Technical Notes and Correspondence

Supervisor Synthesis for Mealy Automata With Output Functions: A Model Transformation Approach

Xiang Yin

***Abstract*—Recently, Ushio and Takai have proposed a Mealy-automata-based framework to study the non-blocking supervisory control problem under partial observation. This framework can handle observation uncertainties by taking both state-dependent observations and nondeterministic outputs into account. In this technical note, we propose a model-transformation-based approach to solve the *supervisor synthesis problem* in this framework. First, we propose a transformation algorithm that transforms the non-blocking supervisor synthesis problem for Mealy automata to a conventional supervisory synthesis problem under partial observation, which can be solved effectively by an existing algorithm. Then we show that the supervisor synthesized for the transformed problem indeed solves the original problem. Our results bridge the gap between the conventional supervisory control framework under partial observation and the recently proposed Mealy automata framework where nondeterministic output function is used.**

***Index Terms*—Discrete event systems, mealy automata, non-blockingness, supervisor synthesis, supervisory control.**

I. Introduction

Supervisory control under partial observation is an important problem that has drawn considerable attention in the discrete-event systems (DES) literature. This problem was initiated in [3], [8], where the partial observability is modeled by using projection function or mask function. Under this conventional framework, an event in a DES is either an observable event or an unobservable event. However, in many cases, the observation of the system is both nondeterministic and state-dependent due to communications, sensor failures or measurement uncertainties. Therefore, several more advanced observation models have been proposed recently in order to handle these issues; see, e.g., [1], [4], [7], [9], [11]–[14], [17]–[19]

In [11]–[13], the problem of sensor failure tolerant supervisory control was formulated. In this framework, the occurrence of an observable event can be observed only when its associated sensor works normally. In other words, an originally observable event can become unobservable after a sensor failure occurs. In [19], the supervisory control problem was studied by using nondeterministic mask. In [4], the authors studied the decentralized supervisory control problem by assuming that the observation is state-dependent (or transition-based). In [1], [7], the supervisory control problem under observation losses was addressed. Recently, Ushio and Takai [17] provided a framework for supervisory control problem under partial observation by using Mealy automata. It was shown that Mealy automata is very suitable for modeling measurement uncertainties, since it can capture both state-dependent observations [4], [18], [20] and nondeterministic outputs, which includes intermittent sensor failures [1] and observation losses [7]. Mealy-automata-based framework has also been used in the fault diagnosis problem; see, e.g., [6], [15].

Although many different approaches have been proposed in the literature to tackle the supervisory control problem under observation uncertainties, several fundamental problems still remain. First, most of the existing works only address the *supervisor existence problem*, which requires to achieve a given specification exactly. However, the *supervisor synthesis problem*, which is more important in practice, has received little attention. In fact, the supervisor synthesis problem is fundamentally more difficult than the supervisor existence problem. For the partially-observed case, even in the conventional supervisory control framework, the supervisor synthesis problem had remained open for more than twenty years, until it was solved very recently by our work [21]. Moreover, the issue of *non-blockingness* is more difficult to handle than the safety requirement. For example, [17] only provides sufficient conditions for the existence of non-blocking supervisor and the supervisor synthesis problem is still not considered.

In this technical note, we also consider the Mealy automata framework, which is general enough to capture many other frameworks. We solve a general *supervisor synthesis problem* in this framework for both safety and non-blockingness specifications without imposing any restrictive assumption. We propose a model-transformation-based approach that consists of two stages. First, we provide a transformation algorithm that transforms the synthesis problem in the Mealy automata framework to a synthesis problem in the conventional framework. Therefore, this allows us to use the synthesis algorithm developed in [21] to solve the transformed problem. Second, we show that the proposed transformation preserves all desired properties of the closed-loop system, i.e., safety, non-blockingness and maximal permissiveness. In other words, the supervisor synthesized for the transformed system is indeed the desired maximally permissive safe and non-blocking supervisor for the original system. We believe that this transformation-based approach brings new insights to this problem and it also bridges the gap between the conventional supervisory control framework and several recently proposed frameworks.

We note that the model transformation technic proposed in this note is similar to the notion of language lifting proposed in [19]. However, they have the following significant differences. First, the notion of language lifting is defined for the purpose of *supervisor existence*, while our transformation technic is also suitable for the purpose of *supervisor synthesis*. Second, our results are developed based on the Mealy automata framework which is more general than the nondeterministic mask used in [19]. Finally, the transformation proposed in this technical note allows us to handle non-blockingness specification, which is also not addressed in [19].

Manuscript received June 13, 2016; accepted August 9, 2016. Date of publication August 17, 2016; date of current version April 24, 2017. Recommended by Associate Editor S. Takai.

The author is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: xiangyin.sjtu@gmail.com; xiangyin@umich.edu).

Digital Object Identifier 10.1109/TAC.2016.2601118

The rest of this note is organized as follows. In Section II, we review the conventional supervisory control framework under partial observation and the supervisory control problem in the Mealy automata framework by using nondeterministic output function. In Section III, we show how to effectively solve the supervisor synthesis problem in the Mealy automata framework. Specifically, a transformation algorithm is proposed and the correctness of the transformation is proved. Section IV discusses some further results for the case where the output function is deterministic. We conclude this note in Section V.

II. Discrete Event System Model

Let Σ be a finite set of events. A finite string $s = \sigma_1 \dots \sigma_n$ is finite sequence of events in Σ and we denote by $|s|$ the length of s. We use ϵ to denote the empty string with $|\epsilon| = 0$. We denote by Σ^* the set of all finite strings including ϵ. A language $L \subseteq \Sigma^*$ is a set of strings. The prefix-closure of language L is defined as $\overline{L} := \{w \in \Sigma^* : \exists v \in \Sigma^* \text{s.t.} wv \in L\}$, where "s.t." means "such that". We say that L is prefix-closed if $\overline{L} = L$. For two sets A and B, $A \subseteq B$ means that A is a subset of B; $A \subset B$ means that A is a *proper* subset of B. We also denote by $A \not\subset B$ that A is *not* a proper subset of B.

A. Standard Supervisory Control Under Partial Observation

A finite-state automaton (FSA) is $G = (Q, \Sigma, f, q_0, Q_m)$, where Q is the finite set of states, Σ is the finite set of events, $f : Q \times \Sigma \to Q$ is the partial transition function, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the set of marked states. The transition function f is extended to $Q \times \Sigma^*$ in the usual manner. The language *generated* by G is $\mathcal{L}(G) := \{s \in \Sigma^* : f(q_0, s)!\}$, where "!" means "is defined". The language *marked* by G is $\mathcal{L}_m(G) := \{s \in \Sigma^* : f(q_0, s) \in Q_m\}$.

In the supervisory control framework [10], we assume that Σ is partitioned as $\Sigma = \Sigma_c \dot\cup \Sigma_{uc}$, where Σ_c and Σ_{uc} denote the set of controllable events and the set of uncontrollable events, respectively. Under the partial observation assumption [3], [8], we also assume that Σ is partitioned as $\Sigma = \Sigma_o \dot\cup \Sigma_{uo}$, where Σ_o and Σ_{uo} denote the set of observable events and the set of unobservable events, respectively. We denote by $P : \Sigma^* \to \Sigma_o^*$ the natural projection; see, e.g., [2]. A (partial observation) supervisor is a function $S : P(\mathcal{L}(G)) \to 2^{\Sigma_c}$, where $S(s)$ is the set of events enabled after observing $s \in P(\mathcal{L}(G))$ (uncontrollable events are always enabled by default). We denote by S/G the closed-loop system under control and $\mathcal{L}(S/G)$ can be computed recursively as follows:

- $\epsilon \in \mathcal{L}(S/G)$; and
- $[s\sigma \in \mathcal{L}(S/G)]$
  $\Leftrightarrow [s\sigma \in \mathcal{L}(G)] \wedge [s \in \mathcal{L}(S/G)] \wedge [\sigma \in S(P(s)) \cup \Sigma_{uc}]$.

We define $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$. Supervisor S is said to be *non-blocking* if $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$.

Let $K = \overline{K} \subseteq \mathcal{L}(G)$ be a non-empty prefix-closed specification language. The standard supervisor control problem under partial observation (SSCPPO) is formulated as follow.

Problem II.1. (SSCPPO) Let G be a plant FSA designed with controllable events Σ_c and observable events Σ_o. Let $K \subseteq \mathcal{L}(G)$ be a non-empty specification language. Find a *non-blocking* supervisor $S : P(\mathcal{L}(G)) \to 2^{\Sigma_c}$ such that

1. S is safe, i.e., $\mathcal{L}(S/G) \subseteq \overline{K}$; and
2. S is maximally permissive, i.e., for any non-blocking S' s.t. $\mathcal{L}(S'/G) \subseteq \overline{K}$, we have $\mathcal{L}(S/G) \not\subset \mathcal{L}(S'/G)$.

Remark II.1 The reason why we require $\mathcal{L}(S/G) \not\subset \mathcal{L}(S'/G)$ rather than $\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G)$ is that a supremal solution satisfying the latter does not exist in general. Instead, there may have several incomparable maximal solutions that solve SSCPPO. It was shown in [21] that SSCPPO can be solved without any assumption on Σ_c and Σ_o.

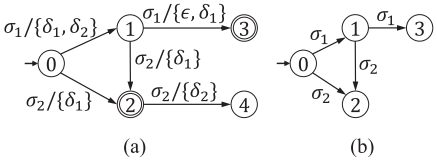

Fig. 1. For G_M, we have $\Sigma_c = \{\sigma_1, \sigma_2\}$. Also, we use double circles to denote marked states in G_M. (a) Mealy Automaton G_M. (b) Specification K.

Hereafter, we will only use the fact that SSCPPO can be effectively solved and the reader is referred to [21] for more details about the solution.

B. Supervisory Control of Mealy Automata With Output Function

In [17], a Mealy-automata-based supervisory control framework is proposed to address both nondeterministic outputs and state-dependent observations. This framework is reviewed as follows. A Mealy automaton with nondeterministic output function is

$$G_M = (Q, \Sigma, f, q_0, Q_m, \Delta, \lambda) \tag{1}$$

where (Q, Σ, f, q_0, Q_m) is a FSA, Δ is a finite set of output symbols and $\lambda : Q \times \Sigma_\epsilon \to 2^{\Delta_\epsilon}$ is the nondeterministic output function, where $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$ and $\Delta_\epsilon := \Delta \cup \{\epsilon\}$. We also denote by Δ^* the set of all finite strings over Δ including ϵ. Let $q \in Q$ be a state and $\sigma \in \Sigma$ be an event defined at q. Then $\lambda(q, \sigma)$ describes the set of potential observations for this transition, which may include ϵ. We define $\lambda(q, \epsilon) = \{\epsilon\}$ for any $q \in Q$. The generated and marked languages of G_M are the same as the generated and marked languages of the corresponding FSA, respectively, i.e., $\mathcal{L}(G_M) := \{s \in \Sigma^* : f(q_0, s)!\}$ and $\mathcal{L}_m(G_M) := \{s \in \Sigma^* : f(q_0, s) \in Q_m\}$. We denote by $O : \Sigma^* \to 2^{\Delta^*}$ the nondeterministic observation mapping. Formally, O is defined recursively by

- $O(\epsilon) = \{\epsilon\}$;
- $O(s\sigma) = \{\alpha\delta \in \Delta^* : \alpha \in O(s) \wedge \delta \in \lambda(f(q_0, s), \sigma)\}$.

Mapping O is also extended to 2^{Σ^*} by $O(L) = \{\alpha \in \Delta^* : \exists s \in L \text{ s.t. } \alpha \in O(s)\}$.

An *extended event* is a tuple $(\sigma, \delta) \in \Sigma \times \Delta_\epsilon$, where (ϵ, ϵ) is the empty extended event. An extended string is a sequence of extended events. For any extended string $s = (\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in (\Sigma \times \Delta_\epsilon)^*$, we define $\Theta_\Sigma(s) := \sigma_1 \dots \sigma_n$ and $\Theta_\Delta(s) := \delta_1 \dots \delta_n$ as its first and its second components, respectively. For any string $\sigma_1 \dots \sigma_n \in \mathcal{L}(G_M)$, we say that $\delta_1 \dots \delta_n$ is an *output realization* of $\sigma_1 \dots \sigma_n$ if $\delta_1 \dots \delta_n \in O(\sigma_1 \dots \sigma_n)$. Note that the output realization of a string is not unique in general. An extended string $s = (\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in (\Sigma \times \Delta_\epsilon)^*$ is said to be generated by G_M if $\Theta_\Sigma(s) \in \mathcal{L}(G_M)$ and $\Theta_\Delta(s)$ is an output realization of $\Theta_\Sigma(s)$. We denote by $\mathcal{L}_e(G_M)$ the set of extended strings generated by G_M.

Example II.1 Let us consider Mealy automaton G_M in Fig. 1(a), where we have $\Sigma = \{\sigma_1, \sigma_2\}$ and $\Delta = \{\delta_1, \delta_2\}$. For each transition, the set of symbols on the right hand side of "σ/" denotes $\lambda(q, \sigma)$, which is the set of potential outputs of this transition. For $\sigma_1\sigma_2 \in \mathcal{L}(G_M)$, we have that $O(\sigma_1\sigma_2) = \{\delta_1\delta_1, \delta_2\delta_1\}$. Therefore, $(\sigma_1, \delta_1)(\sigma_2, \delta_1) \in \mathcal{L}_e(G_M)$.

Under the Mealy automata framework, we still assume that $\Sigma = \Sigma_c \dot\cup \Sigma_{uc}$, where $\Sigma_c \subseteq \Sigma$ is the set of controllable events. However, we do not consider observable events, since observation has already been specified by the output function, which is more general than using Σ_o. Therefore, a supervisor is defined as a function $S : O(\mathcal{L}(G_M)) \to 2^{\Sigma_c}$. We also denote by S/G_M the closed-loop system under control and

denote by $\mathcal{L}_e(S/G_M)$ the generated extended language, which can be computed recursively as follows:

- $(\epsilon,\epsilon)\in\mathcal{L}_e(S/G_M)$;
- $[sa\in\mathcal{L}_e(G_M)]\wedge[s\in\mathcal{L}_e(S/G_M)]\wedge[\Theta_\Sigma(a)\in S(\Theta_\Delta(s))\cup\Sigma_{uc}]$ $\Leftrightarrow[sa\in\mathcal{L}_e(S/G_M)]$.

We also define

$$\mathcal{L}_{e,m}(S/G_M)=\{s\in\mathcal{L}_e(S/G_M):\Theta_\Sigma(s)\in\mathcal{L}_m(G_M)\} \quad (2)$$

$$\mathcal{L}(S/G_M)=\{s\in\Sigma^*:\exists t\in\mathcal{L}_e(S/G_M)\text{ s.t. }\Theta_\Sigma(t)=s\} \quad (3)$$

$$\mathcal{L}_m(S/G_M)=\mathcal{L}(S/G_M)\cap\mathcal{L}_m(G_M). \quad (4)$$

We say that supervisor $S:O(\mathcal{L}(G_M))\to 2^{\Sigma_c}$ is *strongly non-blocking* if $\overline{\mathcal{L}_{e,m}(S/G_M)}=\mathcal{L}_e(S/G_M)$.

Remark II.2 In the Mealy automata framework, Σ^* is the domain of the actual behavior of the system, while Δ^* is the domain of the observed behavior of the system. In other words, suppose that event $\sigma\in\Sigma$ occurs at $x\in X$, then σ is the actual event generated by the system and $\lambda(x,\sigma)$ is the set of observations the supervisor may observe. Therefore, we can only disable the occurrences of events in Σ_c. Once event $\sigma\in\Sigma$ occurs at state $x\in X$, we cannot choose observation in $\lambda(x,\sigma)$ due to measurement uncertainties. In particular, if the output is ϵ, then it means that nothing is observed upon the occurrence of σ due to the lack of sensor or sensor failure.

Remark II.3 It was shown in [17] that standard non-blockingness condition, i.e., $\overline{\mathcal{L}_m(S/G_M)}=\mathcal{L}(S/G_M)$, is not adequate for the Mealy automata framework. For example, let us consider G_M in Fig. 1(a) and consider supervisor S defined as follows:

$$S(\alpha)=\begin{cases}\emptyset & \text{if }\alpha=\delta_2\\ \{\sigma_1\} & \text{otherwise}\end{cases} \quad (5)$$

We have that $\overline{\mathcal{L}_m(S/G_M)}=\overline{\{\sigma_1\sigma_1\}}=\{\epsilon,\sigma_1,\sigma_1\sigma_1\}=\mathcal{L}(S/G_M)$. However, string σ_1 will be blocked if its output is δ_2. This is because that $\overline{\mathcal{L}_m(S/G_M)}=\mathcal{L}(S/G_M)$ only guarantees that each string can reach a marked state under a particular output realization. Recall that the output realization for a string is not unique in general. In order to resolve this issue, we need to require that any string cannot be blocked under any output realization. This is why the notion of strong non-blockingness is used in this framework.

Still, we consider a prefix-closed specification language $K=\overline{K}\subseteq\mathcal{L}(G_M)$. Then we formulate the supervisory control problem for Mealy automata with nondeterministic output function (SCPMNF).

Problem II.2. (SCPMNF) Let G_M be a plant Mealy automaton designed with controllable events Σ_c. Let $K\subseteq\mathcal{L}(G_M)$ be a specification language. Find a *strongly non-blocking* supervisor $S:O(\mathcal{L}(G_M))\to 2^{\Sigma_c}$ s.t.

1. S is safe, i.e., $\mathcal{L}(S/G_M)\subseteq\overline{K}$;
2. S is maximally permissive, i.e., for any strongly non-blocking S' s.t. $\mathcal{L}(S'/G_M)\subseteq\overline{K}$, we have that $\mathcal{L}_e(S/G_M)\not\subset\mathcal{L}_e(S'/G_M)$.

Remark II.4 Similar to the blockingness issue, the maximal permissiveness is also defined in terms of the extended language rather than the generated language. For example, let us still consider G_M in Fig. 1(a) and supervisor S defined by Equation (5). Suppose that S' is another supervisor defined by $S'(\alpha)=\{\sigma_1\},\forall\alpha\in\Delta^*$. Then we have that $\mathcal{L}(S/G_M)=\mathcal{L}(S'/G_M)=\overline{\{\sigma_1\sigma_1\}}$, i.e., their generated languages are exactly the same. However, $(\sigma_1,\delta_2)(\sigma_1,\delta_1)\in\mathcal{L}_e(S'/G_M)$ but $(\sigma_1,\delta_2)(\sigma_1,\delta_1)\notin\mathcal{L}_e(S/G_M)$. This implies that S' is more permissive than S under some possible observation realization of a string, although their generated languages are the same. This is why we choose $\mathcal{L}_e(S/G_M)\not\subset\mathcal{L}_e(S'/G_M)$ as the maximal permissiveness criterion. In general, $\mathcal{L}_e(S/G_M)\not\subset\mathcal{L}_e(S'/G_M)$ and $\mathcal{L}(S/G_M)\not\subset\mathcal{L}(S'/G_M)$ are incomparable, i.e., neither condition implies the other. One instance where these two conditions coincide is discussed in Section IV.

III. Synthesis of Maximally Permissive Non-Blocking Supervisors

In this section, we discuss how to solve SCPMNF. Our approach consists two stages:

1) First, we transform SCPMNF to SSCPPO, which can be effectively solved by the algorithm proposed in [21].
2) Then we show that the supervisor synthesized for the transformed SSCPPO also solves the original SCPMNF.

A. The Transformation Algorithm

First, we present the model transformation procedure.

Let $G_M=(Q,\Sigma,\Delta,f,\lambda,q_0,Q_m)$ be the Mealy automaton under control and $K=\overline{K}\subseteq\mathcal{L}(G_M)$ be the prefix-closed specification language. We denote by $P_\Sigma:(\Sigma\cup\Delta)^*\to\Sigma^*$ the natural projection from $\Sigma\cup\Delta$ to Σ. We construct a new FSA $\tilde{G}=(\tilde{Q},\tilde{\Sigma},\tilde{f},q_0,Q_m)$ and a new prefix-closed specification language $\tilde{K}\subseteq\mathcal{L}(\tilde{G})$ by Algorithm MODEL-TRANSF as follows.

Algorithm 1 MODEL-TRANSF

input : $G_M=(Q,\Sigma,\Delta,f,\lambda,q_0,Q_m)$, K and Σ_c.
output: $\tilde{G}=(\tilde{Q},\tilde{\Sigma},\tilde{f},q_0,Q_m)$, $\tilde{K}$, $\tilde{\Sigma}_c$ and $\tilde{\Sigma}_o$.

1 Build a new FSA $\tilde{G}=(\tilde{Q},\tilde{\Sigma},\tilde{f},q_0,Q_m)$, where
- $\tilde{Q}=Q\cup(Q\times\Sigma)$ is the set of states;
- $\tilde{\Sigma}=\Sigma\cup\Delta_\epsilon$ is the set of events;
- q_0 is the initial state, which is the same as G_M;
- Q_m is the set of marked states, which is the same as G_M;
- $\tilde{f}:\tilde{Q}\times\tilde{\Sigma}\to\tilde{Q}$ is the transition function defined by:

(i) For any $q\in Q\subset\tilde{Q}$ and $\sigma\in\Sigma\subset\tilde{\Sigma}$, we have

$$\tilde{f}(q,\sigma)=(q,\sigma)\text{ if }f(q,\sigma)! \quad (6)$$

(ii) For any $(q,\sigma)\in(Q\times\Sigma)\subset\tilde{Q}$ and $\delta\in\Delta_\epsilon\subset\tilde{\Sigma}$, we have

$$\tilde{f}((q,\sigma),\delta)=f(q,\sigma)\text{ if }\delta\in\lambda(q,\sigma) \quad (7)$$

2 Set $\tilde{K}\leftarrow P_\Sigma^{-1}(\overline{K})\cap\mathcal{L}(\tilde{G})$
3 Set $\tilde{\Sigma}_c\leftarrow\Sigma_c$
4 Set $\tilde{\Sigma}_o\leftarrow\Delta$;

In Algorithm MODEL-TRANSF, language $\tilde{K}$ defined in line 2 can be recognized as follows. Let H be the FSA generating $\overline{K}$, i.e., $\mathcal{L}(H)=\overline{K}$. First, we add self-loop for each event in Δ at each state in H and denote by $\tilde{H}$ the resulting FSA. Then, we take the product composition of $\tilde{H}$ and $\tilde{G}$ and it is easy to verify that the resulting product FSA generates $\tilde{K}$.

Remark III.1 Let us discuss the intuition of the proposed transformation algorithm. The key idea is to separate the control and observation that are originally coupled in the Mealy automaton G_M without essentially affecting the real dynamic of the system. In fact, we see that the transformed FSA $\tilde{G}$ is bipartite automaton consisting of two types of states Q and $Q\times\Sigma$. Intuitively, a state in Q is a real *system state* from which we make a control decision and then move to a state in $Q\times\Sigma$, which is just a *intermediate state*. From this intermediate state, all output symbols associated with this transition can occur and each of them leads to the same system state that is reached in the original model. This essentially captures the state-dependent nondeterministic output without affecting the system's dynamic. The set of controllable events is still Σ_c. Because the transformed FSA is bipartite, after a system event is generated, it forces an output symbol, including ϵ, to

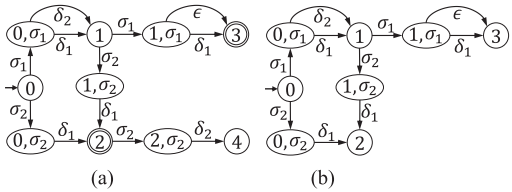

Fig. 2: For $\tilde{G}$: $\tilde{\Sigma}_c = \{\sigma_1, \sigma_2\}$ and $\tilde{\Sigma}_o = \{\delta_1, \delta_2\}$. (a) Transformed $\tilde{G}$. (b) Specification $\tilde{K}$.

occur before the next system event is generated. Finally, we would like to remark that, although event in Σ and event in Δ are generated alternatively in order, they are occurring simultaneously physically. However, we will show later that such discrepancy is not an issue for the purpose of synthesis.

Remark III.2 In [5], a similar transformation was used for the *purpose of learning and inference*. However, in the present technical note, we use the transformation for the *purpose of control synthesis*. It is not always true that a transformation that preserves some desired property for the *original system* can still preserve the same desired property for the *closed-loop system under control*. Consequently, we need to handle the issues of observability, controllability and the specification language, which do not exist in [5]. Moreover, the output function in [5] is *deterministic*, while we consider *non-deterministic output function* in the present technical note.

Let us illustrate the transformation algorithm Algorithm MODEL-TRANSF by the following example.

Example III.1 Let us still consider the Mealy automaton G_M shown in Fig. 1(a) and the specification language K generated by the FSA shown in Fig. 1(b). Then the transformed FSA $\tilde{G}$ and the transformed specification language $\tilde{K}$ are shown in Figs. 2(a) and (b), respectively. Note that the automaton that generates the transformed specification language $\tilde{K}$ can be obtained by first adding self-loops with Δ_ϵ at each state in the automaton generating K and then taking the parallel composition with the transformed plant $\tilde{G}$. Initially, if $\sigma_1 \in \Sigma$ occurs, then we move to intermediate state $(0, \sigma_1)$. From this state, two possible output δ_1 and δ_2 can occur and both of them lead to system state 1, which is the same state reached by σ_1 from state 0 in the original model G_M.

B. Correctness of the Transformation

Now, we discuss how to use the transformed model to synthesize a non-blocking supervisor that works for the original model.

We note that one important property of the transformation is that, although the event sets of G_M and $\tilde{G}$ are different, the domain of control decisions and the domain of observations for each model are exactly the same. Specifically, for both G_M and $\tilde{G}$, a control decision is an element in 2^{Σ_c} and an observation is a sequence of symbols in Δ^*. Therefore, for supervisor $S : P_\Delta(\mathcal{L}(\tilde{G})) \to 2^{\Sigma_c}$ designed for $\tilde{G}$, where $P_\Delta : \tilde{\Sigma}^* \to \Delta^*$, it is still a well-defined supervisor for G_M, since $P_\Delta(\mathcal{L}(\tilde{G})) = O(\mathcal{L}(G_M)) \subseteq \Delta^*$. This allows us to directly use the supervisor synthesized for the transformed model $\tilde{G}$ to control the original model G_M.

Note that, for any string s in $\mathcal{L}(S/\tilde{G})$, we can always write it in the form of $s = \sigma_1\delta_1 \dots \sigma_n\delta_n \in (\Sigma\Delta_\epsilon)^*$, where $\sigma_i \in \Sigma$ and $\delta_i \in \Delta_\epsilon$ That is, if two events in Σ appear in succession in s, then we insert an ϵ-event in between. The following result says that, under the control of the same supervisor $S : \Delta^* \to 2^{\Sigma_c}$, G_M and $\tilde{G}$ essentially generate the same behavior.

Lemma III.1 Let G_M be the Mealy automaton and $\tilde{G}$ be the transformed FSA. Let $S : \Delta^* \to 2^{\Sigma_c}$ be a supervisor. Then

$$(\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in \mathcal{L}_e(S/G_M) \Leftrightarrow \sigma_1\delta_1 \dots \sigma_n\delta_n \in \mathcal{L}(S/\tilde{G}). \quad (8)$$

Proof: We prove by induction on the length of the string. Clearly, $\epsilon \in \mathcal{L}(S/\tilde{G})$ and $(\epsilon, \epsilon) \in \mathcal{L}_e(S/G_M)$ by definition. Now, we assume that Equation (8) holds for $n = k$ and we want to show that it still holds for $n = k + 1$.

First, we suppose that $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(S/G_M)$. By the definition of $\mathcal{L}_e(S/G_M)$, this implies that:

1) $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S/G_M)$;
2) $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(G_M)$; and
3) $\sigma_{k+1} \in S(\delta_1 \dots \delta_k) \cup \Sigma_{uc}$.

By the induction hypothesis, 1) implies that $\sigma_1\delta_1 \dots \sigma_k\delta_k \in \mathcal{L}(S/\tilde{G})$. By the transformation algorithm, we know that $\tilde{f}(q_0, \sigma_1\delta_1 \dots \sigma_k\delta_k) = f(q_0, \sigma_1 \dots \sigma_k) =: q_k$, which also implies that $\tilde{f}(q_k, \sigma_{k+1})!$. Moreover, since $\delta_{k+1} \in \lambda(q_k, \sigma_{k+1})$, we also have $\tilde{f}(q_k, \sigma_{k+1}\delta_{k+1})!$. Since $P_\Delta(\sigma_1\delta_1 \dots \sigma_k\delta_k) = \delta_1 \dots \delta_k$, we know that $\sigma_{k+1} \in S(P_\Delta(\sigma_1\delta_1 \dots \sigma_k\delta_k))$. Moreover, since $\sigma_{k+1} \in \tilde{\Sigma}_{uo}$ and $\delta_{k+1} \in \tilde{\Sigma}_{uc}$, we have that $\sigma_1\delta_1 \dots \sigma_k\delta_k\sigma_{k+1}\delta_{k+1} \in \mathcal{L}(S/\tilde{G})$.

Next, we suppose $\sigma_1\delta_1 \dots \sigma_{k+1}\delta_{k+1} \in \mathcal{L}(S/\tilde{G})$. We also have:

1) $\sigma_1\delta_1 \dots \sigma_k\delta_k \in \mathcal{L}(S/\tilde{G})$;
2) $\sigma_1\delta_1 \dots \sigma_k\delta_k\sigma_{k+1}\delta_{k+1} \in \mathcal{L}(\tilde{G})$;
3) $\sigma_{k+1} \in S(P(\sigma_1\delta_1 \dots \sigma_k\delta_k)) \cup \Sigma_{uc} = S(\delta_1 \dots \delta_k) \cup \Sigma_{uc}$.

By the induction hypothesis, 1) implies that $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S/G_M)$. Moreover, 2) implies that $f(q_k, \sigma_{k+1})!$ and $\delta_{k+1} \in \lambda(q_k, \sigma_{k+1})$, where $q_k = f(q_0, \sigma_1 \dots \sigma_k)$. Therefore, by $\sigma_{k+1} \in S(\delta_1 \dots \delta_k) \cup \Sigma_{uc}$, we have $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(S/G_M)$. ■

With the above result, next, we show that safety is preserved under the proposed transformation.

Lemma III.2 Let G_M be the Mealy automaton, $\tilde{G}$ be the transformed FSA and $S : \Delta^* \to 2^{\Sigma_c}$ be a supervisor. Then we have

$$\mathcal{L}(S/G_M) \subseteq K \Leftrightarrow \mathcal{L}(S/\tilde{G}) \subseteq \tilde{K} \quad (9)$$

Proof: ($\Rightarrow$) Suppose that $\mathcal{L}(S/G_M) \subseteq K$. Let us consider an arbitrary string $\sigma_1\delta_1 \dots \sigma_n\delta_n \in \mathcal{L}(S/\tilde{G})$. Then, by Lemma III.1, we know that $(\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in \mathcal{L}_e(S/G_M)$, which implies $\sigma_1 \dots \sigma_n \in \mathcal{L}(S/G_M) \subseteq K$. Therefore, $\sigma_1\delta_1 \dots \sigma_n\delta_n \in P_\Sigma^{-1}(\overline{K}) \cap \mathcal{L}(\tilde{G}) = \tilde{K}$. Note that for any string in $\mathcal{L}(S/\tilde{G})$ that ends up with a symbol in Σ, we can always extend it to a string ending up with a symbol in Δ, since all events in Δ are uncontrollable. Therefore, considering string $\sigma_1\delta_1 \dots \sigma_n\delta_n$ is w.l.o.g.

($\Leftarrow$) Suppose that $\mathcal{L}(S/\tilde{G}) \subseteq \tilde{K}$. Let us consider an arbitrary string $\sigma_1 \dots \sigma_n \in \mathcal{L}(S/G_M)$. Let $(\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in \mathcal{L}_e(S/G_M)$ be an extended string such that $\delta_1 \dots \delta_n$ is an output realization of $\sigma_1 \dots \sigma_n$. By Lemma III.1, we know that $\sigma_1\delta_1 \dots \sigma_n\delta_n \in \mathcal{L}(S/\tilde{G})$. Assume that $\sigma_1 \dots \sigma_n \notin K$, then we know that $\sigma_1\delta_1 \dots \sigma_n\delta_n \notin P_\Sigma^{-1}(\overline{K}) \cap \mathcal{L}(\tilde{G}) = \tilde{K}$. However, it contradicts the fact that $\mathcal{L}(S/\tilde{G}) \subseteq \tilde{K}$. Therefore, $\sigma_1 \dots \sigma_n \in K$, i.e., $\mathcal{L}(S/G_M) \subseteq K$. ■

The following is a Corollary of Lemma III.2.

Corollary III.1 Let $S : \Delta^* \to 2^{\Sigma_c}$ be a supervisor. Then $\mathcal{L}(S/G_M) = \Theta_\Sigma(\mathcal{L}_e(S/G_M)) = P_\Sigma(\mathcal{L}(S/\tilde{G}))$, where $P_\Sigma : (\Sigma \cup \Delta)^* \to \Sigma^*$.

Next, we show that non-blockingness is also preserved under the proposed transformation.

Lemma III.3 Let G_M be the Mealy automaton, $\tilde{G}$ be the transformed FSA and $S : \Delta^* \to 2^{\Sigma_c}$ be a supervisor. Then S is a strongly non-blocking supervisor for G_M, if and only if, S is a non-blocking supervisor for $\tilde{G}$, i.e.,

$$\overline{\mathcal{L}_{e,m}(S/G_M)} = \mathcal{L}_e(S/G_M) \Leftrightarrow \overline{\mathcal{L}_m(S/\tilde{G})} = \mathcal{L}(S/\tilde{G}) \quad (10)$$

Proof: ($\Rightarrow$) By contradiction. Suppose that $\overline{\mathcal{L}_{e,m}(S/G_M)} = \mathcal{L}_e(S/G_M)$ and assume that $\overline{\mathcal{L}_m(S/\tilde{G})} \subset \mathcal{L}(S/\tilde{G})$. Let $\sigma_1\delta_1 \dots \sigma_n\delta_n \in \mathcal{L}(S/\tilde{G}) \setminus \overline{\mathcal{L}_m(S/\tilde{G})}$ be a blocked string. By Lemma III.1, we know that $(\sigma_1, \delta_1) \dots (\sigma_n, \delta_n) \in \mathcal{L}_e(S/G_M)$.

Since S is strongly non-blocking for G_M, we can find $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)(\sigma_{n+1},\delta_{n+1})\dots(\sigma_{n+k},\delta_{n+k})\in\mathcal{L}_{e,m}(S/G_M)$. Again, by Lemma III.1, we know that $\sigma_1\delta_1\dots\sigma_n\delta_n\sigma_{n+k}\delta_{n+k}\in\mathcal{L}(S/\tilde{G})$. By the definition of $\mathcal{L}_{e,m}(S/G_M)$, we know that $\sigma_1\dots\sigma_{n+k}\in\mathcal{L}_m(G_M)$. Therefore, we know that $\tilde{f}(q_0,\sigma_1\delta\dots\sigma_{n+k}\delta_{n+k})=f(q_0,\sigma_1\dots\sigma_{n+k})\in Q_m$, i.e., $\sigma_1\delta\dots\sigma_{n+k}\delta_{n+k}\in\mathcal{L}_m(S/\tilde{G})$. This contradicts the fact that $\sigma_1\delta_1\dots\sigma_n\delta_n$ is a blocked string.

($\Leftarrow$) By contradiction. Suppose that $\overline{\mathcal{L}_m(S/\tilde{G})}=\mathcal{L}(S/\tilde{G})$ and assume that $\overline{\mathcal{L}_{e,m}(S/G_M)}\subset\mathcal{L}_e(S/G_M)$. Let $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)\in\mathcal{L}_e(S/G_M)$ be a blocked string. By Lemma III.1, $\sigma_1\delta_1\dots\sigma_n\delta_n\in\mathcal{L}(S/\tilde{G})$. Since S is non-blocking for $\tilde{G}$, we can find $\sigma_1\delta_1\dots\sigma_n\delta_n\dots\sigma_{n+k}\delta_{n+k}\in\mathcal{L}_m(S/\tilde{G})$. By Lemma III.1, we know that $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)(\sigma_{n+1},\delta_{n+1})\dots(\sigma_{n+k},\delta_{n+k})\in\mathcal{L}_e(S/G_M)$. Since $\sigma_1\delta_1\dots\sigma_{n+k}\delta_{n+k}\in\mathcal{L}_m(S/\tilde{G})$, by the transformation, $f(q_0,\sigma_1\dots\sigma_{n+k})\in Q_m$, i.e., $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)(\sigma_{n+1},\delta_{n+1})\dots(\sigma_{n+k},\delta_{n+k})\in\mathcal{L}_{e,m}(S/G_M)$. This contradicts the assumption $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)$ is blocked. ■

Finally, we show that supervisor S solves SSCPPO if and only if it solves SCPMNF.

Theorem III.1 S is a maximally permissive safe and non-blocking supervisor for the transformed FSA $\tilde{G}$ with respect to $\tilde{K}$, $\tilde{\Sigma}_c$ and $\tilde{\Sigma}_o$, if and only if, S is a maximally permissive safe and strongly non-blocking supervisor for G_M with respect to K and Σ_c

Proof: ($\Rightarrow$) By contradiction. Let us assume that S is not maximally permissive for G_M. Then we know that there exists another safe and strongly non-blocking supervisor S' such that $\mathcal{L}_e(S/G_M)\subset\mathcal{L}_e(S'/G_M)$. Then, for any string $\sigma_1\delta_1\dots\sigma_n\delta_n\in\mathcal{L}(S/\tilde{G})$, by Lemma III.1, we know that $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)\in\mathcal{L}_e(S/G_M)\subset\mathcal{L}_e(S'/G_M)$. Again, by Lemma III.1, this implies that $\sigma_1\delta_1\dots\sigma_n\delta_n\in\mathcal{L}(S'/\tilde{G})$. Note that for any string $\sigma_1\delta_1\dots\sigma_n$, we can always extend it to a string in the form of $\sigma_1\delta_1\dots\sigma_n\delta_n$, since δ_n is uncontrollable. Therefore, $\mathcal{L}(S/\tilde{G})\subseteq\mathcal{L}(S'/\tilde{G})$. Now let us consider a string $(\sigma_1,\delta_1)\dots(\sigma_n,\delta_n)\in\mathcal{L}_e(S'/G_M)\setminus\mathcal{L}_e(S/G_M)$. By Lemma III.1, we know that $\sigma_1\delta_1\dots\sigma_n\delta_n\in\mathcal{L}(S'/\tilde{G})\setminus\mathcal{L}(S/\tilde{G})$. Therefore, we know that $\mathcal{L}(S/\tilde{G})\subset\mathcal{L}(S'/\tilde{G})$. Moreover, by Lemma III.2 and III.3, we know that S' is safe and strongly non-blocking for G_M implies that S' is also safe and non-blocking for $\tilde{G}$. However, this contradicts the fact that S is a maximally permissive safe and non-blocking supervisor for $\tilde{G}$.

($\Leftarrow$) Still by contradiction. Assume that S is not maximally permissive for $\tilde{G}$. Then there exists a safe and non-blocking supervisor S' such that $\mathcal{L}(S/\tilde{G})\subset\mathcal{L}(S'\tilde{G})$. By Lemma III.1, similar to the "only if" part, we know that $\mathcal{L}_e(S/G_M)\subset\mathcal{L}_e(S'/G_M)$. Moreover, by Lemma III.2 and III.3, we know that S' is safe and non-blocking for $\tilde{G}$ implies that S' is also safe and strongly non-blocking for G_M. However, this contradicts the fact that S is a maximally permissive safe and strongly non-blocking supervisor for G_M. ■

Algorithm 2 MEALY-SYNT

input: G_M, K and Σ_c.
output: S^*.
1 Obtain $\tilde{G}$ and $\tilde{K}$ by Algorithm MODEL-TRANSF;
2 Using Algorithm NB-SOLU in [21] to synthesize a maximally permissive non-blocking supervisor S^* for $\tilde{G}$ with respect to $\tilde{K}$, $\tilde{\Sigma}_c$ and $\tilde{\Sigma}_o$;
3 **return** S^* as the maximally permissive safe and non-blocking supervisor for G_M;

Based on Theorem III.1, Algorithm MEALY-SYNT is proposed in order to solve SCPMNF. First, we transform the Mealy automaton G_M and its specification K to $\tilde{G}$ and $\tilde{K}$, respectively, and formulate the transformed parameters as an instance of SSCPPO. Then we exploit Algorithm NB-SOLU recently proposed in [21] to solve the transformed standard supervisory control problem. Algorithm NB-SOLU is rather complicated and it is beyond the scope of this technical note; the reader is referred to [21] for more details and examples. Note that the complexity of Algorithm NB-SOLU is exponential in the size of $\tilde{G}$ and the transformed model $\tilde{G}$ is linear in the size of the original Mealy automaton G_M. Therefore, the overall complexity of Algorithm MEALY-SYNT is exponential in the size of G_M. However, such an exponential complexity is known to be unavoidable under the partial observation setting [16].

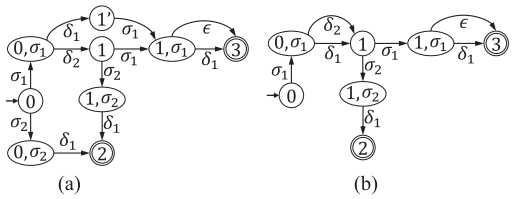

Fig. 3. Two incomparable maximal solutions. (a) $\mathcal{L}(S_1/\tilde{G})$. (b) $\mathcal{L}(S_2/\tilde{G})$.

We illustrate the transformation-based synthesis method by the following example.

Example III.2 We still consider the Mealy automaton G_M shown in Fig. 1(a) and the specification language K generated by the FSA shown in Fig. 1(b). Then the transformed FSA $\tilde{G}$ and the transformed specification language $\tilde{K}$ have been shown in Figs. 2(a) and (b), respectively, where $\tilde{\Sigma}_c=\Sigma$ and $\tilde{\Sigma}_o=\Delta$. Note that, it was shown in [21] that SSCPPO does not have a unique supremal supervisor in general and it may have several *incomparable* maximal supervisors. For example, for the transformed problem, there are two incomparable maximal supervisors S_1 and S_2, whose closed-loop behaviors are shown in Figs. 3(a) and (b), respectively. For S_1, by choosing to enable both σ and σ_2 initially, we cannot enable σ_2 after observing δ_1, since we are not sure whether the system is at state 1 or 2. However, for S_2, by choosing to disable σ_2 initially, event σ_2 can be enabled after observing δ_1, since we know for sure that the current state is 1. Note that, although $\mathcal{L}_e(S_1/G_M)$ and $\mathcal{L}_e(S_2/G_M)$ are incomparable, for this example, we have that $\mathcal{L}(S_2/G_M)\subset\mathcal{L}(S_1/G_M)$. But, as we discussed earlier, it does not imply that S_1 is more permissive than S_2, since S_1 does not allow σ_2 to occur after observing δ_2, while it is allowed by S_2.

IV. Further Results on Maximal Permissiveness

In the above development, we have shown how to synthesize a maximally permissive safe and non-blocking supervisor for Mealy automata with nondeterministic output function. Note that the maximally permissiveness is defined in terms of the extended language of the closed-loop system, i.e., a supervisor S is maximal if for any other safe and non-blocking supervisor S', we have $\mathcal{L}_e(S/G_M)\not\subset\mathcal{L}_e(S'/G_M)$. The motivation for the this requirement has been discussed in Remark II.4. However, in addition to the above requirement, one may impose an additional requirement in terms of the generated language $\mathcal{L}(S/G_M)$ and require that, for any other safe and non-blocking supervisor S', we also have $\mathcal{L}(S/G_M)\not\subset\mathcal{L}(S'/G_M)$. In general, the synthesized maximally permissive supervisor does not satisfy the above requirement. For example, the supervisor S_2 in Fig. 3(b) is a maximally permissive supervisor, but $\mathcal{L}(S_2/G_M)\subset\mathcal{L}(S_1/G_M)$.

Hereafter, we consider a special case, under which the synthesized supervisor is not only maximally permissive in terms of the extended language but also maximally permissive in terms of the generated language. We say that the output function $\lambda:Q\times\Sigma_\epsilon\to 2^{\Delta_\epsilon}$ is *deterministic* if $\forall s\in Q,\forall\sigma\in\Sigma_\epsilon:|\lambda(q,\sigma)|=1$. Then the following result

reveals that the synthesized maximally permissive supervisor is also maximal in terms of the generated language when the output function is deterministic.

Theorem IV.1 Suppose that the output function λ for G_M is deterministic. Let S a maximally permissive safe and strongly non-blocking supervisor. Then for any other safe and strongly non-blocking supervisor S', we have that $\mathcal{L}(S/G_M) \not\subset \mathcal{L}(S'/G_M)$.

Proof: By contradiction. Suppose that S a maximally permissive safe and strongly non-blocking supervisor. We assume that there exists another safe and strongly non-blocking supervisor S' such that $\mathcal{L}(S/G_M) \subset \mathcal{L}(S'/G_M)$.

First, we claim that $\mathcal{L}_e(S/G_M) \subseteq \mathcal{L}_e(S'/G_M)$. To see this, we show that $s \in \mathcal{L}_e(S/G_M)$ implies $s \in \mathcal{L}_e(S'/G_M)$ by induction on the length of s. Clearly, the induction basis holds, since $(\epsilon, \epsilon) \in \mathcal{L}_e(S/G_M)$ and $(\epsilon, \epsilon) \in \mathcal{L}_e(S'/G_M)$ by definition. Now, let us assume that for any $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S/G_M)$, we have $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S'/G_M)$. For the induction step, let us consider an extended string $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(S/G_M)$. This implies that

1) $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S/G_M)$;
2) $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(G_M)$;
3) $\sigma_{k+1} \in S(\delta_1 \dots \delta_k)$.

Since the output function is deterministic, the output realization of $\sigma_1 \dots \sigma_2$ is unique, which is $\delta_1 \dots \delta_n$. Therefore, we know that $\sigma_{k+1} \in S'(\delta_1 \dots \delta_k)$; otherwise $\sigma_1 \dots \sigma_k \sigma_{k+1} \notin \mathcal{L}(S'/G_M)$, which contradicts the fact that $\mathcal{L}(S/G_M) \subset \mathcal{L}(S'/G_M)$. Moreover, by the induction hypothesis, 1) implies that $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k) \in \mathcal{L}_e(S'/G_M)$. Therefore, we know that $(\sigma_1, \delta_1) \dots (\sigma_k, \delta_k)(\sigma_{k+1}, \delta_{k+1}) \in \mathcal{L}_e(S'/G_M)$.

Since $\mathcal{L}(S/G_M) \subset \mathcal{L}(S'/G_M)$, we consider a string $\sigma_1 \dots \sigma_n \sigma_{n+1} \in \mathcal{L}(S'/G_M)$ such that $\sigma_1 \dots \sigma_n \in \mathcal{L}(S/G_M)$ but $\sigma_1 \dots \sigma_n \sigma_{n+1} \notin \mathcal{L}(S/G_M)$. This implies that $\sigma_{n+1} \notin S(\delta_1 \dots \delta_n)$. Still, since the output function is deterministic, we know that the unique output realization of $\sigma_1 \dots \sigma_2$ is $\delta_1 \dots \delta_n$, i.e., for any $(\sigma_1, \delta_1') \dots (\sigma_n, \delta_n') \in \mathcal{L}_e(S/G_M)$, we have $\delta_i' = \delta_i, i = 1, \dots n$. Therefore, $\sigma_{n+1} \notin S(\delta_1 \dots \delta_n)$ implies that $\sigma \dots \sigma_n \sigma_{n+1} \notin \mathcal{L}(S/G_M)$. This together with $\mathcal{L}_e(S/G_M) \subseteq \mathcal{L}_e(S'/G_M)$ implies that $\mathcal{L}_e(S/G_M) \subset \mathcal{L}_e(S'/G_M)$, which contradicts the fact that S is a maximally permissive supervisor. ■

Remark IV.1 It is worth remarking that using Mealy automata with deterministic output function is still more general than using the conventional natural projection or mask framework, since it allows state-dependent observations. In fact, the supervisory control problem under *transition-based observation* investigated by [4] is essentially a special case of Mealy automata with deterministic output function. Specifically, we can define $\Delta = \Sigma$ and for each $q \in Q, \sigma \in \Sigma$, we have $\lambda(q, \sigma) = \sigma$ or ϵ. Namely, $\lambda(q, \sigma) = \sigma$ if this transition is observable and $\lambda(q, \sigma) = \epsilon$ if this transition is unobservable. Although [4] studies a decentralized control problem, only supervisor existence conditions are provided. Therefore, our transformation-based approach also solves the supervisor synthesis problem under transition-based observation in the centralized setting. Moreover, an example was provided in [4] to show that we cannot transform the FSA model for the purpose of control by simply renaming the same event that has different observation properties at different transitions. The issue of this approach is that changing the label of an event may change its controllability. However, the result in this technical note suggests that transition-based observation can be reduced to projection-based observation for the centralized case by separating the issues of controllability and observability appropriately for each transition. We believe similar results can also be established for the decentralized case by using the proposed transformation.

V. Conclusion

In this technical note, we investigated the supervisor synthesis problem in the Mealy automata framework. A model transformation algorithm was proposed to transform this problem to a supervisor synthesis problem in the conventional framework. We showed that the supervisor synthesized for the transformed problem solves the original synthesis problem, since the transformation preserves all desired properties of the closed-loop system. Therefore, the general non-blocking supervisor synthesis problem under observation uncertainties was solved.

Acknowledgment

The author thanks the anonymous reviewers for their useful comments on improving this technical note.

References

[1] M. V. S. Alves, J. C. Basilio, A. E. Carrilho da Cunha, L. K. Carvalho, and M. V. Moreira. Robust supervisory control against intermittent loss of observations. In *Proc. 12th Int. Workshop Disc. Event Syst.*, vol. 12, pp. 294–299, 2014.

[2] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. New York: Springer, 2nd ed., 2008.

[3] R. Cieslak, C. Desclaux, A. S. 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.

[4] Y. Huang, K. Rudie, and F. Lin. Decentralized control of discrete-event systems when supervisors observe particular event occurrences. *IEEE Trans. Autom. Contr.*, vol. 53, no. 1, pp. 384–388, 2008.

[5] M. N. Irfan, C. Oriat, and R. Groz. "Model inference and testing, "in *Advances in Computers*, vol. 89, pp. 89–139, 2013.

[6] N. Kanagawa and S. Takai, "Diagnosability of discrete event systems subject to permanent sensor failures," *Int. J. Control*, vol. 88, no. 12, pp. 2598–2610, 2015.

[7] F. Lin, "Control of networked discrete event systems: Dealing with communication delays and losses," *SIAM J. Control Optimiz.*, vol. 52, no. 2, pp. 1276–1298, 2014.

[8] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 173–198, 1988.

[9] S.-J. Park and K.-H. Cho, "Delay-robust supervisory control of discrete event systems with bounded communication delays," *IEEE Trans. Autom. Contr.*, vol. 51, no. 5, pp. 911–915, 2006.

[10] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optimiz.*, vol. 25, no. 1, pp. 206–230, 1987.

[11] K. Rohloff. "Sensor failure tolerant supervisory control," in *Proc. 44th IEEE Conf. Decision and Control*, pp. 3493–3498, 2005.

[12] K. Rohloff, "Bounded sensor failure tolerant supervisory control," in *Proc. 11th Int. Workshop Discrete Event Systems*, pp. 272–277, 2012.

[13] A. M. Sánchez and F. J. Montoya. "Safe supervisory control under observability failure," *Discrete Event Dynam. Syst.: Theory & Appl.,* vol. 16, no. 4, pp. 493–525, 2006.

[14] S. Shu and F. Lin, "Supervisor synthesis for networked discrete event systems with communication delays," *IEEE Trans. Autom. Control*, vol. 60, no. 8, pp. 2183–2188, 2015.

[15] S. Takai and T. Ushio, "Verification of codiagnosability for discrete event systems modeled by mealy automata with nondeterministic output functions," *IEEE Trans. Autom. Control*, vol. 57, no. 3, pp. 798–804, 2012.

[16] J. N. Tsitsiklis, "On the control of discrete-event dynamical systems," *Math. Control, Signals and Syst.*, vol. 2, no. 2, pp. 95–107, 1989.

[17] T. Ushio and S. Takai, "Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions," *IEEE Trans. Autom. Control*, vol. 61, no. 3, pp. 799–804, 2016.

[18] W. Wang, A. R. Girard, S. Lafortune, and F. Lin, "On codiagnosability and coobservability with dynamic observations," *IEEE Trans. Autom. Control*, vol. 56, no. 7, pp. 1551–1566, 2011.

[19] S. Xu and R. Kumar, "Discrete event control under nondeterministic partial observation," in *Proc. IEEE Conf. Aut. Sci. Eng.*, pp. 127–132, 2009.

[20] X. Yin and S. Lafortune, "Codiagnosability and coobservability under dynamic observations: Transformation and verification," *Automatica*, vol. 61, pp. 241–252, 2015.

[21] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially observed discrete event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 5, pp. 1239–1254, 2016.