53rd IEEE Conference on Decision and Control
December 15-17, 2014. Los Angeles, California, USA

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

Xiang Yin and Stéphane Lafortune

***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 and non-blockingness must be ensured in addition to safety. Our approach is based on the construction of 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 then provide a synthesis algorithm, based on the NB-AIC, that constructs a supervisor that is safe, non-blocking and maximally permissive. This is the first algorithm with such properties.**

I. INTRODUCTION

We consider the control of partially observed Discrete Event Systems (DES). The goal is to restrict the behavior of a DES within a non-prefix-closed legal language, while accounting for the presence of uncontrollable and unobservable events. The non-prefix-closed specification requires us to take both safety and non-blockingness into account.

Supervisory control of centralized and partially observed DES was initially studied in [1], [2], 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. If a language cannot be exactly achieved, then the synthesis problem asks whether we can synthesize a supervisor S_P 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), where G is the plant and $\mathcal{L}_m(H)$ is the non-prefix-closed specification language, which is assumed to be a sublanguage of $\mathcal{L}_m(G)$. This synthesis problem was shown to be decidable in [3] and solvable in [4]. However, since observability may not be preserved under union, no supremal solution exists in general. Hence, 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 still safe and non-blocking; in other words, such solutions are *locally maximal*.

Many approaches have been considered in the literature for synthesizing safe and non-blocking supervisors for partially observed DES; see, e.g., [5], [6], [3], [7], [8], [9]. One approach is to find the supremal controllable *normal* and $\mathcal{L}_m(G)$-closed sublanguage of $\mathcal{L}_m(H)$[1], [2]. In [5], [6], solutions which are provably larger than the supremal controllable normal sublanguage are provided. Another approach is to use nondeterministic supervisors; this was first advocated in [3] and subsequently extended in [7]. In [8], [9], a game-theoretic approach was considered for the synthesis of supervisors. However, each of the above approaches has its own limitations. The solutions in [5], [6] may be too restrictive and may not always exist in general. In [4], the authors provide an algorithm that returns a solution to the problem under consideration; however, maximal permissiveness of that solution is not guaranteed. To the best of our knowledge, the synthesis of non-blocking and safe supervisors that are maximally permissive for partially observed DES remains an open problem.

In our recent work in [10], we defined a new finite bipartite transition system, called the AIC for "All Inclusive Controller". The AIC embeds in its structure all safe control decisions, using suitably defined information states. In this paper, we build on our previous work in [10] and consider non-blockingness in addition to safety, with the goal of synthesizing solutions that are maximally permissive. Our approach is based on the construction of a new finite state transition structure that we call the "Non-Blocking All Inclusive Controller (or NB-AIC herafter). The NB-AIC contains in its transition structure all supervisors that are safe and deadlock-free. Moreover, the NB-AIC can serve as the basis for the synthesis of supervisors that will provably be non-blocking and maximally permissive (while remaining safe). Recall that a supervisor is non-blocking if it is both deadlock-free and livelock-free.

The main contributions of this paper are: (i) The definition of the NB-AIC, which contains all solutions to partial observation supervisory control problems with non-prefix-closed specifications; (ii) The construction algorithm for the NB-AIC; (iii) A new algorithm based on the NB-AIC that returns a solution that is non-blocking and maximally permissive.

The remainder of this paper is organized as follows. Section II describes the model of the system we analyze. In Section III, we formally define a class of bipartite transition systems and summarize our previous work for prefix-closed specifications in [10]. In Section IV, we start from the AIC and define the NB-AIC, the key structure of interest in this paper. Section V provides a algorithm that uses the NB-AIC to synthesize a maximal safe and non-blocking solution. The properties of the synthesis algorithm are established in Section VI. Finally, we conclude the paper in Section VII. Due to space constraints, all proofs have been omitted and they are available in [11].

This work was partially supported by the NSF Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering (grant CCF-1138860).

Xiang Yin and Stéphane Lafortune are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA. {xiangyin,stephane}@umich.edu.

II. SYSTEM MODEL

We assume basic knowledge of DES and common notations (see, e.g., [12]). 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 of the system is described by the prefix-closed language $\mathcal{L}(G)$ generated by G.

In the supervisory control framework [13], a non-prefix-closed language $K \subseteq \mathcal{L}_m(G)$ represents the desired (safe and non-blocking) behavior. A supervisor is imposed on G to achieve the 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. Under the partial observation assumption [1], E is also partitioned into the subset of observable events, E_o, and the subset of unobservable events, E_{uo}. A partial observation supervisor is a function $S_P : P(\mathcal{L}(G)) \to 2^E$, with the following constraint: $E_{uc} \subseteq S_P(s), \forall s \in E_o^*$, where $P : E^* \to E_o^*$ is the natural projection defined in the usual manner (see, e.g., [12]). We say that a control decision is admissible if it satisfies the above constraint and we define $\Gamma = \{\gamma \in 2^E : E_{uc} \subseteq \gamma\}$ as the set of admissible control decisions. 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 in the usual manner.

Let $K = \mathcal{L}_m(H)$. Hereafter, we assume, w.l.o.g., that H and G satisfies the following properties: (i) H is a sub-automaton of G (as defined in [12]); (ii) if $x, y \in X_H$ and $f_G(x, \sigma) = y$ 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). If the original G and H do not satisfy these assumptions, the algorithm in the appendix of [14] can be used to refine both of them and ensure that (i) and (ii) hold. Thus, we can talk of the legality of *states* of X rather than of *strings* of $\mathcal{L}(G)$.

We define two 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, $\text{UR}_\gamma(S) := \{x \in X : (\exists u \in S)(\exists e \in (E_{uo} \cap \gamma)^*)$ s.t. $x = f(u, e)\}$. The *Observable Transition* of the subset of states $S \subseteq X$ under observable event $e \in E_o$ is given by, $\text{Next}_e(S) := \{x \in X : \exists u \in S$ s.t. $x = f(u, e)\}$. Finally, we assume the reader is familiar with the standard DES-theoretic properties of *controllability* and *observability*, which are used in this paper with respect to G, E_{uc}, and E_o, as defined in [12], pages 147 and 178.

III. BIPARTITE TRANSITION SYSTEM

A. Bipartite transition system

We start by defining the general notion of *bipartite transition system* (BTS). Let an information state (IS) be a subset $IS \subseteq X$ of states and denote by $I = 2^X$ the set of all information states.

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

$$T = (Q_Y^T, Q_Z^T, h_{YZ}^T, h_{ZY}^T, E, \Gamma, y_0^T) \tag{1}$$

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: $h_{YZ}^T(y, \gamma) = z$ only if
  - $I(z) = \text{UR}_\gamma(y)$ and $\Gamma(z) = \gamma$
- $h_{ZY}^T : Q_Z^T \times E \to Q_Y^T$ is the partial transition function from Z-states to Y-states, which satisfies the following constraint: $h_{ZY}^T(z, e) = y$ only if
  - $e \in \Gamma(z) \cap E_o$ and $y = \text{Next}_e(I(z))$
- 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, the 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 states augmented with admissible control decisions, 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 a Z-state to Y-state represents the observable transition. This means that y is the set of states reachable from some state of the information state component of the preceding Z-state through the single enabled observable event. We call a sequence in the form of $c_1\sigma_1 \ldots c_n\sigma_n, c_i \in \Gamma, \sigma_i \in E_o, \forall i$ a *run* in T.

The definition of a BTS is based on the plant G. For simplicity, we will omit "with respect to G" in the remainder, if the plant G is clear. Since the control decision for a Y-state may not be unique, given a BTS T, we define $C_T(y) := \{\gamma \in \Gamma : h_{YZ}^T(y, \gamma)!\}$, where ! means is defined, to be the set of control decisions defined at $y \in Q_Y^T$.

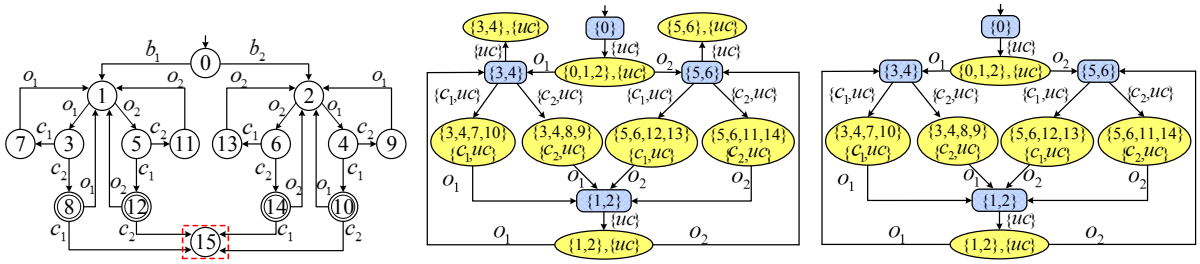
B. Total controller and BTS included supervisor

We discuss the connection between BTS and supervisors in this section. First, we define an important type of BTS called the Total Controller (TC), which enumerates all possible behaviors between the controller and the plant.

Definition 2: (Total controller). The total controller for G is defined as the BTS

$$\mathcal{TC}(G) = (Q_Y^{TCG}, Q_Z^{TCG}, h_{YZ}^{TCG}, h_{ZY}^{TCG}, E, \Gamma, y_0^{TCG})$$

where: (i) h_{YZ}^{TCG} contains all admissible transitions from Y-states to Z-states (i.e., all admissible control decisions at the



(a) Automaton G (b) $\mathcal{AIC}(G)$ (c) $\mathcal{AIC}^{NB}(G)$

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

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). ■

Since $\mathcal{TC}(G)$ contains all admissible control decisions after each observation and 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.

There is a special kind of Z-state in the TC, which has no successors. We say 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, the only deadlock states in the TC are terminal Z-states.

Definition 3: Given a supervisor S_P, $IS^Y_{S_P}(y,s)$ 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 $IS^Y_{S_P}(y_0,s)$ as $IS^Y_{S_P}(s)$.

Also, $IS^Z_{S_P}(z,s)$ is defined analogously, with $IS^Z_{S_P}(s) := IS^Z_{S_P}(z_0,s)$, where $z_0 = h^{TCG}_{YZ}(y_0, S_P(\epsilon))$. ■

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

Definition 4: 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.

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

Definition 5: Given a complete BTS T, a supervisor S_P is said to be included in T if

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

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

Definition 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. ■

C. All Inclusive Controller for Safety

In this section, we briefly review the transition structure called the All Inclusive Controller (AIC), which was used to solve the supervisory control problem in the case of prefix-closed specifications in our previous work [10].

Given an information state $i \in I$, the safety binary function $D_I : I \to \{0,1\}$ is defined by $D_I(i) = 1$ if $\forall x \in i : x \in X_H$ and $D_I(i) = 0$ otherwise. We say that a Y-state is safe if it is currently safe and there *exist* control decisions that maintain the safety property for all possible future behaviors. Since we cannot choose event occurrences, we say that a Z-state is safe if *all* of its successor Y-states are safe. We therefore define two safety binary functions, $D_Y : Y \to \{0,1\}$ and $D_Z : Z \to \{0,1\}$ as follows:

$$D_Y(y) = \begin{cases} 1, & \text{if } D_I(y) = 1 \text{ and} \\ & \exists \gamma \in \Gamma : D_Z(h^{TCG}_{YZ}(y,\gamma)) = 1 \\ 0, & \text{else} \end{cases} \quad (2)$$

$$D_Z(z) = \begin{cases} 1, & \text{if } D_I(I(z)) = 1 \text{ and } \forall e \in \\ & \Gamma(z) \cap E_o : D_Y(h^{TCG}_{ZY}(z,e)) = 1 \\ 0, & \text{else} \end{cases} \quad (3)$$

We say that a control decision γ is safe from Y-state y if $D_Z(h^{TCG}_{YZ}(y,\gamma)) = 1$, since we know that there exists a sequence of safe control decisions in the future. In [10], we showed that the partially observed safety control problem can be mapped to the problem of finding a subsystem of the total controller in which all reachable states are safe.

Definition 7: (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 safe subsystem of $\mathcal{TC}(G)$ consisting of only *safe* reachable Y and Z-states, and the transitions between them in $\mathcal{TC}(G)$. ■

The following theorem show that the AIC (only) contains valid solutions to the safety control problem.

Theorem 1: [10] If $\mathcal{AIC}(G)$ is non-empty, then

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

Example 3.1: Let G be the automaton shown in Figure 1(a). Its corresponding AIC is shown in Figure 1(b). In the

diagram of the AIC, rectangular states correspond to Y-states and oval states correspond to Z-states. For more details about the construction of the AIC, the reader is referred to [10].

Remark 3.1: In Figure 1(b), at the initial Y-state $y_0 = \{0\}$, we can also take control decision $\{c_1, uc\}$. However, event 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, $(\forall e \in \gamma)(\exists x \in \text{UR}_\gamma(i))[f(x,e)!]$. Hereafter, we only keep irredundant control decisions in the AIC; this will not affect its properties.

IV. NON-BLOCKING ALL INCLUSIVE CONTROLLER

In this section, we first define and then present an algorithm to construct the Non-Blocking AIC (NB-AIC), a bipartite transition system obtained from the AIC that contains all safe and non-blocking control policies.

A. Definition of the NB-AIC

Definition 8: (Live decision string). Given a BTS T, for any Y-state y and state $x \in y$ in it, we say a decision string $c_1 c_2 \dots c_n, c_i \in \Gamma$ is *live* for (x, y) 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 $c_{i+1} \in C_T(y_i), \forall i < n$, 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. We say a Y-state y is *live* if for all $x \in y$, (x, y) has a live decision string. ■

Example 4.1: Consider the automaton and its corresponding AIC shown in Figure 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.

Remark 4.1: 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.

The purpose of the notion of liveness is to eliminate one source of blocking: if a Y-state is not live, then no matter what control decision we take, we will always be blocked by some state in it. For a Z-state z, we also need to 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 9: (Deadlock-free Z-state). A Z-state z is said to be *deadlock-free* if for all $x \in I(z)$ we have

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

Otherwise, it is said to be a *deadlcok* state. ■

We are now ready to define a new binary function that captures the blockingness of a Y or Z-state.

Definition 10: (Non-blocking binary function for Y and Z state). For a complete BTS T, we define two binary functions, $B_Y : Y \to \{0, 1\}$ and $B_Z : Z \to \{0, 1\}$ by

$$B_Y(y;T) = \begin{cases} 1, & \text{if } y \text{ is live in } T \\ 0, & \text{else} \end{cases}$$

$$B_Z(z;T) = \begin{cases} 1, & \text{if } z \text{ is deadlock-free and} \\ & \forall e \in \Gamma(z)! \cap E_o : B_Y(h^T_{ZY}(z,e);T) = 1 \\ 0, & \text{else} \end{cases}$$

We will show later that the non-blocking binary function eliminates all the Y- and Z- states that may lead to blocking. Unlike the safety binary function, we do not need the existential quantifier for Y-states, since liveness is a stronger property. But for a Z-state, the universal quantifier is still needed to capture the recursiveness in the definition. Note that the non-blocking binary function depends on the transition system T, i.e., the same system state in different Ts may have different function values, which is not the case for the safety binary function.

Definition 11: (Non-blocking all inclusive controller). The Non-Blocking All Inclusive Controller for G is the largest non-blocking subsystem of $\mathcal{AIC}(G)$. By non-blocking subsystem, we mean that $B_Y(y) = 1$ and $B_Z(z) = 1$ for all Y-states y and for all Z-states z, respectively. We denoted the NB-AIC of G by $\mathcal{AIC}^{NB}(G)$. ■

Note that, in the definition, the largest non-blocking subsystem of the AIC is uniquely defined, since the union of any two non-blocking subsystems is still non-blocking.

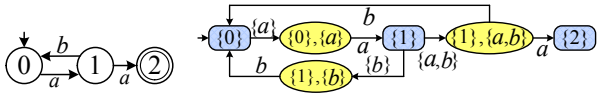
Example 4.2: Going back to Figure 1, the NB-AIC for G is shown in Figure 1(c). Comparing with its AIC, since all Y-states in it are live, the deadlock Z-states that are removed are $(\{3, 4\}, \{uc\})$ and $(\{5, 6\}, \{uc\})$.

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. We use the following terminology for a prefix-closed language L: (i) L is *deadlock-free* if every terminating string in L ends at a marked state in G; (ii) L is *non-blocking* if $\overline{L} \cap \mathcal{L}_m(G) = L$; and (iii) L is safe if $L \subseteq \overline{K}$.

Theorem 2: 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 controllable, observable, safe, and non-blocking, then $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$.

In general, for $L \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, L need not be livelock-free. Consider the automaton G in Fig. 2(a) and its corresponding NB-AIC shown in Fig. 2(b). Clearly, $(ab)^* \in \mathcal{L}_{TS}(\mathcal{AIC}^{NB}(G))$, but it is a livelock language.



(a) Automaton G (b) The corresponding NB-AIC

Fig. 2. For G: $E_{uo} = \emptyset$ and $E_{uc} = b$.

B. Construction of the NB-AIC

In [10], an algorithm is provided for the construction of the AIC structure. Thus we assume that the AIC has already

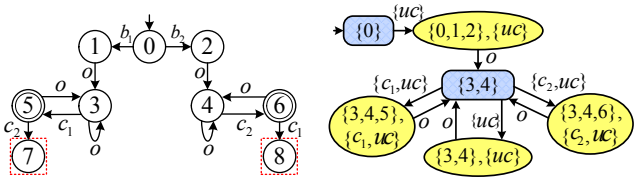
Algorithm 1 $\mathcal{AIC}^{NB}(G) \leftarrow$FIND-NB-AIC($\mathcal{AIC}(G)$)

1: $A \leftarrow \mathcal{AIC}(G)$
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: $\mathcal{AIC}^{NB}(G) \leftarrow$ Accessible(A)

been built and it serves as the basis for the construction of the NB-AIC. 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 we need to prune: (i) a Z-state that is deadlock; (ii) a Y-state that is not live; and (iii) a Y or Z-state that violates completeness (Def. 4). In the algorithm, the elimination of (i), (ii) and (iii) are implemented at lines 2, 4, and 6, respectively. Note that for (ii) and (iii), iteration steps are required, since pruning states may change the liveness or the completeness of the system. However, (i) just needs to be executed once, since the deadlock property does not depend on T.

V. SYNTHESIS ALGORITHM

In this section, we first discuss the difficulty that arises in solving the non-blocking control problem and our approach to overcome it. Then we formally show how to synthesize a maximal non-blocking supervisor from the NB-AIC.



(a) Automaton G (b) The corresponding NB-AIC

Fig. 3. $E_c = \{c_1, c_2\}, E_o = \{o\}$ and state 7 and 8 are illegal.

In the prefix-closed specification case, once the AIC is built, we can randomly pick *one* control decision at *each* information state and this will give us a valid supervisor for safety. However, this strategy may not work in the non-prefix-closed specification case, since the NB-AIC only guarantees that there *exists* a good decision, but arbitrary choosing one control decision may return a livelock solution. This phenomenon was already pointed out in Fig. 2. One conjecture is that we can search through the space of information state based (IS-based) supervisors, which is finite, for the desired maximal solution. However, the next example shows that an IS-based solution does not exist in general.

Example 5.1: Consider the automaton G and it corresponding NB-AIC shown in Figure 3. We see that any fixed control decision at Y-state $\{3,4\}$ will provides a livelock solution. One possible non-blocking control policy is to enable c_1 when we visit $\{3,4\}$ for $2k+1$ times and to enable c_2 when we visit $\{3,4\}$ for $2k$ times, $k \in \mathbb{N}$. However, this is not an IS-based supervisor.

The non-existence of an IS-based supervisor implies that state space refinement is required if we want to synthesize a solution from the NB-AIC. Our synthesis algorithm, is based on the idea of unfolding a BTS. To begin with, we need to build a 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 the it by unfolding the NB-AIC at that point such that a live decision string can be added at the livelock point (**Step 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? The answers to these two questions are obtained by building the *unfolded* BTS (UBTS) defined below.

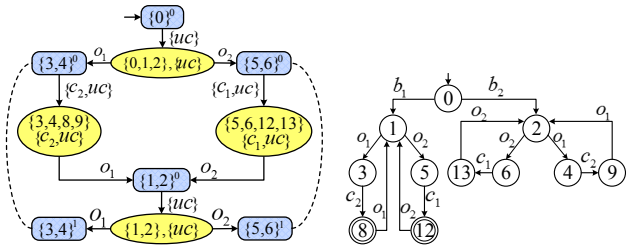
Definition 12: U is an unfolded BTS of a BTS T if it is a finite partial unfolding of T resulting in sets $Q_Y^U = Q_Y^T \times \mathbb{N}$ and $Q_Z^U = Q_Z^T \times \mathbb{N}$ with corresponding transition functions $h_{YZ}^U : Q_Y^U \times \Gamma \to Q_Z^U$ and $h_{ZY}^U : Q_Z^U \times E \to Q_Y^U$ over the extended state space, and such that the following conditions are satisfied:

1) The restrictions of h_{YZ}^U and h_{ZY}^U to domain Q_Y^T and Q_Z^T, respectively, are consistent with h_{YZ}^T and h_{ZY}^T;
2) The restriction of h_{YZ}^U or h_{ZY}^U to domain $\mathbb{N}$ is defined by: the integer component of any state in U is n if there are n states in its predecessors (states that can reach this state from the initial state) that have the same Y- or Z-state component;
3) $(\forall y \in Q_Y^U)[|C_U(y)| \leq 1]$;
4) $(\forall z \in Q_Z^U)(\forall e \in E)[h_{ZY}^{TCG}(z,e)! \Rightarrow h_{ZY}^U(z,e)!]$;
5) There is no cycle in U;
6) 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 simplicity, we write a state (y,n) in the form of y^n. ■

We call U a partial (finite) unfolding because of conditions 1) and 3). By condition 6), any branch of the UBTS ends up with a repeated Y-state or a terminal Z-state. Thus, given a UBTS U, we can merge the terminal Y-state y^n with its predecessor state y^0 and denote the resulting new transition system by $\tilde{U}$, which is a complete (unfolded) BTS. Moreover, we note that the set of supervisors $\mathcal{S}(\tilde{U})$ included in $\tilde{U}$ is singleton, since there is only one control decision at each Y-state in $\tilde{U}$. Thus, we call the unique supervisor included in $\tilde{U}$, *supervisor induced by UBTS* U, and denote it by S_U.

Example 5.2: Consider the automaton G shown in Figure 1. An example of UBTS is given in Figure 4(a). By merging state pair $(\{3,4\}^0, \{3,4\}^1)$ and $(\{5,6\}^0, \{5,6\}^1)$ in U_0 (connected by the dashed line), we can get the corresponding $\tilde{U}_0$. The resulting language $\mathcal{L}(S_{U_0}/G)$ is given in Figure 4(b). By the properties of the NB-AIC, we know that this language is controllable, observable, safe, and deadlock-free. However, we see in the figure that it is not livelock-free.



(a) UBTS U_0 (without the dash lines) (b) $\mathcal{L}(S_{U_0}/G)$

Fig. 4. Example of Steps 1 and 2

Now, we are ready to state our synthesis algorithm.

Step 1: Generate an initial UBTS: The goal of this step is to initially generate an IS-based supervisor via building a UBTS, and it is described formally by Algorithm INITIAL. In order to make the UBTS induced supervisor IS-based, we need to stop once a Y-state is repeated. Thus, the largest index for a Y-state in the UBTS at this step should be 1. The language $\mathcal{L}(S_{U_0}/G)$ is a maximal language, since we take locally maximal control decisions in our construction; however, it may be blocking in general.

Step 1 $U_0 \leftarrow$INITIAL($\mathcal{AIC}^{NB}(G)$)

1: Set $i \leftarrow 1$.
2: Generate a UBTS U_0 as follows: starting from y_0, for each reachable Y-state y pick *one* control $c \in C_{\mathcal{AIC}^{NB}(G)}(y)$ in $\mathcal{AIC}^{NB}(G)$ such that $\forall c' \in C_{\mathcal{AIC}^{NB}(G)}(y) : c \not\subset c'$ and for each reachable Z-state pick *all* observations, until: (i) a terminal Z-state is reached; or (ii) a Y-state that has already been visited is reached.
3: Label each state with a non-negative integer as defined earlier.

Step 2: Detect livelocks: The goal of this step is to detect livelock (if it exists) and find a state where it can be properly broken. First, we observe that any elementary cycle in a livelock of $\mathcal{L}(S_U/G)$ corresponds to the presence of an elementary cycle in $\tilde{U}$. Moreover, since the cycle in $\tilde{U}$ is obtained by merging some terminal Y-state y^m and its corresponding y^0 in U, then for each livelock, there exists a terminal state that contributes to the cycle that leads to the livelock. We call such terminal state the *entrance* of the livelock. For clarity, these concepts are illustrated in Example 5.3. Formal procedures for this step are described in Algorithm DETECT.

Step 2 $(x_e, y_e) \leftarrow$DETECT($U_{i-1}, \mathcal{AIC}^{NB}(G)$)

1: $U_i \leftarrow U_{i-1}$
2: Compute $\mathcal{L}(S_{U_i}/G)$
3: **if** there is no livelock state in $\mathcal{L}(S_{U_i}/G)$ **then**
4: stop and return S_{U_i} as the supervisor
5: **else**
6: find an *entrance* state $y_e \in Q_Y^{U_i}$ for one livelock and a state $x_e \in y_e$ that is also in the livelock.
7: **end if**

Example 5.3: Consider G and its NB-AIC shown in Figure 1. Recall that the UBTS U_0 shown in Figure 4(a) is a valid UBTS returned by Algortihm INITIAL, which induces a livelock language $\mathcal{L}(S_{U_0}/G)$. Consider the livelock $2 \to 4 \to 9 \to 2$, which is due to the presence of the the cycle $\{3,4\} \to \{1,2\} \to \{3,4\}$ in $\tilde{U}_0$ (we omit the Z-states in the cycle since they are uniquely determined). Then we find that $y_e = \{3,4\}^1$ is an entrance of this livelock and return $(4, \{3,4\}^1)$.

Remark 5.1: In Figure 4(a), we can also take 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. In the remainder, we will continue to use the non-IS-based initial setting shown in Example 5.3 as our illustrative example.

Step 3: Resolve livelocks: The goal of the step is to resolve the livelock found in **Step 2**. Specifically, we unfold the UBTS from the entrance state by finding a live decision string in the NB-AIC. Also, to achieve maximality, we want the new added control decisions to be locally maximal. This step is summarized by Algorithm RESOLVE.

Step 3 $U_i \leftarrow$RESOLVE($U_i, (x_e, y_e), \mathcal{AIC}^{NB}(G)$)

1: Find a live control string $c_1 c_2 \dots c_n$ for (x_e, y_e) in the NB-AIC with the property that there does not exist a live decision string $c_1' c_2' \dots c_n'$ such that there exists $I \subseteq \{1, 2, \dots, n\}$ where $c_i \subset c_i'$ for all $i \in I$ and $c_j = c_j'$ for all $j \notin I$.
2: From state y_e, 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_i is defined in Def. 8.
3: Label the new added states with integers.

Remark 5.2: 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.

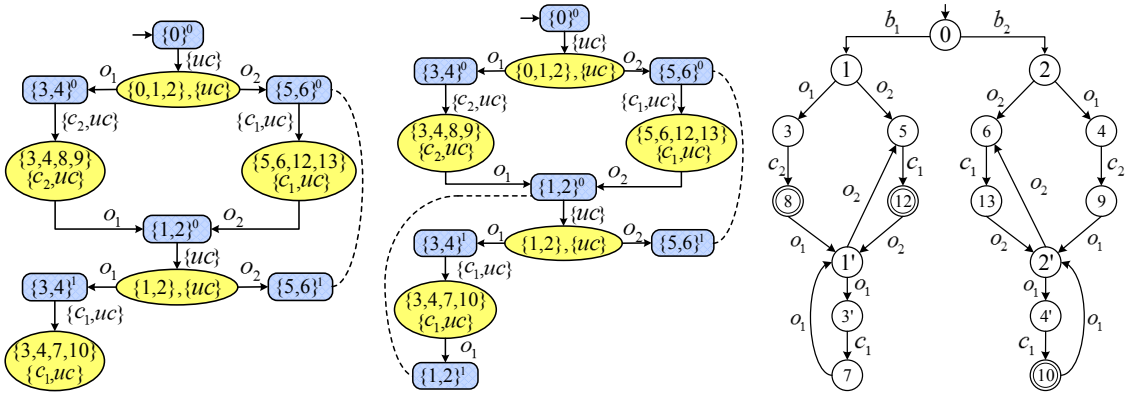
Example 5.4: In the last example, we detected $(4, \{3,4\}^1)$ as the point at which a live decision should be added. One possible choice is to take control decision $\{c_1, uc\}$ at $\{3,4\}^1$, since state 4 will be able to reach marked state 10 via c_1. The resulting BTS U_1' is shown in Figure 5(a).

Step 4: Complete the UBTS: After **Step 3**, the resulting transition system may no longer be a UBTS. Thus, the aim of this step is to complete it as a UBTS such that we can again induce a supervisor from it. This step is given by Algorithm COMPLETE.

Example 5.5: In U_1', event o_1 is enabled but not defined at Z-state $(\{3,4,7,10\}, \{c_1, uc\})$. By observing o_1, a new Y-state $\{1,2\}^1$ will be reached. Since $\{1,2\}$ already exists in the UBTS, we stop and return U_1 shown in Figure 5(b).

Step 5: Iteration: $i \leftarrow i+1$ and go to **Step 2**.

Example 5.6: The U_1 induced language $\mathcal{L}(S_{U_1}/G)$ is shown in Figure 5(c). We see that it is livelock-free. Thus, we stop the synthesis procedure and return $\mathcal{L}(S_{U_1}/G)$, a controllable, observable, safe, and non-blocking solution that



(a) Incomplete UBTS U_1' (b) UBTS U_1 (c) $\mathcal{L}(S_{U_1}/G)$

Fig. 5. Example of Steps 3, 4 and 5

Step 4 $U_i \leftarrow$COMPLETE$(U_i, \mathcal{AIC}^{NB}(G))$

1: For each added Z-state in **Step 3**, complete its observation transitions and, for each reachable Y-state y pick *one* control $c \in C_{\mathcal{AIC}^{NB}(G)}(y)$ in $\mathcal{AIC}^{NB}(G)$ such that $\forall c' \in C_{\mathcal{AIC}^{NB}(G)}(y) : c \not\subset c'$ and, for each reachable Z-state, pick *all* observations, until: (i) a terminal Z-state is reached; or (ii) a Y-state that has already been visited is reached.
2: Augment U_i with these states and transitions.
3: Label the newly added states with integers.

is also maximally permissive (as proved in the next section).

VI. PROPERTIES OF THE ALGORITHM

In this section, we show that (i) the synthesis algorithm presented in Sec. V converges in a finite number of steps and (ii) the resulting solution is maximal. In our synthesis steps, 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. This does not tell us how much space 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 3: The synthesis algorithm converges in a finite number of iterations.

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

Theorem 4: $\mathcal{L}(S_{U_n}/G)$ is a controllable, observable, safe, and non-blocking sub-language.

Theorem 5: $\mathcal{L}(S_{U_n}/G)$ is maximal, i.e.,
$(\forall S' \in \mathcal{S}(\mathcal{AIC}^{NB}(G)))[\mathcal{L}(S_{U_n}/G) \not\subset \mathcal{L}(S'/G)]$.

VII. 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 defined the Non-Blocking All Inclusive Controller, a bipartite transition system whose structure contains all the solutions to the problem. We provided a synthesis algorithm which uses the NB-AIC to synthesize the desired maximal, controllable, and observable sublanguage. In the future, we will investigate: (i) extending the NB-AIC to decentralized systems; and (ii) finding an "optimal" solution w.r.t. some cost criterion.

REFERENCES

[1] F. Lin and W. Wonham, "On observability of discrete-event systems," *Inform. Sciences*, vol. 44, no. 3, pp. 173–198, 1988.
[2] 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.
[3] K. Inan, "Nondeterministic supervision under partial observations," in *11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems*. Springer, 1994, pp. 39–48.
[4] 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.
[5] S. Takai and T. Ushio, "Effective computation of an $l_m(g)$-closed, controllable, and observable sublanguage arising in supervisory control," *Systems & Control Letters*, vol. 49, no. 3, pp. 191–200, 2003.
[6] K. Cai, R. Zhang, and W. M. Wonham, "On relative observability of discrete-event systems," in *Decision and Control, 52th IEEE Conference on*, 2013, pp. 7285–7290.
[7] 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.
[8] A. Arnold, A. Vincent, and I. Walukiewicz, "Games for synthesis of controllers with partial observation," *Theoretical Computer Science*, vol. 303, no. 1, pp. 7–34, 2003.
[9] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin, "Algorithms for omega-regular games with imperfect information," in *Computer Science Logic*. Springer, 2006, pp. 287–302.
[10] X. Yin and S. Lafortune, "A general approach for synthesis of supervisors for partially-observed discrete-event systems," in *19th IFAC World Congress*, 2014, pp. 2422–2428.
[11] ——, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *University of Michigan, Tech. Rep.*, July, 2014.
[12] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.
[13] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[14] 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.