *SPEC*

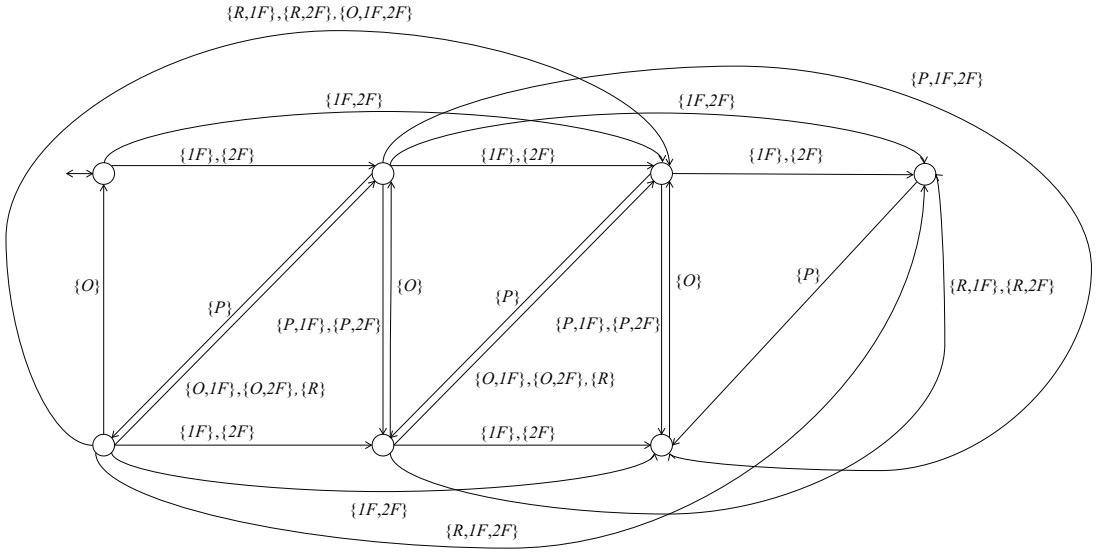

Fig. 5. Example 3: $\mathcal{CN}(G, SPEC)$

[7] S. Takai, T. Ushio. Supervisory control of a class of concurrent discrete event systems under partial observation. *JDEDS*, 15(1):7-32, 2005.

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

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