Synthesis of Maximally Permissive Supervisors for Partially-Observed Discrete-Event Systems

Xiang Yin, *Student Member, IEEE*, and Stéphane Lafortune, *Fellow, IEEE*

***Abstract*—We present new results on the synthesis of safe, non-blocking, and maximally permissive supervisors for partially observed discrete event systems. We consider the case where the legal language is a non-prefix-closed sublanguage of the system language, i.e., non-blockingness must be ensured in addition to safety. To solve this problem, we define a new bipartite transition system, called the Non-blocking All Inclusive Controller (NB-AIC), that embeds all safe and non-blocking supervisors. We present an algorithm for the construction of the NB-AIC and discuss its properties. We obtain the necessary and sufficient conditions for the solvability of the maximally permissive control problem. We then provide a synthesis algorithm, based on the NB-AIC, that constructs a supervisor that is safe, non-blocking and maximally permissive, if one exists. This is the first algorithm with such properties.**

***Index Terms*—Discrete-event systems (DES), maximal permissiveness, partial observation, supervisory control, synthesis.**

I. Introduction

THE problem under consideration in this paper is that of control of partially observed Discrete Event Systems (DES) in the framework of the supervisory control theory initiated by Ramadge and Wonham [1]. This control problem for DES arises in the study of automated systems where the behavior is inherently event-driven, as well as in the study of discrete abstractions of continuous, hybrid, and/or cyber-physical systems. Due to the limited actuating and sensing capabilities in the plant, the DES is partially controlled and partially observed. Formally, using standard notation [2], the problem addressed in this paper is the following: Given a plant modeled by automaton G, whose event set includes uncontrollable events and unobservable events, and given a non-prefix-closed specification language $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$ where H is a trim automaton, synthesize a supervisor S_P for G such that $\mathcal{L}_m(S_P/G) \subseteq \mathcal{L}_m(H)$ (the *safety* specification) and $\mathcal{L}(S_P/G) = \overline{\mathcal{L}_m(S_P/G)}$ (the *non-blocking* specification).

Supervisory control of centralized and partially observed DES was initially studied in [3], [4], in which the necessary and sufficient conditions for exactly achieving a specification language were given. These are the well-known controllability, observability, and $\mathcal{L}_m(G)$-closure conditions. When the given specification language cannot be exactly achieved, one is interested in synthesizing solutions that are not only safe and non-blocking, but also *maximally permissive* in the sense that there does not exist another solution that is strictly larger and is still safe and non-blocking; in other words, such solutions are *locally maximal*. Since observability may not be preserved under union, no supremal solution exists in general (unless additional assumptions are made).

Many approaches have been considered in the literature for synthesizing safe and non-blocking supervisors for partially observed DES; see, e.g., [5]–[12]. One approach is to find the supremal controllable *normal* and $\mathcal{L}_m(G)$-closed sublanguage of $\mathcal{L}_m(H)$, as initially defined in [3], [4]; see also, e.g., [5], [6], [13] for computational algorithms. However, since normality is stronger than observability, such a solution may be too restrictive (even empty). In [7], [8], solutions that are provably larger than the supremal controllable normal sublanguage are provided. In [7], the authors identified a class of observable sublanguages that is invariant under the specifically defined "strict subautomaton union" operation. In [8], the authors identified a new language property, called relative observability, that is stronger than observability, weaker than normality, and preserved under the standard union of languages. The authors also provided an algorithm to compute the supremal controllable and relative observable sublanguage. The solutions obtained by the techniques of [7] and [8] are incomparable and neither of them is maximal in general. Moreover, both techniques may return empty solutions even when non-empty solutions exist. The decidability of the problem of synthesizing a non-empty solution, i.e., a solution that is both safe and non-blocking, was established in [9]. If the decidability condition holds, in [14], the authors provided an algorithm that always returns a non-empty solution; however, the solution obtained is not maximal in general.

On-line and off-line approaches have been developed to compute maximal controllable and observable solutions when the non-blockingness requirement is relaxed, i.e., when the specification is given by a prefix-closed language; see, e.g., [15]–[17]. However, these approaches cannot be applied to the case where the specification is described by a non-prefix-closed language, since the resulting solutions may be blocking. Besides these, some other approaches have also been considered in the literature. In [9], [10], the use of nondeterministic supervisors was advocated. In [11], [12], game-theoretic approaches were considered for the synthesis of supervisors. But the frameworks adopted in these works are different from what we consider in

Manuscript received July 27, 2014; revised November 13, 2014, April 4, 2015, June 5, 2015, and June 29, 2015; accepted July 21, 2015. Date of publication July 23, 2015; date of current version April 22, 2016. This work was supported in part by NSF grants CCF-1138860 (Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering) and CNS-1446298. Recommended by Associate Editor S. Takai.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: xiangyin@umich.edu; stephane@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2015.2460391

this paper. To the best of our knowledge, the synthesis of non-blocking and safe deterministic supervisors that are maximally permissive for partially observed DES has remained an open problem, that we solve in this paper.

We present a new algorithm for synthesizing supervisors for partially observed DES. Our approach is based on the construction of new finite-state bipartite transition structures, using suitably defined information states. We start from the safety specification, by defining a finite bipartite transition system, called the "All Inclusive Controller" (or AIC hereafter), which embeds in its structure all *safe* control decisions. Then we consider non-blockingness in addition to safety, and define another finite bipartite transition system that we call the "Non-Blocking All Inclusive Controller" (or NB-AIC hereafter). The NB-AIC contains in its transition structure all supervisors that are safe and *deadlock-free*. Based on the NB-AIC, we present a synthesis algorithm that results in a safe, *non-blocking* and *maximally permissive* supervisor. Recall that a supervisor is non-blocking if it is both deadlock-free and livelock-free.

The reminder of this paper is organized as follows. In Section II, we revisit some basic terminologies in supervisory control theory and formulate the problem we want to solve. The main contributions of this paper are presented in Sections III–VII, and include the following:

- The definition of a new class of bipartite transition systems that are formulated as game structures between the supervisor and the system (Section III).
- The characterization of transition structures that represent *all* desired supervisors for prefix-closed and non-prefix-closed specification languages, namely the AIC and NB-AIC, respectively (Sections IV and V).
- The construction algorithms for the AIC and NB-AIC (Sections IV and V).
- The necessary and sufficient conditions for the solvability of the problem of synthesizing safe, non-blocking, and maximally permissive supervisors under the partial observation assumption (Section VI).
- An algorithm based on the NB-AIC that returns a solution to the above problem (if one exists) and the correctness proof of the proposed algorithm (Section VI).
- An illustrative example of our synthesis algorithm, for which previous approaches return empty solutions (Section VII).

Finally, we conclude the paper in Section VIII. In addition, Appendix A discusses in more detail implementation issues that arise in the synthesis algorithm of Section VI. The computational complexity of the synthesis algorithm of Section VI is analyzed in Appendix B. Preliminary and partial versions of some of the results in this paper are presented in [18], [19].

II. Problem Formulation

A. System Model

We assume basic knowledge of DES and common notations (see, e.g., [2]). We model a DES as a deterministic finite-state automaton $G = (X, E, f, x_0, X_m)$, where X is the finite set of states, E is the finite set of events, $f : X \times E \to X$ is the partial transition function where $f(x, e) = y$ means that there is a transition labelled by event e from state x to state y, x_0 is the initial state, and X_m is the set of marked states. f is extended to $X \times E^*$ in the usual way. The behavior generated by G is described by $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$, where ! means "is defined"; the marked behavior is $\mathcal{L}_m(G) = \{s \in E^* : f(x_0, s) \in X_m\}$. The prefix-closure of a language L is $\overline{L} = \{s \in E^* : (\exists t \in E^*)[st \in L]\}$; we say that L is prefix-closed if $L = \overline{L}$.

In the supervisory control framework initiated in [1], a supervisor is imposed on G to achieve some specification by dynamically enabling/disabling events. The event set E is partitioned into two disjoint subsets: E_c, the subset of controllable events, and E_{uc}, the subset of uncontrollable events. Since some of the events may not be observed [3], E is also partitioned into the subset of observable events, E_o, and the subset of unobservable events, E_{uo}. The natural projection, $P : E^* \to E_o^*$ is defined by

$$P(\epsilon) = \epsilon, \quad P(s\sigma) = \begin{cases} P(s)\sigma & \text{if } \sigma \in E_o \\ P(s) & \text{if } \sigma \in E_{uo}. \end{cases}$$

We say that a control decision $\gamma \in 2^E$ is admissible if $E_{uc} \subseteq \gamma$ and define $\Gamma = \{\gamma \in 2^E : E_{uc} \subseteq \gamma\}$ as the set of admissible control decisions. A partial observation supervisor is a function $S_P : \mathcal{L}(G) \to \Gamma$ such that $\forall s, t \in \mathcal{L}(G) : P(s) = P(t) \Rightarrow S_P(s) = S_P(t)$. We use the notation S_P/G to represent the controlled system and the language generated by S_P/G, denoted by $\mathcal{L}(S_P/G)$, is defined recursively as follows:

i) $\epsilon \in \mathcal{L}(S_P/G)$; and
ii) $[s \in \mathcal{L}(S_P/G) \wedge s\sigma \in \mathcal{L}(G) \wedge \sigma \in S_P(s)] \Leftrightarrow [s\sigma \in \mathcal{L}(S_P/G)]$.

We also define $\mathcal{L}_m(S_P/G) = \mathcal{L}(S_P/G) \cap \mathcal{L}_m(G)$.

B. Supervisory Control Under Partial Observation

First, we revisit some common terminology in the DES literature. Let language $K \subseteq \mathcal{L}_m(G)$ be a non-prefix-closed language that represents the desired (safe and non-blocking) behavior to be achieved under control. K is said to be: (i) *controllable* (w.r.t. G and E_{uc}) if $\overline{K}E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}$; (ii) *observable* (w.r.t. G, E_c and E_o) if for all $s \in \overline{K}$ and $\sigma \in E_c$ such that $s\sigma \in \overline{K}$, $P^{-1}P(s)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}$, where $P^{-1}(s) := \{t \in E^* : P(t) = s\}$; and (iii) $\mathcal{L}_m(G)$-closed if $\overline{K} \cap \mathcal{L}_m(G) = K$. It has been shown in [3] that these three conditions together provide the necessary and sufficient conditions for the existence of a supervisor that exactly achieves the given language K, as formally stated in the following theorem.

Theorem II.1. (Controllability and Observability Theorem, [3]): Consider DES G, with E_c and E_o. Consider also $K \subseteq \mathcal{L}_m(G)$. There exists a supervisor $S_P : \mathcal{L}(G) \to \Gamma$ such that $\mathcal{L}_m(S_P/G) = K$ and $\mathcal{L}(S_P/G) = \overline{K}$ *iff*

1) K is controllable w.r.t. $\mathcal{L}(G)$ and E_{uc};
2) K is observable w.r.t. $\mathcal{L}(G)$, E_o and E_c;
3) K is $\mathcal{L}_m(G)$-closed.

If a given specification language cannot be exactly achieved, then the synthesis problem asks whether we can find a controllable, observable and $\mathcal{L}_m(G)$-closed *sublanguage* of the specification language, and one that is "as large as possible."

Formally, we formulate the *Basic Supervisory Control and Observation Problem: Non-blocking and Maximally Permissive Case* (BSCOP-NB$^{\max}$) addressed in this paper as follows.

Definition II.1. (BSCOP-NB$^{\max}$): Given DES G and specification $K \subseteq \mathcal{L}_m(G)$, find a supervisor S_P such that

(i) $\mathcal{L}_m(S_P/G) \subseteq K$ and $\mathcal{L}(S_P/G) = \overline{\mathcal{L}_m(S_P/G)}$;
(ii) There does not exist a supervisor S'_p satisfying (i) such that $\mathcal{L}_m(S_P/G) \subset \mathcal{L}_m(S'_P/G)$.

We also define problem BSCOP-NB if (ii) is relaxed, i.e., we only require S_P satisfying (i).

Unlike the fully observed case, in which a supremal solution always exists, no supremal solution exists for BSCOP-NB$^{\max}$ since observability is not preserved under union, in general. Consequently, there may be several incomparable maximal solutions (w.r.t. set inclusion) for BSCOP-NB$^{\max}$. To guarantee the existence of a supremal solution, additional assumptions are needed. For instance, if $E_c \subseteq E_o$, then controllability and observability together imply normality (see, e.g., [2]), which means that the supremal controllable and normal sublanguage solves BSCOP-NB$^{\max}$. However, this assumption is not required in this paper.

Let $K = \mathcal{L}_m(H) = \mathcal{L}(H) \cap \mathcal{L}_m(G)$, for some trim (i.e., accessible and co-accessible) automaton $H = (X_H, E, f_H, x_{H,0}, X_{H,m})$. Hereafter, we assume, without loss of generality, that H and G satisfy the following properties: (i) H is a sub-automaton of G (as defined in [2]); (ii) if $x, y \in X_H$ and $f(x,\sigma) = y$ then $f_H(x,\sigma)$ is defined and $f_H(x,\sigma) = y$. In words, all states of H are *legal* and *all transitions in G between legal states are also legal* (and thus in H). This assumption is usually referred to as "H is a *strict* sub-automaton of G" in the literature. If the original G and H do not satisfy the above conditions, the algorithm in [5] (see also the appendix of [17]) can be used to refine both of them and ensure that (i) and (ii) hold. This algorithm involves taking the product of the original G with a suitably modified H with a completed transition function and extracting from it the new G and H automata. Therefore, the automata obtained are at most of quadratic size as compared with the original automata. This refinement of H and G (if needed) will simplify the subsequent analysis. Namely, we can talk of the legality of *states* of X rather than of *strings* of $\mathcal{L}(G)$: a state $x \in G$ is legal (safe) iff $x \in X_H$.

For later use, we define the following terminology for a prefix-closed language $L \subseteq \mathcal{L}(G)$, given automaton G: (i) L is *deadlock-free* (w.r.t. G) if $(\forall s \in L)[\delta_L(s) = \emptyset \Rightarrow f(x_0, s) \in X_m]$, where $\delta_L(s) := \{e \in E : se \in L\}$; (ii) L is *non-blocking* (w.r.t. G) if $\overline{L \cap \mathcal{L}_m(G)} = L$; and (iii) L is safe (w.r.t. K) if $L \subseteq \overline{K}$. By the controllability and observability theorem, BSCOP-NB$^{\max}$ and the problem of finding a maximal controllable, observable, safe and non-blocking sublanguage of $\overline{K}$ are equivalent.

Given an automaton G, an *execution* is a sequence $\langle x_1, \sigma_1, \ldots, \sigma_{k-1}, x_k\rangle$, where $x_i \in X, \sigma_i \in E$ and $x_{i+1} = f(x_i, \sigma_i)$, $\forall i \in \{1, 2, \ldots, k-1\}$. We say that an execution forms a *cycle* if $x_1 = x_k$; we say that a cycle is an *elementary cycle* if $\forall i, j \in \{1, 2, \ldots, k-1\} : i \neq j \Rightarrow x_i \neq x_j$. A strongly connected component (SCC) in G is a maximal set of states $C \subseteq X$ such that $\forall x, y \in C, \exists s \in E^* : f(x, s) = y$; a SCC C is said to be non-trivial if $\forall x, y \in C, \exists s \in E^* \setminus \{\epsilon\} : f(x, s) = y$. A *livelock* in G is a non-trivial SCC C such that: (i) $C \cap X_m = \emptyset$, i.e., there is no marked state in it; and (ii) $\forall x \in C, \forall \sigma \in E : f(x, \sigma) \in C$, i.e., there is no transition defined out of it. We say that $\langle x_1, \sigma_1, \ldots, \sigma_{k-1}, x_k\rangle$ is an *elementary livelock cycle* if: (i) it is an elementary cycle; and (ii) there exists a livelock C, such that $\{x_1, x_2, \ldots, x_{k-1}\} \subseteq C$. We say that $L \subseteq \mathcal{L}(G)$ is a *livelock language* if any automaton generating L contains a livelock; otherwise, we say that L is *livelock-free*. Clearly, L is non-blocking if and only if it is both deadlock-free and livelock-free.

Finally, we define three operators that will be used in this paper.

The *Unobservable Reach* of the subset of states $S \subseteq X$ under the subset of events $\gamma \subseteq E$ is given by

$$\mathrm{UR}_\gamma(S) \coloneqq \{x \in X : \exists u \in S, \exists e \in (E_{uo} \cap \gamma)^* \text{ s.t. } x = f(u, e)\}. \tag{1}$$

The *Extended Unobservable Reach* of the subset of states $S \subseteq X$ under the subset of events $\gamma \subseteq E$ is given by

$$\mathrm{UR}^+_\gamma(S) := \mathrm{UR}_\gamma(S) \cup \{x \in X : (\exists u \in \mathrm{UR}_\gamma(S))(\exists e \in (E_o \cap \gamma)) \text{ s.t. } x = f(u, e)\}. \tag{2}$$

The *Observable Reach* of the subset of states $S \subseteq X$ under observable event $e \in E_o$ is given by

$$\mathrm{Next}_e(S) := \{x \in X : \exists u \in S \text{ s.t. } x = f(u, e)\}. \tag{3}$$

III. Bipartite Transition System

A. Bipartite Transition System

We start by defining the general notion of a *Bipartite Transition System* (denoted by BTS hereafter). Let an information state (denoted by IS herafter) be a subset $IS \subseteq X$ of states and denote by $I = 2^X$ the set of all information states.

Definition III.1. (Bipartite Transition System): A bipartite transition system T w.r.t. G is a 7-tuple

$$T = \left(Q_Y^T, Q_Z^T, h_{YZ}^T, h_{ZY}^T, E, \Gamma, y_0^T\right) \tag{4}$$

where

- $Q_Y^T \subseteq I$ is the set of Y-states;
- $Q_Z^T \subseteq I \times \Gamma$ is the set of Z-states and $I(z)$ and $\Gamma(z)$ denote, respectively, the information state and the control decision components of a Z-state z, so that $z = (I(z), \Gamma(z))$;
- $h_{YZ}^T : Q_Y^T \times \Gamma \to Q_Z^T$ is the partial transition function from Y-states to Z-states, which satisfies the following constraint: for any $y \in Q_Y^T, z \in Q_Z^T$ and $\gamma \in \Gamma$, we have

$$h_{YZ}^T(y, \gamma) = z \Rightarrow [I(z) = \mathrm{UR}_\gamma(y)] \wedge [\Gamma(z) = \gamma] \tag{5}$$

- $h_{ZY}^T : Q_Z^T \times E \to Q_Y^T$ is the partial transition function from Z-states to Y-states, which satisfies the following constraint: for any $y \in Q_Y^T, z \in Q_Z^T$ and $e \in E$, we have

$$h_{ZY}^T(z, e) = y \Rightarrow [e \in \Gamma(z) \cap E_o] \wedge [y = \mathrm{Next}_e(I(z))] \tag{6}$$

- E is the set of events of G;
- Γ is the set of admissible control decisions of G;
- $y_0^T \in Q_Y^T$ is the initial Y-state where $y_0^T = \{x_0\}$.

Intuitively, a BTS is a "game structure" between the controller and the system. A Y-state is an information state where control decisions are made (i.e., the controller plays). A Z-state is an information state augmented with an admissible control decision, i.e., $z = (I(z), \Gamma(z))$, from which observable events occur (i.e., the system plays). A transition from a Y-state to a Z-state represents the unobservable reach and "remembers" the set of enabled events from the Y-state that leads to it. This means that $I(z)$ is the set of states reachable from some state in the preceding Y-state through some string of enabled unobservable events, and that $\Gamma(z)$ is the control decision made in the preceding Y-state. A transition from Z-state z to Y-state y labeled by $e \in E_o$ represents an observable reach of G. This means that y is the set of states reachable from some state of the information state component of z through enabled observable event e, according to the definition of the Next function. Finally, we call a sequence in the form of $c_1\sigma_1 \ldots c_n\sigma_n, c_i \in \Gamma, \sigma_i \in E_o$ such that $h^T_{YZ}(y, c_1)!, h^T_{ZY}(h^T_{YZ}(y, c_1), \sigma_1)!, \ldots$, a *run* in T from y.

The definition of a BTS is based on the plant G. For simplicity, we will omit "with respect to G" in the remainder, if it is clear which plant G is being considered. Since the control decision for a Y-state may not be unique (i.e., a Y-state may have several distinct successor Z-states), given a BTS T, we define $C_T(y) := \{\gamma \in \Gamma : h^T_{YZ}(y, \gamma)!\}$, to be the set of control decisions defined at $y \in Q^T_Y$.

B. Total Controller and BTS Included Supervisor

We discuss the connection between BTS and supervisors in this section.

Definition III.1 provides a general definition for a BTS. However, for the purpose of control, we also want a BTS to satisfy the two following conditions: (i) for any reachable Y-state, there *exists* at least one control decision at that state and; (ii) *for all enabled* observable events at a Z-state, their corresponding transitions should be defined if they exist in G. This leads to the notion of a *complete* BTS.

Definition III.2: A BTS T is said to be *complete* if:

1) $(\forall y \in Q^T_Y)[C_T(y) \neq \emptyset]$ and;
2) $(\forall z \in Q^T_Z)(\forall e \in \Gamma(z) \cap E_o)[(\exists x \in I(z) : f(x, e)!) \Rightarrow h^T_{ZY}(z, e)!]$.

Note that "disable all (controllable) events" is a valid control decision, but $C_T(y) = \emptyset$ means there is *no* control decision, which is not valid.

We are now ready to define an important type of BTS called the Total Controller (denoted by TC herafter), which embeds all possible behaviors between the controller and the plant.

Definition III.3. (Total Controller): The total controller for G is defined as the BTS $\mathcal{TC}(G) = (Q^{TCG}_Y, Q^{TCG}_Z, h^{TCG}_{YZ}, h^{TCG}_{ZY}, E, \Gamma, y^{TCG}_0)$, where: (i) h^{TCG}_{YZ} contains all admissible transitions from Y-states to Z-states, i.e., all admissible control decisions at the respective states of G; and (ii) h^{TCG}_{ZY} contains all admissible transitions from Z-states to Y-states, i.e., all feasible and enabled observable events at the respective states of G. Specifically, h^{TCG}_{YZ} and h^{TCG}_{ZY} are obtained by replacing "$\Rightarrow$" in (5) and (6) in Definition III.1 with "$\Leftrightarrow$".

Since $\mathcal{TC}(G)$ has transitions defined for all admissible control decisions after each observation and for all possible event observations after each control decision, it contains all strings in $\mathcal{L}(G)$ and also every admissible supervisor. As a consequence, this structure contains all possible supervisors and possible languages under control, no matter safe or unsafe, or blocking or non-blocking.

There is a special kind of Z-states in $\mathcal{TC}(G)$ that have no successors. We say that a Z-state z is *terminal* if $(\forall x \in I(z))(\forall e \in E_o \cap \Gamma(z))[f(x, e)$ is not defined$]$. Consequently, $\mathcal{TC}(G)$ can only end up with terminal Z-states.

Definition III.4: Given a supervisor S_P and string $s \in \mathcal{L}(S_P/G)$, $IS^Y_{S_P}(y, s)$ is defined to be the Y-state that results from the occurrence of string s, when starting in Y-state y. This can be computed recursively as follows:

$$IS^Y_{S_P}(y, \epsilon) := y$$
$$IS^Y_{S_P}(y, s\sigma) := \begin{cases} h^{TCG}_{ZY}(h^{TCG}_{YZ}(IS^Y_{S_P}(y, s), S_P(s)), \sigma), & \text{if } \sigma \in E_o \cap S_P(s) \\ IS^Y_{S_P}(y, s), & \text{if } \sigma \in E_{uo} \cap S_P(s) \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

For brevity, we write $IS^Y_{S_P}(y^{TCG}_0, s)$ as $IS^Y_{S_P}(s)$.

Also, $IS^Z_{S_P}(z, s)$ is defined analogously as

$$IS^Z_{S_P}(z, \epsilon) := z$$
$$IS^Z_{S_P}(z, s\sigma) := \begin{cases} h^{TCG}_{YZ}(h^{TCG}_{ZY}(IS^Z_{S_P}(z, s), \sigma), S_P(s\sigma)), & \text{if } \sigma \in E_o \cap S_P(s) \\ IS^Z_{S_P}(z, s), & \text{if } \sigma \in E_{uo} \cap S_P(s) \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

For brevity, we write $IS^Z_{S_P}(z_0, s)$ as $IS^Z_{S_P}(s)$, where $z_0 = h^{TCG}_{YZ}(y^{TCG}_0, S_P(\epsilon))$.

In the above definition, we use h^{TCG}_{ZY} and h^{TCG}_{YZ} to evaluate the information state evolution, since $\mathcal{TC}(G)$ captures *all* possible transitions, hence it captures the control actions of S_P. For simplicity, we will drop the superscript hereafter and write h^{TCG}_{ZY} and h^{TCG}_{YZ} as h_{ZY} and h_{YZ}, respectively.

Now, given a complete BTS, it is possible for us to decode supervisors from it, as we explain next.

Defintion III.5: Given a complete BTS T, a supervisor S_P is said to be *included in* T if

$$(\forall s \in \mathcal{L}(S_P/G)) \left[S_P(s) \in C_T\left(IS^Y_{S_P}(s)\right)\right].$$

$\mathcal{S}(T)$ denotes the set of all supervisors included in T.

Definition III.6: Given a complete BTS T, a language L is said to be *generated by* T if $(\exists S_P \in \mathcal{S}(T))[\mathcal{L}(S_P/G) = L]$. $\mathcal{L}_{TS}(T)$ denotes the set of all languages generated by T.

Lemma III.1: Given a supervisor S_P, for any string $s \in \mathcal{L}(S_P/G)$, we have $I(IS^Z_{S_P}(s)) = \{v \in X : \exists s' \in \mathcal{L}(S_P/G)$ s.t. $P(s) = P(s') \wedge v = f(x_0, s')\}$.

Proof: Follows from the relevant definitions, by induction on the length of s. ■

This lemma simply says that the information state reached by supervisor S_P upon the occurrence of string s is only determined by its projection $P(s)$. Thus, strings that have the same projection will lead to the same information state.

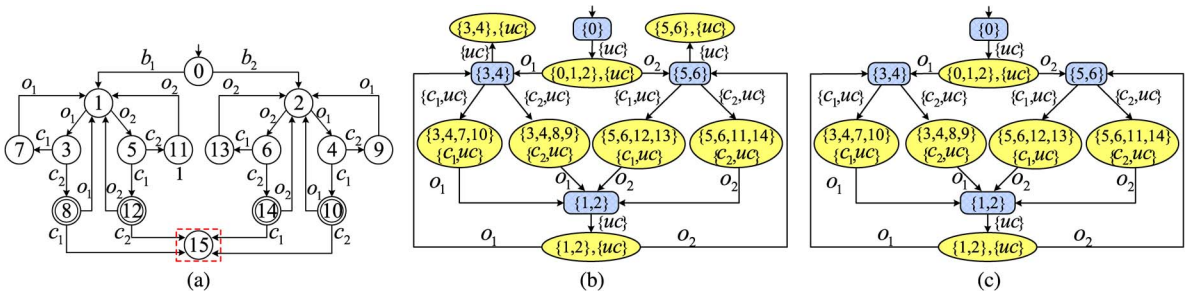

Fig. 1. An example of (NB-)AIC. For G: $E_c = \{c_1, c_2\}$, $E_o = \{o_1, o_2\}$ where state 15 is illegal. uc denotes all uncontrollable events. (a) Automaton G. (b) $\mathcal{AIC}(G)$. (c) $\mathcal{AIC}^{NB}(G)$.

IV. The All Inclusive Controller

In this section, we restrict our attention to the case of prefix-closed specifications, i.e., only safety needs to be ensured. We first define the recursive structure of the *All Inclusive Controller* that contains all solutions to the safety control problem. Then we discuss the properties of the All Inclusive Controller and provide its construction algorithm.

A. Definition of the AIC

Since any state in an information state reachable in a BTS is itself reachable from the initial state of G, we say that an information state $i \in I$ *violates safety* if there exists a state $x \in i$ such that $x \notin X_H$. (Recall that X_H is the set of legal states.) Then, we define the safety function for information state (w.r.t. X_H) $D_I : I \to \{0, 1\}$ by: $D_I(i) = 1 \Leftrightarrow i \subseteq X_H$. We say that a supervisor S_P maintains safety if $\mathcal{L}(S_P/G) \subseteq \mathcal{L}(H)$.

Theorem IV 1: Supervisor S_P maintains safety if and only if $\forall s \in \mathcal{L}(S_P/G) : D_I(I(IS^Z_{S_P}(s))) = 1$.

Proof: Follows from Lemma III.1. ■

The above theorem allows us to transfer the safety control problem under partial observation to the problem of finding a subsystem of the total controller in which all reachable states are safe. To formally describe this transformation, let us first define the following notions.

Definition IV.1: Given two BTSs $T_1 = (Q^{T_1}_Y, Q^{T_1}_Z, h^{T_1}_{YZ}, h^{T_1}_{ZY}, E, \Gamma, y^{T_1}_0)$ and $T_2 = (Q^{T_2}_Y, Q^{T_2}_Z, h^{T_2}_{YZ}, h^{T_2}_{ZY}, E, \Gamma, y^{T_2}_0)$, we say T_1 is a *subsystem* of T_2, denoted by $T_1 \sqsubseteq T_2$ if $Q^{T_1}_Y \subseteq Q^{T_2}_Y, Q^{T_1}_Z \subseteq Q^{T_2}_Z$ and for any $y \in Q^{T_1}_Y, z \in Q^{T_1}_Z, \gamma \in \Gamma$ and $e \in E$, we have that

1) $h^{T_1}_{YZ}(y, \gamma) = z \Rightarrow h^{T_2}_{YZ}(y, \gamma) = z$; and
2) $h^{T_1}_{ZY}(z, e) = y \Rightarrow h^{T_2}_{ZY}(z, e) = y$.

We define $T_1 \cup T_2 = (Q^{T_1}_Y \cup Q^{T_2}_Y, Q^{T_1}_Z \cup Q^{T_2}_Z, h^{T_1 \cup T_2}_{YZ}, h^{T_1 \cup T_2}_{ZY}, E, \Gamma, y^{T_1 \cup T_2}_0)$ to be the *union* of T_1 and T_2 if for any $y \in Q^{T_1}_Y \cup Q^{T_2}_Y, z \in Q^{T_1}_Z \cup Q^{T_2}_Z, \gamma \in \Gamma$ and $e \in E$, we have that

1) $h^{T_1 \cup T_2}_{YZ}(y, \gamma) = z \Leftrightarrow \exists i \in \{1, 2\} : h^{T_i}_{YZ}(y, \gamma) = z$; and
2) $h^{T_1 \cup T_2}_{ZY}(z, e) = y \Leftrightarrow \exists i \in \{1, 2\} : h^{T_i}_{ZY}(z, e) = y$.

We are now ready to define the structure of the All Inclusive Controller that contains all safe solutions to the control problem.

Definition IV.2. (All Inclusive Controller): The All Inclusive Controller for G, $\mathcal{AIC}(G) = (Q^{AICG}_Y, Q^{AICG}_Z, h^{AICG}_{YZ}, h^{AICG}_{ZY}, E, \Gamma, y^{AICG}_0)$, is defined as the largest subsystem of $\mathcal{TC}(G)$ such that

1) $\mathcal{AIC}(G)$ is complete;
2) $\forall z \in Q^{AICG}_Z : D_I(I(z)) = 1$.

By largest subsystem, we mean that for any $T \sqsubseteq \mathcal{TC}(G)$ satisfying 1) and 2), we have that $T \sqsubseteq \mathcal{AIC}(G)$.

Remark 1: In the above definition, the largest subsystem of the TC is uniquely defined, since by Definition IV.1, the TC only has a finite number of subsystems and the union of any two subsystems satisfying the above properties still satisfies these properties. In the definition, we can also add another condition that $\forall y \in Q^{AICG}_Y : D_I(y) = 1$. However, since $y \subseteq I(z)$ whenever $z = h_{YZ}(y, \gamma)$, the above definition suffices. In the remainder of this paper, we only consider the reachable part of the AIC, i.e., we assume that all Y and Z-states in the AIC are reachable from the initial state of the AIC. Formally, we assume that for any Y-state $y \in Q^{AICG}_Y$ (respectively, Z-state $z \in Q^{AICG}_Z$) there exists $S_P \in \mathcal{S}(\mathcal{AIC}(G))$ and a string $s \in \mathcal{L}(S_P/G)$ such that $IS^Y_{S_P}(s) = y$ (respectively, $IS^Z_{S_P}(s) = z$).

Example IV.1: Let G be the automaton shown in Fig. 1(a). The resulting AIC of G is shown in Fig. 1(b) (a formal construction algorithm will be described later). In the diagram of the AIC, (blue) rectangular states correspond to Y-states and (yellow) oval states correspond to Z-states. For Y-state $\{3, 4\}$, we can make control decision $\{c_1\}$ or $\{c_2\}$, however, we cannot make control decision $\{c_1, c_2\}$, since it will unobservably lead to illegal state 15 before a new event is observed. The same is true for Y-state $\{5, 6\}$.

Remark 2: In Fig. 1(b), at the Y-state $y_0 = \{0\}$, we can also make control decision $\{c_1, uc\}$. However, c_1 will never be executed within its unobservable reach. Formally, we say that a control decision $\gamma \in \Gamma$ is *irredundant* at $i \in I$ if $(\forall e \in \gamma)(\exists x \in \mathrm{UR}_\gamma(i))[f(x, e)!]$. Hereafter, we only keep irredundant control decisions in the AIC; this will not affect its properties.

B. Properties of the AIC

The following results show that the AIC structure defined in the preceding section contains and only contains all solutions to the safety control problem. Due to space constraints, their proofs have been omitted and they are available in [20].

Theorem IV.2: There exists a supervisor S_P for system G such that $\mathcal{L}(S_P/G) \subseteq \mathcal{L}(H)$ *iff* $\mathcal{AIC}(G)$ exists.

Theorem IV.3: If $\mathcal{AIC}(G)$ exists, then

$$(\overline{L} = L \neq \emptyset \wedge L \subseteq \mathcal{L}(H) \wedge L \text{ is controllable and observable}) \Leftrightarrow L \in \mathcal{L}_{TS}(\mathcal{AIC}(G))$$

C. Construction Algorithm

Definition IV.2 provides us with a direct way of constructing the AIC. One way to proceed is to first build the total controller TC and then search through the whole (Y and Z) state space to iteratively prune states that violate the safety condition or completeness condition. This approach does work, but it is not the most computationally efficient. Here, we provide an alternative approach that replaces this whole search by a *single* calculation each time we would like to determine the safety of a particular information state. First, inspired by the technique used in [21], we define the set $X_e \subseteq X$ called the "extended specification". In words, the extended specification is defined as the set of states that should never be reached because even if all events in E_c are disabled forever thereafter, there still exists some sequence of events such that some illegal state will be reached.

Definition IV.3: The extended specification (w.r.t. the set of illegal states X_H) $X_e \subseteq X$ is defined by $X_e := \{x \in X : \exists s \in E_{uc}^* \text{ s.t. } f(x,s) \notin X_H\}$.

Note that the extended specification can be calculated off-line all at once or online if so desired (see, e.g., [21]). In the remainder of this paper, we assume that the extended specification has already been pre-calculated.

Similarly to the safety function, by using the extended specification, we define the extended safety function $D_I^e : I \to \{0,1\}$ by $D_I^e(i) = 1 \Leftrightarrow i \cap X_e = \emptyset$.

The following theorem shows how the states of the AIC can be easily determined by using the extended specification. Its proof is available in [20].

Theorem IV.4: For any Y-state y, suppose that $y \in Q_Y^{AICG}$, then for any control decision $\gamma \in \Gamma$, $\gamma \in C_{\mathcal{AIC}(G)}(y)$ if and only if $D_I^e(\text{UR}_\gamma^+(y)) = 1$.

Theorem IV.4 provides an easy way to construct the AIC. At any Y-state, we can determine whether or not taking control decision γ is safe by verifying whether or not $\text{UR}_\gamma^+(y)$ satisfies the extended specification. This idea is implemented by Algorithm FIND-AIC, in which parameter AIC represents the AIC we need to construct, with $AIC.Y$ and $AIC.Z$ being its set of Y and Z states, respectively, and $AIC.h$ being its transition function. Initially, we need to check whether or not there exists an uncontrollable string from the initial state to an illegal state. If not, we know that the AIC exists and we set $AIC.Y$ to $\{y_0\}$ and then start the depth first search, which is implemented by the procedure DoDFS. Lines 7 and 8 are used to find the safe control decisions. This is done by considering each admissible control decision $\gamma \in \Gamma$ and determining whether it is safe or not. For each control decision, we compute its extended unobservable reach, and determine the value of $D_I^e(\text{UR}_\gamma^+(y))$. If the control decision is safe, then we move to the successor Z-state $h_{YZ}(y,\gamma)$ and for each such Z-state, we compute all possible Y-state successors and make a recursive call. This recursive procedure allows us to traverse the whole reachable space of Y and Z-states. Since the number of information states is finite, the algorithm will eventually terminate.

Algorithm 1 $AIC \leftarrow$ FIND-AIC(G)

```
1: If ∃s ∈ E*_uc : f(x0, s) ∉ X_H then
2:     return the AIC does not exist
3: end if
4: AIC.Y ← {y0}, AIC.Z ← ∅, and AIC.h ← ∅
5: DoDFS(G, y0, AIC)
6: Procedure DoDFS(G, y, AIC)
7:     for all γ ∈ Γ do
8:         if D^e_I(UR^+_γ(y)) = 1 then
9:             z ← h_YZ(y, γ)
10:            AIC.h ← AIC.h ∪ {(y, γ, z)}
11:            if z ∉ AIC.Z then
12:                AIC.Z ← AIC.Z ∪ {z}
13:                for all e ∈ γ ∩ E_o do
14:                    y' ← Next_e(I(z))
15:                    AIC.h ← AIC.h ∪ {(z, e, y')}
16:                    if y' ∉ AIC.Y then
17:                        AIC.Y ← AIC.Y ∪ {y'}
18:                        DoDFS(G, y', AIC)
19:                    end if
20:                end for
21:            end if
22:        end if
23:    end for
24: end procedure
```

Proposition IV.1: The running time of FIND-AIC is of $O(|X||E|2^{|X|+|E_c|})$.

Proof: Algorithm FIND-AIC is implemented by a search through the space of Y-states, and in the worst case there are $2^{|X|}$ Y-states. For each Ystate encountered, a maximum of $2^{|E_c|}$ control decisions should be considered. For each control decision, we need to compute the unobservable reach and the extended unobservable reach, which can be done together in $O(|X||E|)$ time. Checking whether the extended unobservable reach satisfies the extended specification or not can be done in time $O(|X|)$. Finally, there are at most $|E_o|$ successors from z, and we need to take $O(|X||E_o|)$ time to compute all of them. Combining the above together, the total running time is therefore $O([|X||E| + |X| + |X||E_o|]2^{|X|}2^{|E_c|})$, which can be simplified to $O(|X||E|2^{|X|+|E_c|})$. ■

V. The Non-Blocking All Inclusive Controller

In this section, we tackle the case of non-prefix-closed specifications. We first define the Non-Blocking AIC (NB-AIC), a bipartite transition system obtained from the AIC that contains all *safe and non-blocking* control policies; then we investigate its construction and properties.

A. Definition of the NB-AIC

Definition V.1. (Live Decision String): Given a BTS T, for any Y-state $y \in Q_Y^T$ and state $x \in y$ in it, we say that a decision

string $c_1c_2\dots c_n$, where $c_i \in \Gamma$ for $i = 1,\dots n$, is *live* for (y,x) in T if there exists a string $s = \xi_1\sigma_1\xi_2\dots\sigma_{n-1}\xi_n$, where $\xi_i \in (E_{uo}\cap c_i)^*, \sigma_i \in E_o \cap c_i$, such that $f(x,s) \in X_m$ and $\forall i < n : c_{i+1} \in C_T(y_i)$, where y_i is the unique Y-state following the run $c_1\sigma_1\dots\sigma_{i-1}c_i\sigma_i$ in T from y. We say that a Y-state y is *live* in T if for all $x \in y$, (y,x) has a live decision string.

Example V.1: Consider the automaton G and its corresponding AIC shown in Fig. 1. $\{uc\}\{c_2, uc\}$ is a live decision string for state $1 \in \{1,2\}$, since string o_1c_2, which leads state 1 to marked state 8, exists under this decision string.

Intuitively, the liveness property of a Y-state simply says that given a current information state, for each state in it, we can always find a sequence of control decisions under which this state will be able to reach some marked state through some string. The verification of the liveness property of a Y-state is a reachability problem in an automaton that is built from the original BTS by explicitly adding transitions to capture reachability within states in Z-states. Details can be found in the Appendix.

The purpose of the above notion of liveness of information states is to eliminate one source of blocking: clearly, if a Y-state is not live, then no matter what control decision we take at that Y-state, we will always be blocked by some state in it.

In the case of Z-states, we introduce a notion of deadlock-freeness to complement the notion of liveness of Y-states. Specifically, for a Z-state z, we require that any state $x \in I(z)$ should either have an unobservable path to a marked state or a path that goes outside of the Z-state; otherwise, it will also be a source of blocking. This leads to the following definition, which depends on Z-state z and on G, but not on the BTS that z is part of.

Definition V.2. (Deadlock-Free Z-State): A Z-state z is said to be *deadlock-free* if for all $x \in I(z)$ we have

$$(\exists s \in (\Gamma(z)\cap E_{uo})^*)[f(x,s) \in X_m] \vee$$
$$(\exists s \in (\Gamma(z)\cap E_{uo})^*(\Gamma(z)\cap E_o))[f(x,s) \text{ is defined}] \quad (7)$$

Otherwise, z is said to be a *deadlock* Z-state.

We are now ready to define the NB-AIC structure, which contains all safe and non-blocking solutions.

Definition V.3. Non-blocking All Inclusive Controller): The Non-Blocking All Inclusive Controller for G, $\mathcal{AIC}^{NB}(G) = (Q_Y^{NBG}, Q_Z^{NBG}, h_{YZ}^{NBG}, h_{ZY}^{NBG}, E, \Gamma, y_0^{NBG})$, is defined as the largest subsystem of $\mathcal{AIC}(G)$ such that

1) $\mathcal{AIC}^{NB}(G)$ is complete;
2) $\forall y \in Q_Y^{NBG} : y$ is live;
3) $\forall z \in Q_Z^{NBG} : z$ is deadlock-free.

In the above definition, the largest non-blocking subsystem of the AIC is uniquely defined, since the union of any subsystems satisfying the above properties still satisfies these properties. Similar to the case of the AIC, we also only consider the reachable part of the NB-AIC hereafter.

Example V.2: Going back to Fig. 1, the NB-AIC for G is shown in Fig. 1(c). Comparing with its AIC, since all Y-states in it are live, the deadlock Z-states that are removed are $(\{3,4\},\{uc\})$ and $(\{5,6\},\{uc\})$.

B. Properties and Construction Algorithm

By definition, the NB-AIC is also a complete BTS. Thus, we can talk about the properties of its generated language, which are given in the following theorem.

Theorem V.1: The language generated by the NB-AIC, $\mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, satisfies the following two properties:

1) If $L = \overline{L} \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, then L is controllable, observable, safe, and deadlock-free;
2) If $L = \overline{L}$ is non-empty, controllable, observable, safe, and non-blocking, then $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$.

Proof:

1) Since the NB-AIC is a subsystem of the AIC, we know that $\mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G)) \subseteq \mathcal{L}_{TS}(\mathcal{AIC}(G))$. Thus, L is controllable, observable and safe by Thm. IV.3. For $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, there exists $S_P \in \mathcal{S}(\mathcal{AIC}^{NB}(G))$ such that $\mathcal{L}(S_P/G) = L$. Now, let us assume that L has a deadlock, which implies that there exists $s \in L$ such that $f(x_0,s) \notin X_m$ and $\delta_L(s) = \emptyset$, where $\delta_L(s) := \{e \in E : se \in L\}$. In terms of information state evolution, we know that $f(x_0,s) \in IS^Z_{S_P}(s)$. By Def. V.2, this implies that the Z state $IS^Z_{S_P}(s)$ is a deadlock state, which contradicts the definition of the NB-AIC. Thus, L is deadlock-free.
2) We prove by contrapositive, i.e., we show that if $L \notin \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$ then L cannot cannot simultaneously be non-empty, controllable, observable, safe, and non-blocking. Since we know that $\mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G)) \subseteq \mathcal{L}_{TS}(\mathcal{AIC}(G))$, there are two cases for $L \notin \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$:

Case 1) $L \notin \mathcal{L}_{TS}(\mathcal{AIC}(G))$. By Thm. IV.3, L cannot be non-empty, controllable, observable and safe at the same time.

Case 2) $L \in \mathcal{L}_{TS}(\mathcal{AIC}(G))$ but $L \notin \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$. Since L is generated by the AIC, it is controllable, observable and safe and there exists S_P such that $\mathcal{L}(S_P/G) = L$. We now show that in this case L is blocking. By Def. V.3, it can be shown by contradiction that there exists $s \in L$ such that one of the two following cases holds: (i) $IS^Z_{S_P}(s)$ is a deadlock Z-state. By Def. V.2, L is blocking; (ii) $IS^Y_{S_P}(s)$ is not live. If $y = IS^Y_{S_P}(s)$ is not live, then by Def. V.1, there exists at least one state in y where no control decision can be made to lead it to a marked state. Specifically, $(\exists t \in \mathcal{L}(S_P/G) : P(t) = P(s))(\forall v \in E^* : tv \in \mathcal{L}(S_P/G))[f(x_0,tv) \notin X_m]$. Thus, L is blocking. ■

Note that for $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, L need not be livelock-free in general. Let us consider the automaton G in Fig. 2(a) and its corresponding NB-AIC shown in Fig. 2(a). Clearly, $(ab)^* \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, but it is a livelock language. However, the above statement is true when G is acyclic, i.e., there is no cycle in G, since in this case, the deadlock-freeness condition and the non-blockingness condition are equivalent. Therefore, we have the following result.

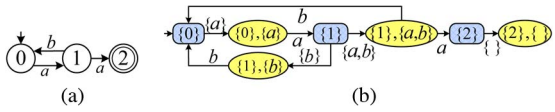

Fig. 2. For $G: E_{uo} = \emptyset$ and $E_{uc} = \{b\}$. (a) Automaton G. (b) The corresponding NB-AIC.

Corollary V.1: If G is acyclic, then $L = \overline{L}$ is nonempty, controllable, observable, safe and non-blocking *iff* $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$.

Algorithm 2 $\mathcal{AIC}^{NB}(G) \leftarrow$ FIND-NB-AIC(G)

1: $A \leftarrow$ FIND-AIC(G)
2: Delete all Z-states in A that are deadlock states
3: **while** exists Y-state in A that is not live **do**
4: Delete all Y-states in A that are not live
5: **while** exists Y-state in A that has no successor **do**
6: Delete all such Y-states in A and delete all their predecessor Z-states
7: **end while**
8: **end while**
9: **if** the initial Y-state has been removed **then**
10: **return** the NB-AIC does not exist
11: **end if**
12: $\mathcal{AIC}^{NB}(G) \leftarrow$ Accessible(A)

The construction procedure for the NB-AIC is given by Algorithm FIND-NB-AIC. The basic idea of the construction algorithm follows directly from the definition. We need to keep pruning states from the AIC structure until convergence. Specifically, there are three kinds of states that we need to prune:

(i) All Z-state that are deadlock states;
(ii) All Y-states that are not live; and
(iii) All Y or Z-states that violate the definition of completeness (Def. III.2).

In the algorithm, the elimination of (i), (ii) and (iii) are implemented in line-2, line-4 and line-6, respectively. Note that for (ii) and (iii), iteration steps are required, since pruning states may change the liveness or the completeness of the transition system. However, (i) just needs to be executed once, since the deadlock property does not depend on T.

Proposition V.1: The running time of FIND-NB-AIC is in $O(|X||E|2^{2|X|+|E_c|-1})$.

Proof: The proof is given in the Appendix. ■

VI. Synthesis of Maximally Permissive Supervisors

A. Synthesis Algorithm

We now tackle the synthesis problem for non-prefix-closed specification languages, i.e., non-blockingness must be ensured in addition to safety. Formally, we show how to synthesize a maximal non-blocking supervisor from the NB-AIC.

First, we say that a supervisor S_P is *information-state-based* (IS-based) if

$$(\forall s, t \in \mathcal{L}(S_P/G))\left[IS^Y_{S_P}(s) = IS^Y_{S_P}(t) \Rightarrow S_P(s) = S_P(t)\right]$$

In other words, an IS-based supervisor takes the same control decision every time it visits the same information state. Thus, we can define an IS-based supervisor as $S_I: I \to 2^E$. We define $\mathcal{S}_I(\mathcal{AIC}(G)) \subseteq \mathcal{S}(\mathcal{AIC}(G))$ as the set of IS-based supervisors included in the AIC. Clearly, the cardinality of $\mathcal{S}_I(\mathcal{AIC}(G))$ is finite.

In the prefix-closed case, once the AIC is built, we can randomly pick *one* control decision and fix it at *each* reachable information state and this will give us a (IS-based) supervisor for safety. However, this strategy may not work in the non-prefix-closed case, since the NB-AIC only guarantees that there *exists* a good decision, but arbitrarily choosing one control decision may return a livelock solution. This phenomenon was already pointed out by the example in Fig. 2. Moreover, if we go back to the example in Fig. 2, we find that we cannot remove any (Y or Z) state from the NB-AIC, otherwise, some safe and nonblocking solutions will be excluded. This means that the NB-AIC is already the most "compact" structure that contains all non-blocking solutions, even if it contains some livelock solutions. One conjecture is that we can search through the space of IS-based supervisors, which is finite, for the desired maximal solution. Unfortunately, an IS-based solution does not exist in general; an example where this occurs is presented in Section VII.

The non-existence, in general, of an IS-based supervisor that is both safe and non-blocking implies immediately that state space refinement is required if we want to synthesize a solution from the NB-AIC. Our synthesis algorithm, which is described formally below, is based on the idea of suitably "unfolding" the NB-AIC. To begin with, we need to build an IS-based supervisor (**Step 1**) and then determine whether or not there exists a livelock in it (**Step 2**). If not, then we are done and return the solution. If yes, then we need to break the livelock at some point and resolve it by unfolding the NB-AIC at that point such that a live decision string can be added at the livelock point (**Steps 3 and 4**). This will give us a new (non-IS-based) supervisor. Finally, we need to go back to **Step 2** and test again until the iteration converges (**Step 5**). However, two questions arise: (i) *Where* should we break a livelock? and (ii) *How* can we unfold the NB-AIC? In order to answer these two questions, we first define the concept of "extended BTS" and then we use this notion to define "unfolded" BTS.

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ be the set of non-negative integers. E is called an extended BTS (EBTS) of T if it is a partial unfolding of a BTS T resulting in sets $Q^E_Y = Q^T_Y \times \mathbb{Z}$ and $Q^E_Z = Q^T_Z \times \mathbb{Z}$ with corresponding transition functions $h^E_{YZ}: Q^E_Y \times \Gamma \to Q^E_Z$ and $h^E_{ZY}: Q^E_Z \times E \to Q^E_Y$ over the extended state space, such that the restrictions of h^E_{YZ} and h^E_{ZY} to domains Q^T_Y and Q^T_Z, respectively, are consistent with h^T_{YZ} and h^T_{ZY} whenever h^E_{YZ} and h^E_{ZY} are defined. Specifically, $h^E_{YZ}((y,n),\gamma)$ (respectively, $h^E_{ZY}((z,n),e)$) is of the form $(h^T_{YZ}(y,\gamma), \delta(y,n,\gamma))$ (respectively, $(h^T_{ZY}(z,e), \delta(z,n,e))$), where $\delta: (Q^T_Y \cup Q^T_Z) \times \mathbb{Z} \times (\Gamma \cup E) \to \mathbb{Z}$ is some updating

function for the integer component of the state. (The exact form of δ is left unspecified for the purpose of this general definition.) Given an EBTS E, its included supervisors and its generated languages are defined analogously as before for a BTS in Def. III.5 and Def. III.6, respectively; we will still use the notations $\mathcal{S}(E)$ and $\mathcal{L}_{TS}(E)$ to represent the supervisors included in E and the languages generated by E, respectively. Clearly, if E is a complete EBTS of a complete BTS T, then $\mathcal{S}(E) \subseteq \mathcal{S}(T)$ and $\mathcal{L}_{TS}(E) \subseteq \mathcal{L}_{TS}(T)$.

The definition of an EBTS only requires that the restriction of the transition function to domains Q_Y^T and Q_Z^T be consistent with the BTS. However, we also want that the restriction of the transition function to domain $\mathbb{Z}$ satisfy certain rules (namely, it should "remember" the number of times the current state has been visited). This leads to the notion of an *unfolded* BTS (UBTS), which is a particular type of EBTS defined as follows. For simplicity, we will write state (y, n) as y^n. Given an extended state $x^n \in Q_Y^E \cup Q_Z^E$, $Pre_Y^E(x^n)$ and $Pre_Z^E(x^n)$ denote, respectively, the set of Y-states and the set of Z-states that can reach this state through some runs in E, excluding itself; also, we call x^n a *control state* if $n \in \mathbb{N}$ and a *transient state* if $n \in \mathbb{Z} \setminus \mathbb{N}$.

Definition VI.1: We say that U is an unfolded BTS of a complete BTS T if it is an EBTS of T, such that:

1) $(\forall y^n \in Q_Y^U)[|C_U(y^n)| \leq 1]$;
2) $(\forall z^n \in Q_Z^U)(\forall e \in E)[h_{ZY}(z, e)! \Rightarrow h_{ZY}^U(z^n, e)!]$;
3) There are no cycles in U;
4) For any $y^n \in Q_Y^U$, if $n \in \mathbb{N}$, then $n = |\{y^{\tilde{n}} \in Pre_Y^U(y^n) : \tilde{n} \in \mathbb{N}\}|$. Similarly, for any $z^n \in Q_Z^U$, if $n \in \mathbb{N}$, then $n = |\{z^{\tilde{n}} \in Pre_Z^U(z^n) : \tilde{n} \in \mathbb{N}\}|$.
5) The terminal states of U are either (i) terminal Z-states or (ii) Y-states of the form y^n with $n \geq 1$. ■

For brevity, hereafter, we also write $y^n \xrightarrow{c}_U z^{n'}$ for $h_{YZ}^U(y^n, c) = z^{n'}$ and $z^n \xrightarrow{\sigma}_U y^{n'}$ for $h_{ZY}^U(z^n, \sigma) = y^{n'}$.

Conditions 1) and 2) together imply that except for Y-states with no defined control decision, a UBTS will be complete. Condition 4) says that the integer component of any control state in U is n if there are n control states in its predecessors that have the same Y-or Z-state component. By condition 5), any branch of the UBTS ends up with a repeated control Y-state or a terminal Z-state. Thus, given a UBTS U, we can merge each terminal Y-state $y^n, n \geq 1$ with its predecessor state y^0 and denote the resulting new EBTS by $\tilde{U}$. Specifically, $\tilde{U}$ is obtained by removing states $R := \{y^n \in Q_Y^U : |C_U(y^n)| = 0\}$ from U and for any $y^n \in R$, any transition that originally goes to state y^n in U will go to the corresponding state y^0 in $\tilde{U}$. By definition of a UBTS, $\tilde{U}$ is a complete EBTS. Moreover, we note that the set of supervisors $\mathcal{S}(\tilde{U})$ included in $\tilde{U}$ is a singleton, since there is only one control decision at each Y-state in $\tilde{U}$. Thus, we call the unique supervisor included in $\tilde{U}$ the *supervisor induced by UBTS* U and denote it by S_U. Similarly, for any Y-state $y \in Q_Y^{\tilde{U}}$, we denote by $c_y^{\tilde{U}}$ the unique control decision defined at y, i.e., $C_{\tilde{U}}(y) = \{c_y^{\tilde{U}}\}$. The supervisor S_U can be *realized* by an automaton $A_U = (Q_Y^{\tilde{U}}, E, \xi, q_0, Q_Y^{\tilde{U}})$, where q_0 is the initial Y-state of $\tilde{U}$ and $\xi : Q_Y^{\tilde{U}} \times E \to Q_Y^{\tilde{U}}$ is a partial function defined by: for any $q \in Q_Y^{\tilde{U}}, \sigma \in E$, we have (i) $\xi(q, \sigma) = q$ if $\sigma \in c_y^{\tilde{U}} \cap E_{uo}$; (ii) $\xi(q, \sigma) = h_{ZY}^{\tilde{U}}(h_{YZ}^{\tilde{U}}(y, c_y^{\tilde{U}}), \sigma)$ if $\sigma \in c_y^{\tilde{U}} \cap E_o$; and (iii) $\xi(q, \sigma)$ is undefined if $\sigma \notin c_y^{\tilde{U}}$. Then we can compute the controlled behavior by $\mathcal{L}(S_U/G) = \mathcal{L}(A_U \times G)$, where "$\times$" denotes the usual product composition operation of automata; see, e.g., [2] (p. 78).

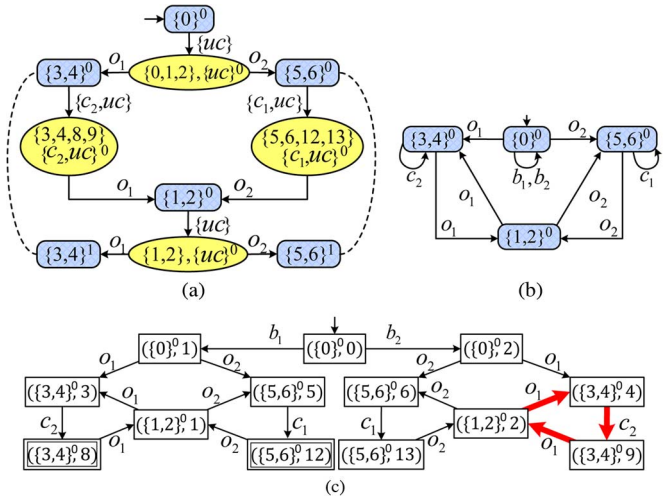

Fig. 3. Example of **Steps 1 and 2**. (a) UBTS U_0 (without the dashed lines). (b) A_{U_0}. (c) $\mathcal{L}(S_{U_0}/G) = \mathcal{L}(A_{U_0} \times G)$.

If $\mathcal{L}(S_U/G)$ is a livelock language, then there exists an elementary livelock cycle $\langle q_1, \sigma_1, \ldots, \sigma_{k-1}, q_k \rangle$ in $A_U \times G$ such that $\exists i \in \{1, \ldots, k-1\} : \sigma_i \in E_o$, since U only contains deadlock-free Z-states. We call such a cycle a *critical elementary livelock cycle* (CELC). In our problem, any CELC in a livelock of $A_U \times G$ corresponds to the presence of some elementary cycle in $\tilde{U}$. Moreover, since a cycle in $\tilde{U}$ is obtained by merging some terminal Y-state y^m and its corresponding y^0 in U, then for a CELC, there exists some terminal Y-state in U that leads to it. We call such a terminal Y-state an *entrance Y-state* of the CELC. More specifically, let $\langle q_1, \sigma_1, \ldots, \sigma_{k-1}, q_k \rangle$ be a CELC. Note that q_i is in the form of $(y_i^{n_i}, x_i)$. Then, there exists an observable event $\sigma_i, i \in \{1, \ldots, k-1\}$ such that $q_{i+1} = (y_{i+1}^0, x_{i+1})$ but $h_{ZY}^U(h_{YZ}^U(y_i^{n_i}, c_{y_i^{n_i}}^{\tilde{U}}), \sigma_i) = y_{i+1}^m, m \neq 0$, where $c_{y_i^{n_i}}^{\tilde{U}}$ is the unique control decision defined at $y_i^{n_i}$ in $\tilde{U}$. In other words, y_{i+1}^m is a terminal Y-state of U, which is not in $\tilde{U}$. Then y_{i+1}^m is an entrance Y-state of the CELC and we call $x_{i+1} \in y_{i+1}$ a corresponding state in the entrance Y-state. In Definition V.1, we introduced the notion of live decision string for a state pair $(y, x), y \in Q_Y^T, x \in y$ in a BTS T. We say that a live decision string $c_1 c_2 \ldots c_n$ is *locally maximal* for (y, x) if there does not exist another live decision string $c_1' c_2' \ldots c_n'$ for (y, x) in T such that $\forall i \in \{1, 2, \ldots, n\} : c_i \subseteq c_i'$ and $\exists j \in \{1, 2, \ldots, n\} : c_j \subset c_j'$.

Example VI.1: Consider the automaton G shown in Fig. 1. An example of UBTS is given in Fig. 3(a); it is an unfolding of $\mathcal{AIC}^{NB}(G)$. By merging state pairs $(\{3,4\}^0, \{3,4\}^1)$ and $(\{5,6\}^0, \{5,6\}^1)$ in U_0 (connected by the dashed lines), we get the corresponding EBTS $\tilde{U}_0$. The induced supervisor S_{U_0} is realized by the automaton A_{U_0} shown in Fig. 3(b). The language of the controlled system $\mathcal{L}(S_{U_0}/G) = \mathcal{L}(A_{U_0} \times G)$ is given in Fig. 3(c). By the properties of the NB-AIC, we know that the language is controllable, observable, safe, and

deadlock-free. However, we see that it is blocking. In $A_{U_0} \times G$, we see that $\langle(\{3,4\}^0,4), c_2, (\{3,4\}^0,9), o_1, (\{1,2\}^0,2), o_1, (\{3,4\}^0,4)\rangle$ is a CELC, which is due to the presence of the cycle $\{3,4\}^0 \to \{1,2\}^0 \to \{3,4\}^0$ in $\tilde{U}_0$ (we omit the Z-states in the cycle since they are uniquely determined). Therefore, $\{3,4\}^1$ is an entrance Y-state of this CELC and $4 \in \{3,4\}$ is a corresponding state in it.

We are now ready to state our synthesis algorithm, which is formally presented in Algorithm NB-SOLU. For the sake of readability, we decompose Algorithm NB-SOLU into five steps that are mapped to the corresponding lines in the statement of the algorithm.

Algorithm 3 $S_{U_k} \leftarrow$ NB-SOLU($\mathcal{AIC}^{NB}(G)$)

1: Set $i \leftarrow 0, Q_Y^{U_i} \leftarrow \{y_0^0\}, M = 0$.
2: EXPAND(U_i)
3: $i \leftarrow i+1, U_i \leftarrow U_{i-1}$
4: **while** $\mathcal{L}(S_{U_{i-1}}/G)$ is a livelock language **do**
5: find an *entrance* state $y_e^k \in Q_Y^{U_i}$ for one CELC and a corresponding state $x_e \in y_e$ that is also in the livelock.
6: Find a locally maximal live decision string $c_1 c_2 \dots c_n$ for (y_e, x_e) in the NB-AIC.
7: From state y_e^k, augment U_i with run $c_1\sigma_1 \dots \sigma_{n-1}c_n$ and the Y and Z-states reachable along its prefixes, where σ_j is defined in Def. V.1. Specifically, we augment U_i with the following transitions:

$$y_e^k \xrightarrow{c_1}_{U_i} z_1^{k_1} \xrightarrow{\sigma_1}_{U_i} y_1^{k_2} \dots \xrightarrow{\sigma_{n-1}}_{U_i} y_{n-1}^{k_{2n-2}} \xrightarrow{c_n}_{U_i} z_n^{k_{2n-1}}$$

where the values of y_j and z_j are determined by h_{ZY} and h_{YZ}, respectively, by the definition of an EBTS and $k_j = M - j$, for any $j = 1, \dots, 2n-1$.
8: $M \leftarrow M - 2n + 1$.
9: EXPAND(U_i)
10: $i \leftarrow i+1, U_i \leftarrow U_{i-1}$
11: **end while**
12: return $S_{U_{i-1}}$
13: **procedure** EXPAND(U)
14: **while** $\exists y^n \in Q_Y^U$ such that $C_U(y^n) = \emptyset \wedge n = 0$
15: **or** $\exists z^n \in Q_Z^U$ such that

$$\exists \sigma \in \Gamma(z) \cap E_o : h_{ZY}(z,\sigma)! \wedge h_{ZY}^U(z^n,\sigma) \text{ is not defined} \tag{8}$$

do
16: **for all** $y^n \in Q_Y^U$ such that $C_U(y^n) = \emptyset \wedge n = 0$ **do**
17: Find a control decision $c \in C_{\mathcal{AIC}^{NB}(G)}(y)$ in $\mathcal{AIC}^{NB}(G)$ such that $\forall c' \in C_{\mathcal{AIC}^{NB}(G)}(y) : c \not\subset c'$
18: Augment U with transition: $y^n \xrightarrow{c}_U z^{n'}$, where $z = h_{YZ}(y,c)$ and

$$n' = \left|\left\{\tilde{z}^{\tilde{n}} \in Pre_Z^U(y^n) : \tilde{z} = z \text{ and } \tilde{n} \geq 0\right\}\right|$$

19: **end for**
20: **for all** $z^n \in Q_Z^U$ such that (8) holds **do**
21: **for all** $\sigma \in \Gamma(z) \cap E_o$ satisfying (8) **do**
22: Augment U with transition: $z^n \xrightarrow{\sigma}_U y^{n'}$, where $y = h_{ZY}(z,\sigma)$ and

$$n' = \left|\left\{\tilde{y}^{\tilde{n}} \in Pre_Y^U(z^n) : \tilde{y} = y \text{ and } \tilde{n} \geq 0\right\}\right|$$

23: **end for**
24: **end for**
25: **end while**
26: **end procedure**

***Step 1**—Generate an Initial UBTS (Lines 1–2):* The goal of this step is to initially generate an IS-based supervisor via building a UBTS from the NB-AIC. First, we set U_0 to be the UBTS that only contains the initial state y_0^0 of the NB-AIC and call procedure EXPAND (lines 13–26). This procedure expands the initial state and constructs a UBTS by a breadth-first search in the NB-AIC. First, pick a locally maximal control decision for y_0^0; then, for the Z-state encountered, find all its Y-state successors and pick one locally maximal control decision for each of them, and so forth, until: (i) a terminal Z-state is reached; or (ii) a Y-state y^n whose information state component has already been visited is reached, i.e., $n \neq 0$. Note that, all the states added by EXPAND are control states, since the integer components are always greater than or equal to zero. Since the construction procedure stops once a Y-state is repeated, the largest index for a Y-state in the UBTS at this step should be 1 and the UBTS induced supervisor is IS-based. Note that the language $\mathcal{L}(S_{U_0}/G)$ is a maximal language, since we take locally maximal control decisions in the construction procedure; however, it may be blocking in general.

***Step 2**—Detect Livelock (Lines 4–5):* The goal of this step is to detect a livelock (if one is present) and find a state where it can be properly broken. If $\mathcal{L}(S_{U_i}/G)$ is livelock-free, then we stop the algorithm and return the current UBTS as the solution. If not, we need to find *one* CELC causing livelock and a corresponding entrance Y-state, as defined earlier.

***Step 3**—Resolve Livelock (Lines 6–7):* This step aims to resolve the livelock found in **Step 2**. Specifically, we unfold the UBTS from an entrance Y-state of the livelock by finding a live decision string in the NB-AIC. The states added at this step are transient states and we use a global variable M in Algorithm NB-SOLU to remember how many transient states we have added to U. Consequently, all the transient states in U have different (negative) integer components. Also, to achieve maximality, all newly added control decisions are locally maximal.

Remark 3: To find such locally maximal live decision strings, one approach is to first find an arbitrary live string and then sequentially replace each control decision in it by a larger one, whenever feasible, from c_1 to c_n. A formal algorithm for this construction is given in the Appendix.

***Step 4**—Complete The UBTS (Line 8):* After **Step 3**, the resulting transition system may no longer be a UBTS. Thus, we need to complete U_i as a UBTS such that we can again induce a supervisor from it. This step is implemented by calling again the procedure EXPAND, which finds one control decision for each Y-state that has no successors, and adds all observations

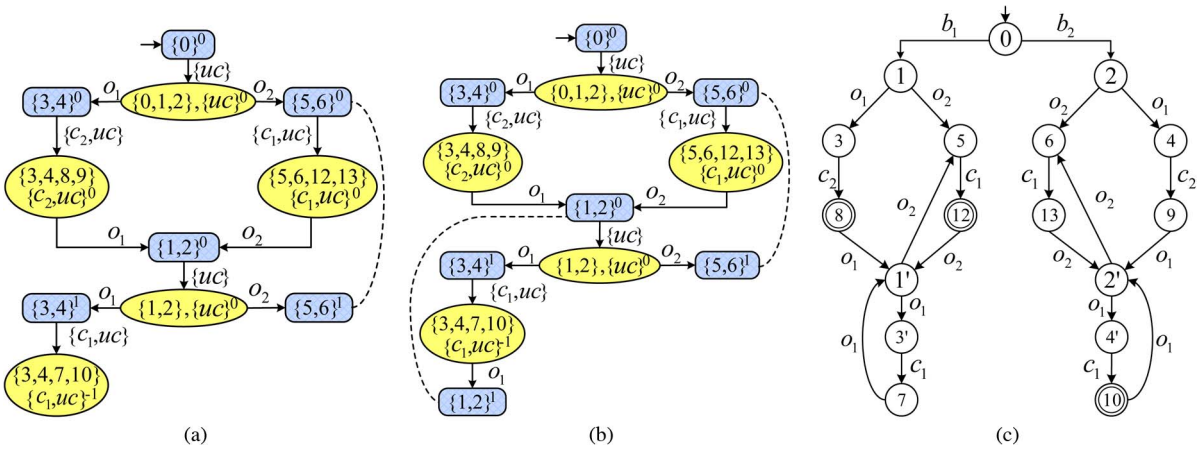

Fig. 4. Example of **Steps 3, 4 and 5**. Note that states in $A_{U_1} \times G$ have been renamed for simplicity. (a) Incomplete UBTS U_1'. (b) UBTS U_1. (c) $\mathcal{L}(S_{U_1}/G) = \mathcal{L}(A_{U_1} \times G)$.

for each Z-state that has some defined observations (i.e., is not terminal).

***Step 5**—Iteration:* Finally, we need to go back to **Step 2** until the iteration stops, i.e., until all livelocks have been resolved.

Example VI.2: Consider the automaton G and its NB-AIC from Fig. 1. Consider the UBTS U_0 and its induced language $\mathcal{L}(S_{U_0}/G)$ shown in Fig. 3. We see that U_0 is a valid UBTS generated after **Step 1**, which ends up with the repeated Y-states $\{3,4\}^1$ and $\{5,6\}^1$, but it induces a livelock solution. Consider the CELC highlighted in Fig. 3 as we have discussed in Example VI.1. In **Step 2**, we find that $y_e = \{3,4\}^1$ is an entrance Y-state of this livelock and return $(\{3,4\}^1, 4)$. For **Step 3**, one possible choice is to take control decision $\{c_1, uc\}$ at $\{3,4\}^1$, since state 4 will be able to reach marked state 10 via c_1. Therefore, a transient Z-state $(\{3,4,7,10\}, \{c_1, uc\})^{-1}$ is added and the resulting BTS U_1' is shown in Fig. 4(a). However, in U_1', the enabled observable event o_1 is not defined at Z-state $(\{3,4,7,10\}, \{c_1, uc\})^{-1}$. Thus, **Step 4** will call procedure EXPAND again to complete the UBTS by adding a new Y-state $\{1,2\}^1$ that can be reached by observing o_1 into U_1'. Since $\{1,2\}$ already exists in the UBTS, we stop the procedure EXPAND and get U_1 shown in Fig. 4(b) and its induced language $\mathcal{L}(S_{U_1}/G)$ is shown Fig. 4(c). Since $\mathcal{L}(S_{U_1}/G)$ is livelock-free, we stop the synthesis procedure and return it as a maximal controllable, observable, safe, and non-blocking solution.

Remark 4: In Fig. 3(a), we could also select control decision $\{c_2, uc\}$ at state $\{5,6\}^0$. It can be easily verified that this will induce a non-blocking and IS-based solution. Thus we can stop the synthesis at **Step 2** and return this solution. However, as discussed earlier, the above situation may not always hold. This is why we chose the non-IS-based solution to illustrate all the steps of Algorithm NB-SOLU.

Remark 5: In the prefix-closed case, since all the states are marked, only **Step 1** is required in Algorithm NB-SOLU. Note that S_{U_0} is an IS-based supervisor. Therefore, there always exists at least one *IS-based* supervisor S_I such that $\mathcal{L}(S_I/G)$ is a maximal controllable, observable and safe language. Moreover, if $E_c \subseteq E_o$, then this IS-based supervisor will generate the unique supremal controllable and observable sublanguage. Once the AIC is built, such an IS-based supervisor can be obtained by a simple breadth-first search on the AIC, which takes time of $O(|E_o|2^{|X|+|E_c|})$ in the worst case.

B. Correctness of the Synthesis Algorithm

In this section, we show that (i) the synthesis algorithm presented in the previous section converges in a finite number of iterations and (ii) the resulting solution is maximal.

In the synthesis steps of Algorithm NB-SOLU, the supervisor should not only know its current information state, but it also needs to remember the number of times the current state has been visited. However, this does not tell us how much memory we need to realize the supervisor. The following theorem reveals that the supervisor can be represented in a finite structure, i.e., the resulting language is regular.

Theorem VI.1: Algorithm NB-SOLU converges in a finite number of iterations.

Proof: Suppose that $x \in y^n$ is detected in $A_{U_i} \times G$ at **Step 2**, where x is a corresponding state of an entrance Y-state y^n, $n \geq 1$ and $i \geq 1$. This implies that there exists a CELC $\langle (y^0, x), \sigma_1, \ldots, \sigma_k, (y^0, x) \rangle$ in $A_{U_i} \times G$. We define the pair being *resolved* for the CELC as the last state in the CELC before the final state (y^0, x) such that (i) its first component is y^0; and (ii) it is entered by an observable event. More specifically, we can write this CELC in the form of

$$\begin{aligned}
&(y^0, x) \xrightarrow{\sigma_1^1 \sigma_2^1 \ldots \sigma_{k_1}^1} (y^0, x^2) \xrightarrow{\sigma_1^2 \sigma_2^2 \ldots \sigma_{k_2}^2} (y^0, x^3) \\
&\cdots \xrightarrow{\sigma_1^j \sigma_2^j \ldots \sigma_{k_j}^j} (y^0, x^{j+1}) \xrightarrow{\sigma_1^{j+1} \sigma_2^{j+1} \ldots \sigma_{k_{j+1}}^{j+1}} \cdots \\
&\xrightarrow{\sigma_1^{r-1} \sigma_2^{r-1} \ldots \sigma_{k_{r-1}}^{r-1}} (y^0, x^r) \xrightarrow{\sigma_1^r \sigma_2^r \ldots \sigma_{k_r}^r} (y^0, x)
\end{aligned} \tag{9}$$

where $\sigma_{k_j}^j \in E_o, j = 1, \ldots, r$ and (y^0, x^r) is the pair being *resolved*. Note that, inside of the CELC, cycle $\langle y^0, \ldots, y^{n-1}, y^0 \rangle$ in A_{U_i} may be involved for r times. Fig. 5 illustrates the notions of detected and resolved states in the context of the CELC.

In order to prove the theorem, it suffices to prove that any pair $(y^0, x), x \in y$ can be resolved at most once in **Step 3**.

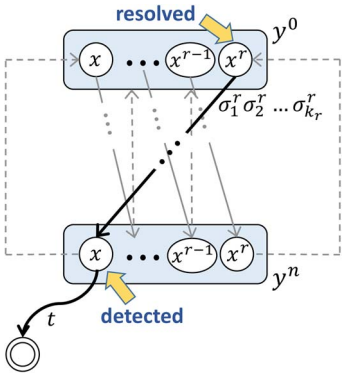

Fig. 5. Conceptual illustration of the proof of Theorem VI.1.

To see this, let us first suppose that $x \in y^n$ was detected in $A_{U_i} \times G$ at **Step 2**, where $n \geq 1$ and $i \geq 1$. Let $(y^0, x^r), x^r \in y$ be the corresponding pair being resolved. Note that x^r and x need not necessarily be the same state. Since **Step 3** is executed after detecting $x \in y^n$, the above CELC will be broken and a path from $x \in y^n$ to a marked state is introduced. Formally, in $A_{U_{i+1}} \times G$, we know that $f_{[A_{U_{i+1}} \times G]}((y^0, x^r), \sigma_1^r\sigma_2^r \ldots \sigma_{k_r}^r) = (y^n, x)$ and there exists a string $t \in E^*$ such that $f_{[A_{U_{i+1}} \times G]}((y^0, x^r), \sigma_1^r\sigma_2^r \ldots \sigma_{k_r}^r t) \in Q_Y^{U_{i+1}} \times X_m$, where $f_{[A_{U_{i+1}} \times G]}$ denotes the transition function of $A_{U_{i+1}} \times G$. This path is also illustrated in Fig. 5. In fact, t can be the corresponding live path for the live decision string introduced at y^n as defined in Def. V.1.

Now, let us assume that after some iteration steps, $x^r \in y^0$ is resolved again for a CELC in $A_{U_{i+q}} \times G, q \geq 2$. This means that (y^0, x^r) is in a livelock of $A_{U_{i+q}} \times G$. Moreover, we have already shown that there exists a string $\sigma_1^r\sigma_2^r \ldots \sigma_{k_r}^r t$ from (y^0, x^r) to a marked state in $A_{U_{i+1}} \times G$. Such a marked state is also reachable from (y^0, x^r) in $A_{U_{i+q}} \times G$, since U_{i+q} is unfolded from U_{i+1} and any states reachable in U_{i+1} are still reachable in U_{i+q}. More specifically, such a marked state can be reached from (y^0, x^r) via the same string $\sigma_1^r\sigma_2^r \ldots \sigma_{k_r}^r t$ as above. Therefore, (y^0, x^r) cannot be in any livelock, which gives us a contradiction. Thus we conclude that any pair $(y, x), x \in y$ can be resolved at most once in **Step 3**.

Let the set of Y-states of $\mathcal{AIC}^{NB}(G)$ be denoted by Q_Y^{NBG}. Then, $\sum_{y \in Q_Y^{NBG}} |y| \leq \sum_{i=1}^{|X|} i\binom{|X|}{i} = |X|2^{|X|-1}$ gives an upper bound for the number of iterations.

Proposition VI.1: If the NB-AIC has been constructed, then the running time of Algorithm NB-SOLU is $O([|X|^3 2^{|X|} + |E|]|X|2^{2|X|+|E|})$.

Proof: The proof is given in the Appendix. ■

Suppose that Algorithm NB-SOLU stops after n steps of iteration and returns UBTS U_n; then the induced supervisor S_{U_n} has the following properties.

Theorem VI.2: $\mathcal{L}(S_{U_n}/G)$ is a controllable, observable, safe, and non-blocking language.

Proof: Follows directly from Theorem V.1 and the livelock-free stopping condition in **Step 2**. ■

Theorem VI.3: $\mathcal{L}(S_{U_n}/G)$ is maximal, i.e.

$$\left(\forall S' \in \mathcal{S}\left(\mathcal{AIC}^{NB}(G)\right)\right)\left[\mathcal{L}(S_{U_n}/G) \not\subset \mathcal{L}(S'/G)\right].$$

Proof: We prove this theorem by contradiction. Assume that $\mathcal{L}(S_{U_n}/G)$ is not maximal, i.e., $\exists S' \in \mathcal{S}(\mathcal{AIC}^{NB}(G))$ such that $\mathcal{L}(S_{U_n}/G) \subset \mathcal{L}(S'/G)$. This implies the following two facts[1]:

1) $(\forall s \in \mathcal{L}(S_{U_n}/G))[S_{U_n}(s) \subseteq S'(s)]$;
2) $(\exists t \in \mathcal{L}(S_{U_n}/G))[S_{U_n}(t) \subset S'(t)]$.

Let us consider the string $t \in \mathcal{L}(S_{U_n}/G)$ such that $S_{U_n}(t) \subset S'(t)$ and $S_{U_n}(t') = S'(t'), \forall t' \in \overline{\{t\}} \setminus \{t\}$. Then we know that $IS^Y_{S_{U_n}}(t) = IS^Y_{S'}(t)$, and we call this Y-state y. Then, for the control decision at y in S_{U_n}, i.e., $S_{U_n}(t)$, one of the two following cases holds:

(i) $S_{U_n}(t)$ is a control decision returned by **Step 1 or 4**. By the construction rule, we know that $\forall c' \in C_{\mathcal{AIC}^{NB}(G)}(y)$: $S_{U_n}(t) \not\subset c'$. Since $S' \in \mathcal{S}(\mathcal{AIC}^{NB}(G))$, by Def. III.5, we know that $S_{U_n}(t) \subset S'(t)$ cannot happen.

(ii) $S_{U_n}(t)$ is a control decision returned by **Step 3**. Suppose that $S_{U_n}(t)$ is in a live control decision string $c_1c_2 \ldots c_n$ and let $w := \xi_1\sigma_1\xi_2 \ldots \xi_{i-1}\sigma_{i-1}\xi_n$ be the corresponding live path as defined in Def. V.1. We assume, without loss of generality, that $S_{U_n}(t) = c_1, S_{U_n}(t\xi_1\sigma_1) = c_2, \ldots, S_{U_n}(t\xi_1\sigma_1\xi_2 \ldots \sigma_{n-1}) = c_n$. Consider another live control decision string $c_1'c_2' \ldots c_n'$, where $c_i' := S'(t\xi_1\sigma_1\xi_2 \ldots \xi_{i-1}\sigma_{i-1}), 1 \leq i \leq n$. Such a live control decision string is well defined since $\mathcal{L}(S_{U_n}/G) \subset \mathcal{L}(S'/G)$ and tw is also in $\mathcal{L}(S'/G)$. By fact 2) above we know that $c_i \subseteq c_i', i \geq 2$. Moreover, we know that $c_1 \subset c_1'$. Thus, $c_1'c_2' \ldots c_n'$ is strictly lager than $c_1c_2 \ldots c_n$, which contradicts the fact that $c_1c_2 \ldots c_n$ is locally maximal.

For each case, we obtain a contradiction. Thus, no more permissive supervisor exists. ■

Remark 6: The intuition behind the above proof is that it is impossible to construct a supervisor that generates a language strictly larger than the one obtained by the proposed algorithm, since we have taken either locally maximal control decisions [case (i)] or locally maximal control decision strings [case (ii)]. For the first case, it is easy to see that the control decision $S_{U_n}(t)$ is locally maximal. For the second case, it does not mean that we cannot find a *single* control decision c_1' such that $c_1 \subset c_1'$. However, if we do so, then $c_1'c_2 \ldots c_n$ will not be a live decision string. The intuition behind this phenomenon is that, in partially-observed DES, enabling more events at the current state may result in more conservative decisions in the future. In other words, the control decision string $c_1c_2 \ldots c_n$ is locally maximal *as a whole*.

Recall that the NB-AIC exists if there exists a non-empty solution to the problem under consideration and Algorithm NB-SOLU always returns a maximal solution in a finite number of iterations if the NB-AIC exists. Consequently, we have the following theorem.

Theorem VI.4: BSCOP-NB$^{\max}$ is solvable if and only if $\mathcal{AIC}^{NB}(G)$ exists.

Hence, the existence of the NB-AIC provides the solvability condition for BSCOP-NB$^{\max}$. This extends the results in [9] and [14], that can only be applied to BSCOP-NB.

[1]Without loss of generality, we assume that the supervisors are irredundant.

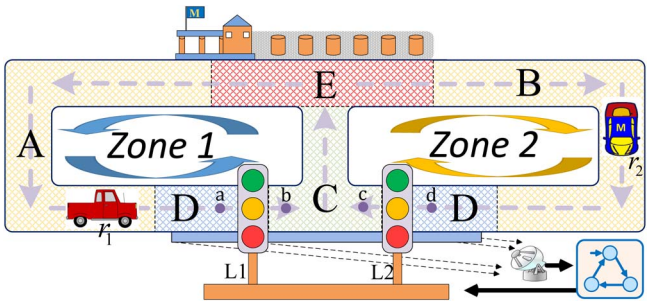

Fig. 6. Example discussed in Section VII.

C. Discussion and Computational Complexity

We have solved the maximally permissive supervisor synthesis problem for both prefix-closed and non-prefix-closed specification languages. It was shown in [1] that when the plant can be fully observed, under the assumption that $H \sqsubseteq G$, the maximal permissive supervisor can be repressed in the form of $S : X \to \Gamma$. Analogously, for the partially-observed prefix-closed specification case, since the information state we defined captures all the information we need to solve the problem, the supervisor we synthesized is in the form of $S_P : I \to \Gamma$. For the non-prefix-closed specification case, we have shown that the information state is not sufficient anymore to carry all the information we need for synthesis purposes; in this case, the "real" information state is the information state originally defined augmented with an integer that represents the number of times the current state has been visited. Thus, the maximally permissive supervisor is in fact in the form of $S_P : I \times \mathbb{Z} \to \Gamma$.

We have shown in Proposition VI.1 that the worst-case time complexity of the synthesis algorithm is exponential in both $|X|$ and $|E|$. One may ask that whether we can get rid of such high computational complexity. However, it was shown in [22] that there is no polynomial algorithm to synthesize a partial observation supervisor, even if it is known *a priori* that such a supervisor exists. Therefore, the exponential complexity we obtained in the synthesis of maximally permissive safe and non-blocking supervisors seems to be unavoidable. Such computational complexity is due to the partially observed nature of the system, and the only way to overcome this is to put more sensors in the plant. But the motivation for this paper is that in many cases the designer has no such option and partial observability is the problem one must tackle.

VII. Illustrative Example

We illustrate the synthesis algorithm of Section VI-A, Algortihm NB-SOLU, by an illustrative example. In particular, this example shows that: (i) IS-based supervisors that are both safe and non-blocking may not always exist in general; and (ii) a maximal solution can still be obtained by using Algorithm NB-SOLU, even when the algorithms in [7] and [8] return empty solutions.

System Model: Consider the following guideway problem: A town is divided into two zones, zone 1 and zone 2, with single-way streets as shown in Fig. 6. At the top of the zones, there is a recycling station. Everyday, *only one* zone will send a robot (r_1 or r_2) to clean up the streets. The robot sent by zone 1 can only move counter-clockwise, i.e., move forward or turn left; the robot sent by zone 2 can only move clockwise.

Control: There are two traffic lights, L_1 and L_2, close to the bottom intersection as shown in the figure. The lights control the robots as follows: When L_1 is red, if robot r_1 is at point a, then it must wait until the light turns green; if robot r_2 is at point c, then it can choose to wait there or turn right. The effect of L_2 is analogous.

Sensing: There is a radar around the traffic lights that detects whether there is a robot in region D, which is in front of each light, every time unit. However, the radar cannot distinguish which zone the detected robot belongs to.

Specification: Since all streets are one-way streets, with legal directions shown in Fig. 6, we do not want movement in the reverse direction to happen. Without any traffic light, the robot from zone 1 could possibly violate this specification by entering zone 2 through the points a, b, c and d. Clearly, if both L_1 and L_2 are kept red, then the above specification can be satisfied trivially. However, in order for the robot to be able to unload the trash it collected along the streets, we require that the robot should always be able to enter region E. In summary, the goal for us is to design a control policy for the traffic lights for one day's operation based on the radar information and such that the above requirements are satisfied.

The above problem can be modeled as a supervisory control problem under partial observation. First, we use unobservable and uncontrollable events a_1 and a_2 to represent the nondeterministic initial setting, since we do not know where the robot starts from. Event o is used to model the event that the radar detects a robot in region D, which is observable but not controllable. We use event c_1 to represent that there is a robot that crosses L_1 (from the RHS to the LHS or from the LHS to the RHS); this event is controllable but not observable. We define c_2 analogously for the control effect of L_2. Events b_1 and b_2 represent that robots r_1 and r_2 unload their trash, respectively; these events are unobservable and uncontrollable. The automaton model G of this system is shown in Fig. 7(a), in which states 9 and 10 are illegal states.

The corresponding NB-AIC for G is shown in Fig. 7(b). By applying Algorithm NB-SOLU, we first obtain the initial UBTS U_0 shown in Fig. 7(c), which induces a livelock solution. Thus, we need to unfold from the entrance Y-state {3,4}, which results in the UBTS U_1 shown in Fig. 7(d). UBTS U_1 induces the controllable, observable, safe, and non-blocking sublanguage $\mathcal{L}(\tilde{U}_1/G)$ shown in Fig. 7(e). Moreover, this language is maximal.

This example, while simple, has important implications. First, note that the solution obtained by Algorithm NB-SOLU is a non-IS-based solution, since it enables c_1 when state $\{3, 4\}$ is visited for $2k + 1$ times and it enables c_2 when state $\{3, 4\}$ is visited for $2k$ times, $k \in \mathbb{N}$. Moreover, we see that any fixed control decision at Y-state {3,4} will result in a livelock solution. This verifies our earlier assertion in Section VI that IS-based solutions may not exist in general and that the unfolding steps of Algorithm NB-SOLU are indeed needed. Second, for this problem, the supremal controllable normal solution and the solutions obtained by using the methods in [7], [8]

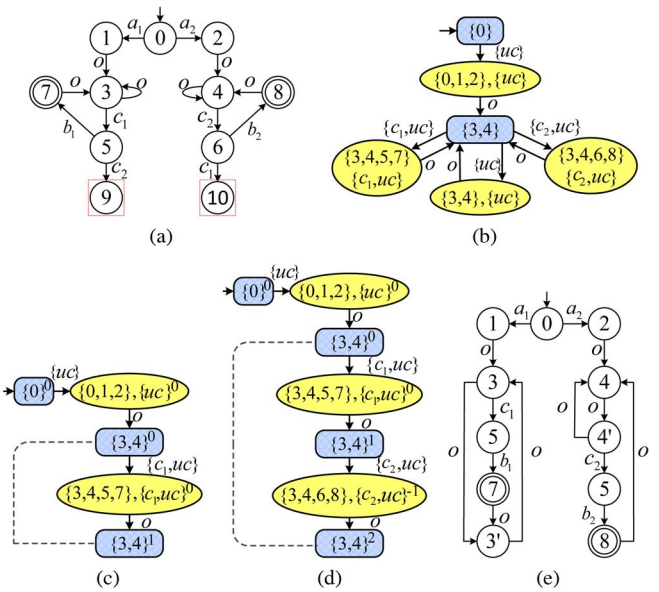

Fig. 7. For G: $E_c = \{c_1, c_2\}$, $E_o = \{o\}$, and states 9 and 10 are illegal. (a) Automaton G. (b) The corresponding NB-AIC. (c) UBTS U_0. (d) UBTS U_1. (e) $\mathcal{L}(S_{U_1}/G)$.

are all empty, even though a solution exists. In general, the solution obtained by the approach proposed in this paper is incomparable with those obtained by using the methods in [7], [8], even though they may not be maximal. How to synthesize a maximal solution that contains a particular language is the topic of ongoing research.

VIII. Conclusion

We solved the previously open problem of synthesizing a controllable, observable, and locally maximal sublanguage of a given non-prefix-closed language. This results in a supervisor that is safe, non-blocking, and maximally permissive for a partially observed DES. For this purpose, we first defined the All Inclusive Controller, a bipartite transition system whose structure contains all the safe solutions. We then defined the Non-Blocking All Inclusive Controller, another new bipartite transition system obtained from the AIC that takes non-blockingness into account in addition to safety. We provided a synthesis algorithm that uses the NB-AIC to synthesize the desired maximal, controllable, and observable sublanguage. Finally, the convergence and maximality of this algorithm were proved. In the future, we will investigate: (i) extending the NB-AIC to decentralized systems; and (ii) finding an "optimal" solution with respect to some cost criterion.

Appendix

A. Implementation of the Algorithms

In this section, we discuss implementation issues related to the construction and synthesis algorithms in the paper. Specifically, we answer the following two questions.

1) How to verify the liveness property defined in Def. V.1?
2) How to find a local maximal live decision string $c_1 c_2 \dots c_n$ for any state pair $(y, x), y \in Q_Y^T, x \in y$ in a BTS T?

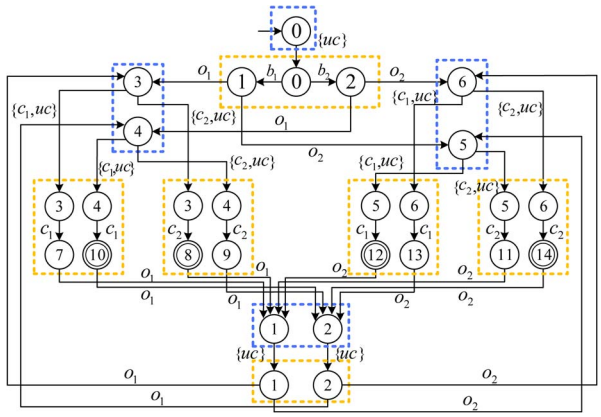

Fig. 8. Example of Inter-Connected System: The figure shows the corresponding ICS for the automaton and its NB-AIC shown in Fig. 1. The blue dashed rectangles and yellow dashed rectangles correspond to the Y-states and the Z-states in the BTS, respectively.

The key to these two problems is to build an automaton that contains all state connections *inside* of each Y-or Z-state in the BTS. We call such an automaton the *Inter-Connected System* (ICS) (see Fig. 8).

Definition A.1. (Inter-Connected System): Given a bipartite transition system T (w.r.t. G), its corresponding *Inter-Connected System* is defined as the automaton $ICS^T = (Q^{ICS^T}, \Sigma^{ICS^T}, \delta^{ICS^T}, q_0^{ICS^T}, Q_m^{ICS^T})$, where

- $Q^{ICS^T} \subseteq (Q_Y^T \times X) \cup (Q_Z^T \times X)$ is the set of states defined by
  - $(y, x) \in Q^{ICS^T}$ if $y \in Q_T^Y$ and $x \in y$
  - $(z, x) \in Q^{ICS^T}$ if $z \in Q_T^Z$ and $x \in I(z)$;
- $\Sigma^{ICS^T} = E \cup \Gamma$ is the set of events;
- $\delta^{ICS^T} : Q^{ICS^T} \times \Sigma \to Q^{ICS^T}$ is the partial transition function defined by: for any $\gamma \in \Gamma, \sigma \in E$
  - $\delta^{ICS^T}((y, x_1), \gamma) = (z, x_2)$ if $x_1 = x_2$ and $h_{YZ}^T(y, \gamma) = z$
  - $\delta^{ICS^T}((z, x_1), \sigma) = (z, x_2)$ if $f(x_1, \sigma) = x_2$ and $\sigma \in \Gamma(z) \cap E_{uo}$
  - $\delta^{ICS^T}((z, x_1), \sigma) = (y, x_2)$ if $f(x_1, \sigma) = x_2$, $\sigma \in \Gamma(z) \cap E_o$ and $h_{ZY}^T(z, \sigma) = y$
- $q_0^{ICS^T} = (\{x_0\}, x_0)$ is the initial state;
- $Q_m^{ICS^T} = \{(z, x) \in Q^{ICS^T} : x \in X_m\}$ is the set of marked states. ■

The ICS for an EBTS U is defined analogously.

Given an automaton, we say a state is *co-accessible* if there is a string from this state to a marked state and we say an automaton is *co-accessible* if all states in it are co-accessible; see, e.g., [2]. The following result says that to verify the liveness a Y-state in T it suffices to verify the co-accessibility of its corresponding states in ICS^T.

Proposition A.1: Given a BTS T and its Inter-Connected System ICS^T, a Y-state y in T is live if and only if for any state $x \in y$ in it, $(y, x) \in Q^{ICS^T}$ is co-accessible in ICS^T.

Proof: ($\Rightarrow$) Since the Y-state y is live in T, then Definition V.1 implies that for any state $x \in y$ in it, there exists a decision string $c_1c_2\ldots c_n$ such that under this decision string there exists a string $s = \xi_1\sigma_1\xi_2\ldots\sigma_{n-1}\xi_n, \xi_i \in (E_{uo} \cap c_i)^*, \sigma_i \in E_o \cap c_i$, such that $f(x,s) \in X_m$. By the definition of the ICS, such a string $w = c_i\xi_1\sigma_1c_2\ldots\sigma_{n-1}c_n\xi_n$ also exists in ICS^T and $\delta^{ICS^T}((y,x),w) \in Q_m^{ICS^T}$. Thus, (y,x) is co-accessible.

($\Leftarrow$) By construction. Recall that $\Sigma^{ICS^T} = E_o \cup E_{uo} \cup \Gamma$. Then we first define two natural projections $P_C : (E_o \cup E_{uo} \cup \Gamma)^* \to \Gamma^*$ and $P_{CO} : (E_o \cup E_{uo} \cup \Gamma)^* \to (E_o \cup \Gamma)^*$, i.e., for any $s \in (\Sigma^{ICS^T})^*$, $P_C(s)$ is of the form $c_1c_2c_3\ldots, c_i \in \Gamma$ and $P_{CO}(s)$ is of the form $c_1\sigma_1c_2\sigma_2\ldots, c_i \in \Gamma, \sigma_i \in E_o$.

Since for any $x \in y$, (y,x) is co-accessible in ICS^T, we can find a string $t = e_1e_2\ldots e_m \in (\Sigma^{ICS^T})^*$ such that $\delta^{ICS^T}((y,x),t) \in Q_m^{ICS^T}$. By Definition V.1, it is clear that $P_C(t)$ is a live decision string for (y,x). Consequently, y is live in T. ■

Corollary A.1.: Given a BTS T, all Y-states in T are live if and only ICS^T is co-accessible.

In the construction algorithm of the NB-AIC, we need to check whether or not there exists a Y-state in a BTS that is not live. By Corollary A.1, this suffices to check the co-accessibility of the ICS^T. Specifically, we need to first build $ICS^{\mathcal{AIC}(G)}$, the ICS of the AIC; then, for each iteration step, we check whether or not $ICS^{\mathcal{AIC}(G)}$ is co-accessible. If a state $(y,x) \in Q^{ICS^{\mathcal{AIC}(G)}}$ is not co-accessible, then (i) the Y-state y in $\mathcal{AIC}(G)$ should be removed and; (ii) the set of states $\{(y',x') \in Q^{ICS^{\mathcal{AIC}(G)}} : y' = y\}$ should also be removed from the ICS; we then repeat until the ICS is accessible.

Now we are ready to show how to find a locally maximal live decision string $c_1c_2\ldots c_n$ for (y,x), as needed in Algorithm NB-SOLU. In the proof of Proposition A.1, we have already shown how to find a live decision string for a given (y,x). For computation simplicity, we can find a shortest live path s in ICS^T such that $\delta^{ICS^T}((y,x),s) \in Q_m^{ICS^T}$ and get the shortest live decision string $c_1c_2\ldots c_n = P_C(s)$ and its corresponding run $c_1\sigma_1c_2\sigma_2\ldots\sigma_{n-1}c_n = P_{CO}(s)$. To find a locally maximal decision string, our approach is simply to sequentially replace each single control decision in $P_C(s)$ by one that is as large as possible. Specifically, we start from c_1 and see whether or not we can pick a control decision c_1' in $C_{\mathcal{AIC}^{NB}(G)}(y)$ such that: (i) $c_1 \subset c_1'$ and (ii) $c_1'c_2\ldots c_n$ is also a live decision string. If c_1' satisfies these two conditions, then we replace c_1 by c_1', and try to grow c_1', and so forth, until we cannot find a larger one. Then we proceed to analyze $c_2, c_3, \ldots$ by the same manner. The only difference is that when we try to replace c_i by c_i', we just need to consider the existence of the decision string $c_i'c_{i+1}\ldots c_n$ and do not need to consider those that have already been grown to be maximal (namely, c_1 to c_{i-1}). This procedure is formally described by Algorithm L-MAX.

Algorithm 4 $c_1c_2\ldots c_n \leftarrow$ L-MAX$(y, c_1\sigma_1c_2\sigma_2\ldots\sigma_{n-1}c_n)$

1: $i \leftarrow 1, y_1 \leftarrow y$
2: **while** $i \le n$ **do**
3: **for all** $c' \in C_{\mathcal{AIC}^{NB}(G)}(y_i)$ **do**
4: **if** $c_i \subset c'$ and the run $c'\sigma_ic_{i+1}\sigma_{i+1}\ldots\sigma_{n-1}c_n$ is defined at y_i in the NB-AIC **then**
5: $c_i \leftarrow c'$
6: **end if**
7: **end for**
8: $y_{i+1} \leftarrow h_{ZY}(h_{YZ}(y_i, c_i), \sigma_i)$
9: $i \leftarrow i + 1$
10: **end while**

B. Complexity Analysis

Proof of Proposition V.1.:

Proof: First, we need to build $\mathcal{AIC}(G)$, which can be done in $O(|X||E|2^{|X|+|E_c|})$ as discussed earlier. For each Z-state in $\mathcal{AIC}(G)$, checking whether it is deadlock-free can be done in $O(|X||E|)$. Thus, line 2 in the algorithm can be done in $O(|X||E|2^{|X|+|E_c|})$. As discussed in Appendix A, before starting the iteration, we need to build $ICS^{\mathcal{AIC}(G)}$, which has $\sum_{y\in Q_Y^{AICG}}|y| + \sum_{z\in Q_z^{AICG}}|I(z)| \le (1+2^{|E_c|})\sum_{i=1}^{|X|} i\binom{|X|}{i} = (1+2^{|E_c|})|X|2^{|X|-1}$ number of states and $2^{|E_c|}\sum_{y\in Q_Y^{AICG}}|y| + |E|\sum_{z\in Q_z^{AICG}}|I(z)| \le (2^{|E_c|}+2^{|E_c|}|E|)|X|2^{|X|-1}$ number of transitions; we denote these upper bounds by n_1 and n_2, respectively. Thus, the construction of $ICS^{\mathcal{AIC}(G)}$ can be done in $O(|X||E|2^{|X|+|E_c|-1})$.

Since we need to remove at least one Y-state for each iteration step, the whole iteration procedure will execute at most $|Q_Y^{AICG}|$ number of times, which is bounded by $2^{|X|}$. For each iteration step, by Corollary A.1, we need to verify the co-accessability of $ICS^{\mathcal{AIC}(G)}$, which can be done in $O(n_1+n_2)$; then we search through the state space of Q_Y^{AICG} and remove the Y-states that have no successors and the corresponding states in the ICS, which is still bounded by $O(n_1+n_2)$. Thus, the total complexity for the construction of the NB-AIC is $O(|X||E|2^{2|X|+|E_c|-1})$. ■

Proof of Proposition VI.1:

Proof: First, in Algorithm 3, the EBTS $\tilde{U}_i$ contains at most $|X|^2 2^{2|X|}$ Y-states and the same number of Z-states. Therefore, in the ICS $ICS^{\tilde{U}_i}$, there are at most $n_1' := |X|^3 2^{2|X|+1}$ states and $n_2' := |X|^3 2^{2|X|+|E|+1}$ transitions. The above n_1' and n_2' are estimated based on the fact that the largest superscript of any control Y-state y is $|y|$ and for each iteration we introduce at most $|X|2^{|X|}$ transient Y-states. Now we are ready to analyze the complexity of the synthesis algorithm.

First, let us consider the complexity of each single iteration step (**Step 2–4**):

- **Step 2** is a livelock detection problem in the ICS of $\tilde{U}_i$, which can be done in $O(n_1' + n_2')$.
- **Step 3** involves two problems:
  1) a shortest path search problem in the ICS of $\tilde{U}_i$, which requires $O(n_1' + n_2')$ and
  2) the problem of growing this path to be maximal. For this problem, since such path has a length $N = |X|2^{|X|+1}$, in the worst case, then it requires a complexity of $O(N2^{|E_c|} + (N-2)2^{|E_c|} + \cdots + 2^{|E_c|}) = O(((N+1)^2/4)2^{|E_c|})$.

- **Step 4** requires calling the procedure EXPAND, which can be done in $O(|E_o|2^{|X|+|E_c|} + |E_o||X|2^{|X|})$.

Thus, the complexity of a single iteration step is of $O([|X|^3 2^{|X|} + |E|]2^{|X|+|E|})$.

In the convergence proof of Algorithm NB-SOLU, we have already shown that $|X|2^{|X|-1}$ provides an upper bound for the number of iterations. Combining this with the above results, we get that the total complexity of Algorithm NB-SOLU is $O([|X|^3 2^{|X|} + |E|]|X|2^{2|X|+|E|})$. ■

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for useful comments on improving this paper and pointing out a mistake in an earlier version of the proof of Theorem VI.1.

REFERENCES

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Cont. Opt.*, vol. 25, no. 1, pp. 206–230, 1987.
[2] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems,* 2nd ed. New York: Springer, 2008.
[3] F. Lin and W. Wonham, "On observability of discrete-event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 173–198, 1988.
[4] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Trans. Autom. Control*, vol. 33, no. 3, pp. 249–260, 1988.
[5] H. Cho and S. Marcus, "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation," *Math. Control Sig. Syst.*, vol. 2, no. 1, pp. 47–69, 1989.
[6] R. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *Syst. Control Lett.*, vol. 15, no. 2, pp. 111–117, 1990.
[7] S. Takai and T. Ushio, "Effective computation of an $L_m(G)$-closed, controllable, observable sublanguage arising in supervisory control," *Syst. Control Lett.*, vol. 49, no. 3, pp. 191–200, 2003.
[8] K. Cai, R. Zhang, and W. Wonham, "Relative observability of discrete-event systems and its supremal sublanguages," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 659–670, 2015.
[9] K. Inan, "Nondeterministic supervision under partial observations," in *Proc. 11th Int. Conf. Anal. Optim. Syst.: Discrete Event Syst.*, 1994, pp. 39–48.
[10] R. Kumar, S. Jiang, C. Zhou, and W. Qiu, "Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control," *IEEE Trans. Autom. Control*, vol. 50, no. 4, pp. 463–475, 2005.
[11] A. Arnold, A. Vincent, and I. Walukiewicz, "Games for synthesis of controllers with partial observation," *Theor. Comp. Sci.*, vol. 303, no. 1, pp. 7–34, 2003.
[12] K. Chatterjee, L. Doyen, T. Henzinger, and J.-F. Raskin, "Algorithms for omega-regular games with imperfect information," in *Computer Science Logic*. New York: Springer, 2006, pp. 287–302.
[13] J. Komenda and J. van Schuppen, "Control of discrete-event systems with partial observations using coalgebra and coinduction," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 15, no. 3, pp. 257–315, 2005.
[14] T.-S. Yoo and S. Lafortune, "Solvability of centralized supervisory control under partial observation," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 16, no. 4, pp. 527–553, 2006.
[15] J. Fa, X. Yang, and Y. Zheng, "Formulas for a class of controllable and observable sublanguages larger than the supremal controllable and normal sublanguage," *Sys. Control Lett.*, vol. 20, no. 1, pp. 11–18, 1993.
[16] M. Heymann and F. Lin, "On-line control of partially observed discrete event systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 4, no. 3, pp. 221–236, 1994.
[17] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 6, no. 4, pp. 379–427, 1996.
[18] X. Yin and S. Lafortune, "A general approach for synthesis of supervisors for partially-observed discrete-event systems," in *Proc. 19th IFAC World Congress*, 2014, pp. 2422–2428.
[19] X. Yin and S. Lafortune, "Synthesis of maximally permissive nonblocking supervisors for partially observed discrete event systems," in *Proc. 53rd IEEE Conf. Decision Control*, 2014, pp. 5156–5162.
[20] X. Yin and S. Lafortune, "Supplement material for the paper 'Synthesis of maximally permissive supervisors for partially-observed discrete-event systems'." [Online]. Available: http://www-personal.umich.edu/xiangyin/TACsup.pdf
[21] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin, "Variable lookahead supervisory control with state information," *IEEE Trans. Autom. Control*, vol. 39, no. 12, pp. 2398–2410, 1994.
[22] J. Tsitsiklis, "On the control of discrete-event dynamical systems," *Math. Control, Signals Syst.*, vol. 2, no. 2, pp. 95–107, 1989.

Xiang Yin (S'14) was born in Anhui, China, in 1991. He received the B.Eng degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2012 and the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, in 2013, where he is currently pursuing the Ph.D degree.

His research interests include supervisory control of discrete-event systems, model-based fault diagnosis, formal methods, game theory and their applications to cyber and cyber-physical systems.

Stéphane Lafortune (F'99) received the B.Eng degree from Ecole Polytechnique de Montréal, Montréal, QC, Canada, in 1980, the M.Eng degree from McGill University, Montréal, in 1982, and the Ph.D degree from the University of California at Berkeley, in 1986, all in electrical engineering.

Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of electrical engineering and computer science. He is the Lead Developer of the software package UMDES and co-developer of DESUMA. He co-authored he textbook *Introduction to Discrete Event Systems—Second Edition* (Springer, 2008). He is Editor-in-Chief of the *Journal of Discrete Event Dynamic Systems: Theory and Applications*. His research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems.

Dr. Lafortune received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the IEEE Control Systems Society, in 1994 and 2001, respectively.