A Learning-based Synthesis Approach to Decentralized Supervisory Control of Discrete Event Systems with Unknown Plants

Jin Dai and Hai Lin

Abstract—**In this paper, we consider the problem of automatic synthesis of decentralized supervisor synthesis for uncertain discrete event systems. In particular, we study the case when the uncontrolled plant is unknown *a priori*. To deal with the unknown plants, we first characterize the co-normality of prefix-closed regular languages and propose formulas for computing the supremal co-normal sublanguages; then sufficient conditions for the existence of decentralized supervisors are given in terms of language controllability and co-normality and a learning-based algorithm to synthesize the supervisor automatically is proposed. Moreover, the paper also studies the on-line decentralized supervisory control of concurrent discrete event systems that are composed of multiple interacting unknown modules. We use the concept of modular controllability to characterize the necessary and sufficient conditions for the existence of the local supervisors, which consist of a set of local supervisor modules, one for each plant module and which determines its control actions based on the locally observed behaviors, and an on-line learning-based local synthesis algorithm is also presented. The correctness and convergence of the proposed algorithms are proved, and their implementation are illustrated through examples.**

Index Terms—**Discrete-event systems, supervisor synthesis, regular language learning, controllability, decentralized control.**

I. INTRODUCTION

The discrete event system (DES) supervisory control theory initiated by Ramadge and Wonham [14], [15] has been widely used to model and control large-scale systems, including multi-agent systems, traffic networks and manufacturing systems, see e.g., [2,8] and references therein. In [14], the notion of language controllability is introduced to propose the necessary and sufficient condition for the existence of a supervisor that achieves a desired controlled behavior for a given discrete event system under the complete observation of events. When the events are not completely observed by the supervisor, an additional condition of observability introduced by Cieslak et al. [3] is adopted for the existence of the supervisor. Since more and more complex systems built up nowadays are becoming physically distributed and networked, such as multi-agent systems and computer networks, a monolithic (centralized) control design that requires the computation of the global plant often suffers from the so called "state-explosion" problem since this computation grows exponentially as the number of components in the plant grow [17]; motivated by this fact, the decentralized control architecture of DESs, in which the specification is achieved through the joint efforts of more than one supervisors, has arisen in the study of supervisory control problems and has attracted many researchers' interest, see e.g. [18] [7] [13].

This work was supported by the National Science Foundation (NSF-CNS-1239222 and NSF- EECS-1253488)

The authors are with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. {jdai1, hlin1}@nd.edu.

Lots of efforts have been devoted to the research of decentralized supervisory control problems. In [11] a sufficient condition for the existence of decentralized supervisors that the controlled behavior of the system lies in a given range expressed by local specifications was proposed. Cieslak et al. [3] considered the decentralized control where the specification is given as a prefix-closed regular language, and the property of co-observability was introduced in place of observability and the result was then generalized to the case of non-prefix-closed specification by Rudie and Wonham [18]. In [20], Willner and Heymann studied the decentralized supervisory control, except that the system they considered were of concurrent nature.They introduced the notion of language separability and under the assumption that $\Sigma_{i,uc} \cap \Sigma_j = \varnothing$ for all $i \neq j$, they proved that the separability and controllability of the specification is a necessary and sufficient condition that guarantees that the decentralized control can achieve the optimal behavior achievable by a centralized supervisor. In [6], the authors studied a version of decentralized control problem that is more general than those reported above, and provided a necessary and sufficient condition for the existence of decentralized supervisors for keeping the controlled behavior of the system in a given range. Results of [11] and [20] can be viewed as special cases of [6].

However, most of the existing synthesis algorithms for either monolithic or decentralized supervisors require prior and complete knowledge of the uncontrolled plant. This requirement has been pointed out to be unreasonable [5] in some cases due to the uncertain nature of the plant. The first case is the environment uncertainty, when dealing with online supervisory control problems, it is often impossible to determine when and where the underlined processes are terminated and/or initiated, which implies the time-varying nature of the DES plant to be studied. Secondly, for large-scale or concurrent systems that are composed of multiple interacting modules, obtaining a precise plant model requires the computation of the compositional behaviors of all the components and hence suffers from the aforementioned state-explosion problem. Thirdly, even though we can obtain a precise DES model before the system is running, if some unknown faults or failures occur during the evolution of the controlled system, which may add unknown transitions and change the controllability and/or observability of the events, we can hardly determine the post-fault model in an online manner, and then constructing a consistent DES model is impossible.

There have been some studies of the uncertainty of the plant in centralized supervisory control. Lin considered the plant as a set of possible plants and designed robust supervisor applicable for the whole range of plants [10]. In recent years, fault-tolerant control scheme has been proposed to deal with the faults occurring during the evolution of the system. Wen et al. [19] proposed a framework of fault-tolerant supervisory control of DESs, in which the supervisor was designed to ensure the recovery from fault. Liu and Lin [12] investigated the reliable decentralized supervisory control problem of DES under the general architecture, which seek the minimal number of supervisors required for correct functionality of the supervised systems.

This paper differs from the aforementioned work in the sense that we focus on automatic synthesis of decentralized supervisors when the plant is unknown instead of partially known or bounded cases. The contribution of this paper is as follows: first, we propose a sufficient condition for the existence of decentralized supervisor in terms of language controllability and co-normality; secondly, to deal with the uncertain or even unknown nature of the plant, we propose an L^* learning based synthesis algorithm where new dynamical membership queries are used instead of static ones in the original learning procedure, and the algorithm can synthesize a sub-optimal decentralized supervisors that are consistent with the supremal controllable and co-normal sublanguage of the given prefix-closed specification language; thirdly, we go one step further and study the decentralized control problem of concurrent systems, and modular synthesis algorithm for the local supervisors is proposed.

The remainder of this paper is organized as follows. Section II briefly reviews the result of decentralized supervisory control theory. The background information about L^* learning procedure is presented in Section III. The discussion about language co-normality and the sufficient conditions for the existence of decentralized supervisor based on co-normality are given in Section IV. The modified L^* learning algorithm for supervisor synthesis is provided in section V. Necessary and sufficient conditions for the existence of local supervisors for concurrent discrete event systems are discussed in Section VI, along with the synthesis algorithm for local supervisors. At last we provide concluding remarks and discuss the future work.

II. Notions and Preliminaries

This paper discusses the supervisory control framework for discrete event systems that is developed by Ramadge and Wonham [14] [15]. For the readers convenience, some background results from the cited references are first provided in this section. For a detailed introduction of the theory, readers may refer to [2] [8].

An uncontrolled system, called a *plant*, is modeled by a deterministic finite automaton (DFA) $G = (Q, \Sigma, q_0, \delta, Q_m)$, where Σ is the set of events, Q is the set of states, $q_0 \in Q$ is the initial state, $Q_m \subseteq Q$ is the set of marked states, δ is the (partial) transition function. Let Σ^* denote the set of all finite strings over Σ plus the null string ϵ, then δ can be extended to $\delta : Q \times \Sigma^* \rightarrow Q$ in the natural way. The languages generated by G is given by $L(G) = \{s \in \Sigma^* | \delta(q_0, s) \text{ is defined}\}$ and the language marked by G is given by $L_m(G) = \{s \in \Sigma^* | s \in L(G), \delta(q_0, s) \in Q_m\}$. The prefix closure $\overline{K}$ of a language $K \subseteq \Sigma^*$ is the set of all prefixes of strings in K. K is called *prefix-closed* if $K = \overline{K}$.

In supervisory control theory, the event set are partitioned into the set of controllable events and the set of uncontrollable events, i.e., $\Sigma = \Sigma_{uc} \dot{\cup} \Sigma_c$. A supervisor S is a pair (R, ψ) where R is a DFA which recognizes a language over the same event set as G, and ψ is a feedback map from the event set and the states of R to the set $\{enable, disable\}$. If X denotes the event set of R, then $\psi : \Sigma \times X \rightarrow \{enable, disable\}$ satisfies $\psi(\sigma, x) = enable$ if $\sigma \in \Sigma_{uc}$, i.e., the supervisor can only disable the occurrences of controllable events. R is considered to be driven by the strings generated by G, and in turn, at each state of R, the control policy $\psi(\sigma, x)$ makes the decision about the occurrence of event $sigma$ at the corresponding state of G. The behavior of the supervised system (a.k.a., closed-loop systen) is represented by a DFA S/G. The language generated by the closed-loop system is denoted by $L(S/G)$ and its marked language is given by $L_m(S/G) = L(S/G) \cap L_m(G)$. It is worth noting that, for regular language cases, the control exercised by such a supervisor for a plant G is equivalent to the behavior of another automaton S that runs in parallel with G and at each state of G disables a subset of the controllable events, if S is a sub-automaton of G. Therefore, $L(S/G) = L(S||G)$ and $L_m(S/G) = L_m(S||G) = L(S||G) \cap L_m(G)$, where "$||$" stands for the parallel composition of two automata [8]. Given a non-empty prefix-closed specification $K \subseteq L(G)$, a supervisor S exists such that $L(S||G) = K$ if and only if K is *controllable*, i.e., $\overline{K}\Sigma_{uc} \cup L(G) \subseteq \overline{K}$ [14]. If not, then a supervisor is synthesized to achieve the supremal controllable (also prefix-closed) sublanguage of K, namely, $\sup C(K)$.

Moreover, the supervisor may also be constrained to observe only events in a specified set of observable events and the event set is then divided into the subset of unobservable and observable events, i.e., $\Sigma = \Sigma_{uo} \dot{\cup} \Sigma_o$. The presence of partial observation can be captured by an the natural projection mapping $P : \Sigma \rightarrow \Sigma_o \cup \{\epsilon\}$ that erases the occurrence of all unobservable events. A prefix-closed language $K \subseteq L(G)$ is said to be *observable* if the conditions $s, t \in \overline{K}, \sigma \in \Sigma$, $P(s) = P(t)$, $s\sigma \in \overline{K}$, and $t\sigma \in L(G)$ together imply $t\sigma \in \overline{K}$ [9]. Given a non-empty prefix-closed specification $K \subseteq L(G)$, a supervisor S exists such that $L(S||G) = K$ if and only if K is controllable and observable [2].

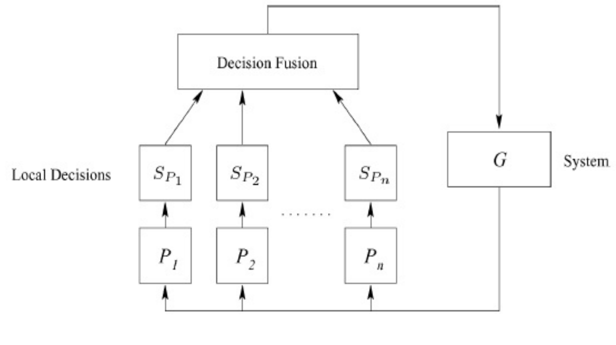


Fig. 1 Decentralized control architecture. [2]

In this paper, we start from the general decentralized supervisory control problem as shown in Fig. 1, where the plant is controlled jointly by n local supervisors, each of which observes the locally observable events and controls the locally controllable events. Let $I = \{1, 2, \ldots, n\}$ denote the indices of the supervisors. For each $i \in I$, let Σ_i denote the i-th local event set, hence $\Sigma = \bigcup_{i\in I} \Sigma_i$. Σ_i is divided into the set of controllable events $\Sigma_{i,c}$ and the set of uncontrollable events $\Sigma_{i,uc}$. Let $\Sigma_{i,o}$ and $\Sigma_{i,uo}$ denote the sets of locally observable and unobservable events, respectively. For notational simplicity, the set of globally controllable events is denoted as $\Sigma_c = \bigcup_{i\in I} \Sigma_{i,c}$ and the globally observable event set is denoted as $\Sigma_o = \bigcup_{i\in I} \Sigma_{i,o}$. The sets $\Sigma_{uc} = \Sigma - \Sigma_c = \bigcap_{i\in I} \Sigma_{i,uc}$ and $\Sigma_{uo} = \Sigma - \Sigma_o = \bigcap_{i\in I} \Sigma_{i,uo}$ denote the uncontrollable and unobservable event sets, respectively, with the natural projection mapping $P_{\Sigma_i} : \Sigma^* \to \Sigma_{i,o}^*$. We use P_i to replace P_{Σ_i} and use P to denote the natural projection from Σ^* to Σ_o^* in the rest of this paper. The closed-loop behavior of the DES under decentralized supervisors $\{S_i\}_{i\in I}$ is denoted as $L(\{\tilde{S}_i\}_{i\in I}, G)$, where $\tilde{S}_i$ is formed as an extended version of S_i by letting $\tilde{S}_i$ not observe any events in $\Sigma - \Sigma_i$ and permanently enables events in $\Sigma - \Sigma_{i,c}$.

Let $In(\sigma) = \{i \in I | \sigma \in \Sigma_{i,c}\}$ denote the index set. A language $K \subseteq L(G)$ is said to be

- *normal* [3] [9] if $P^{-1}P(K) \cap L(G) = K$.
- *C&P co-observable* with respect to $A \subseteq \Sigma_c$ [7] if for any $s \in \overline{K}$ and any $\sigma \in A$ such that $s\sigma \in L(G) - \overline{K}$, then there exists $i \in In(\sigma)$ such that: for any $s' \in \overline{K}$, $[P_i(s) = P_i(s')] \wedge [s'\sigma \in L(G)] \Rightarrow [s'\sigma \notin \overline{K}]$.
- *D&A co-observable* with respect to $A \subseteq \Sigma_c$ [7] if for any $s \in \overline{K}$ and any $\sigma \in A$ such that $s\sigma \in \overline{K}$, then there exists $i \in In(\sigma)$ such that: for any $s' \in \overline{K}$, $[P_i(s) = P_i(s')] \wedge [s'\sigma \in L(G)] \Rightarrow [s'\sigma \in \overline{K}]$.

Remark 1: The terms "$C\&P$" and "$D\&A$" in the definition of co-observability stands for "conjunctive architecture and permissive decision rule" and "disjunctive architecture and anti-permissive decision rule", which corresponds to an "AND" and "OR" fusion rule in Fig. 1, respectively. In general, we may use "co-observability" when the considered language is either $C\&P$ or $D\&A$ co-observable.

Let us first introduce a motivating example to illustrate the problems to be solved in this paper.

Example 1: Consider a multi-robot system that consists of two *Pioneer3-DX* robots, one with an automated robotic arm and the other is equipped with a gripper, as shown in Fig. 2, respectively.

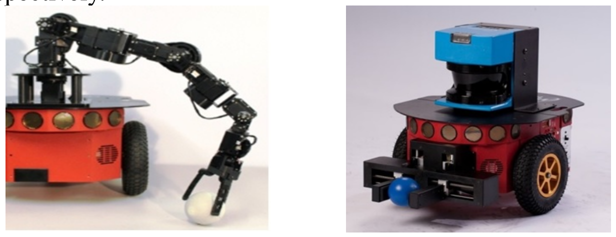


Fig. 2 Robot 1 with a robotic arm and Robot 2 with a gripper

The working procedure of the two robots is depicted in Fig. 3. Robot 1 starts from its "home" Home 1, and grabs some parts ffrom Workstation 1, and then transports the parts to Workstation 2 along the Rails 1 and 3. Once the parts are processed by Workstation 2, Robot 2, which starts form Home 2, will pick them up and transport them between Home 2 and Workstations 1 and 2 along the Rails 2 and 3. Rails 1, 2 and 3 are all bidirectional.

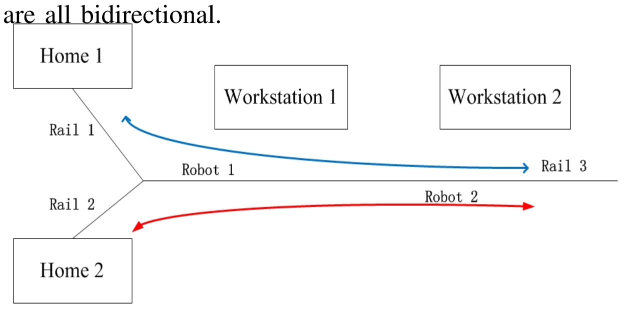
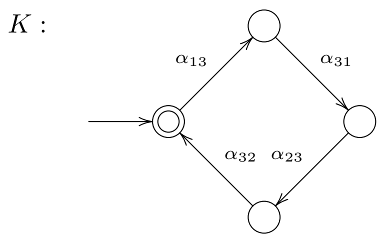


Fig. 3 Multi-robot coordination system

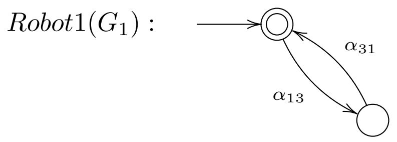
The control specifications for the two-robot system are:

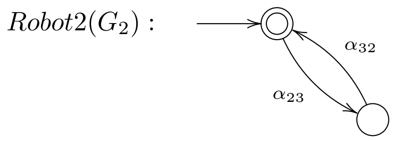
- Robot 2 shall pick up a part delivered by Robot 1 after it has been processed at Workstation 1.
- Since the rails are bidirectional and since Rail 3 is shared by the two robots, the robots cannot be on Rail 3 at the same time.

For $i \in I = \{1, 2\}$, define α_{i3} and α_{3i} to be the events representing that Robot i is transporting from Rail i to Rail 3 and back from Rail 3 to Rail i, respectively. Then the local event sets are $\Sigma_1 = \{\alpha_{13}, \alpha_{31}\}$ and $\Sigma_2 = \{\alpha_{23}, \alpha_{32}\}$. Without loss of generality, we assume that all the local events are locally controllable, i.e., $\Sigma_{uc} = \Sigma_{1,uc} = \Sigma_{2,uc} = \varnothing$. From above, the specification for the two robots is given by $K = \overline{(\alpha_{13}\alpha_{31}\alpha_{23}\alpha_{32})^*}$, and a DFA that recognizes K is given as follows.



Note that the whole plant model of the motion of each robot is only partially given (with respect to the movement along the given rails), and one possible motion model pair (not unique representation) is given as follows, we have to design decentralized (local) supervision rules for each robot such that K can be fulfilled by the joint work of Robots 1 and 2.





Based on this example, there arises two problems naturally: first, whether or not we can synthesize the decentralized supervisors for each robot such that the specification is achieved;

secondly, if the answer to the first problem is positive, then whether or not we can compute the local supervisors based only the local events and avoid the computation of the behaviors $G_1||G_2$ so that the complexity can be reduced. After the discussion in Sections V and VI we will revisit this illustrative example.

Formally, in this paper, we aim at solving the problem that given a non-empty prefix-closed specification $K \subseteq L(G)$, find the decentralized supervisors S_i such that $L(\{S_i\}_{i\in I}, G)$ with no prior knowledge of G, and we expect to apply a modified L^* learning method.

III. L^* LEARNING

In this section, we briefly introduce the background information of the L^* algorithm for regular language learning, which plays an important role in the derivation of our supervisor synthesis algorithms. The L^* learning algorithm introduced by Angluin [1] and improved by Rivest and Schapire [16] learns an unknown regular language U over alphabet Σ and produces a minimal deterministic finite automaton (DFA) that accepts it. The algorithm infers the structure of the DFA by asking an *oracle* that answers two types of queries. The first type is a *membership query*, in which L^* asks whether a string $s \in \Sigma^*$ is included in U. The second type is a *conjecture*, in which L^* constructs a conjectured DFA M and asks whether M is such that $L(M) = U$. If $L(M) \neq U$ the oracle returns a counterexample, which is a string s in the symmetric difference of $L(M)$ and U. At any given time, L^* has, in order to construct the conjectured DFA M, information about a finite collection of strings over Σ, classified either as members or non-members of U. L^* creates an *observation table* to incrementally record and maintain the information whether strings in Σ^* belong to U. The observation table is a three-tuple (S, E, T) consisting of: a non-empty finite set S of prefix-closed languages, a non-empty finite set E of suffix-closed languages and a function $T : (S \cup S\Sigma)E \to \{0, 1\}$ which is often referred to as the *membership oracle*, i.e., the function T takes strings in $s \in (S \cup S\Sigma)$ onto 0 if they are not in K, otherwise the function T returns 1.

The ith observation table constructed by L^* will be denoted as T_i. Each table can be depicted as a 2-dimensional array whose rows are labeled by strings $s \in S \cup S\Sigma$ and whose columns are labeled by symbols $\sigma \in E$. The entries in the labeled rows and columns are given by the function value $T(s\sigma)$. The *row function* $row : (S \cup S\Sigma) \to \{0, 1\}^{|E|}$ denotes the table entries in the row labeled by string $s \in S \cup S\Sigma$.

An observation table (S, E, T) is said to be

- *closed* if for all $t \in S\Sigma$, there exists an $s \in S$ such that $row(t) = row(s)$.
- *consistent* if there exist strings $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, and for all $\sigma \in \Sigma$, $row(s_1\sigma) = row(s_2\sigma)$.
- *complete* if it is closed and consistent.

Once the observation table is complete, a candidate DFA $M(S, E, T) = (Q, q_0, \delta, Q_m)$ over the alphabet Σ is constructed isomorphically by the following rules:

$$\begin{aligned}
&Q = \{row(s) : s \in S\}, \\
&q_0 = row(\epsilon), \\
&Q_m = \{row(s) : (s \in S) \wedge (T(s) = 1)\}, \\
&\delta(row(s), \sigma) = row(s\sigma).
\end{aligned}$$

The L^* learning algorithm learns the target unknown regular language in the following manner [1]:

Algorithm 1 L^* learning algorithm [1].

```
Set $S = \epsilon$ and $E = \epsilon$.
Use the membership oracle to form the initial observation table $T_i(S, E, T)$ where $i = 1$
while $T_i(S, E, T)$ is not completed do
  if $T_i$ is not consistent then
    find $s_1, s_2 \in S$, $\sigma \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $T(s_1\sigma e) \neq T(s_2\sigma e)$;
    Add $\sigma e$ to $E$;
    extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries to the oracle.
  end if
  if $T_i$ is not closed then
    find $s_1 \in S$, $\sigma \in \Sigma$ such that $row(s_1\sigma)$ is different from $row(s)$ for all $s \in S$;
    Add $s_1\sigma$ to $S$;
    extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries to the oracle.
  end if
end while
Once $T_i$ is complete, let $M_i = M(T_i)$ as the conjectured acceptor of $K$
Ask the oracle the validity of $M_i$.
if the oracle declares that the conjecture to be false and a counterexample $t \in \Sigma^*$ is generated then
  Add $t$ and all its prefixes into $S$;
  extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries to the oracle;
end if
Set $i = i + 1$ and return to while until the oracle declares that the conjecture is true.
return $M_i$.
```

The DFA M is presented as a conjecture to the oracle. If the conjecture is correct, i.e., $L(M) = U$, then the oracle returns "True" with the current DFA M; otherwise, a counterexample $c \in (U - L(M)) \cup (L(M) - U)$ is generated by the oracle. The L^* algorithm analyzes the counterexample c and finds the longest prefix c_p of c that witnesses the difference between $L(M)$ and U. Adding c_p to S reflects the difference in next conjecture by splitting states in M. Once c_p is added to S, L^* iterates the entire process to update M with respect to c_p.

The L^* algorithm is guaranteed to construct a minimal DFA accepting the unknown regular language U using only $O(|\Sigma|n^2 + n\log m)$ membership queries and at most $n - 1$ equivalence queries, where n is the number of states in the final DFA and m is the length of the longest counterexample provided by the oracle when answering equivalence queries [1].

IV. Decentralized Control Based on Co-normality

In this section we study the decentralized supervisory control problem of discrete event systems with partial observations, which is first introduced in [11]. Furthermore, we generalize the setting of [11] by relaxing the assumption that the local event set Σ_i is the union of local controllable events $\Sigma_{i,c}$ and local observable events $\Sigma_{i,o}$, which is clearly restrictive since in many systems, there may exist events that are neither locally controllable nor locally observable.

We start from the notion of language decomposability introduced by Rudie and Wonham [18], which plays an important role in decentralized supervisory control theory.

Definition 1: A language $K \subseteq L(G)$ is said to be *decomposable* (with respect to G and $\{P_i\}_{i\in I}$) if $K = L(G) \cap (\bigcap_{i\in I} P_i^{-1}P_i(K))$.

In general, decomposability is stronger than co-observability, i.e., a language K is decomposable implies that K is also co-observable. However, under certain conditions of decentralized local control [18], decomposability and co-observability are equivalent.

The following lemma lays the foundation of the necessary and sufficient conditions for the existence of decentralized supervisors in terms of language decomposability.

Lemma 1: [6] Let G be a plant, Σ be the global event set and $\Sigma_i \subseteq \Sigma$ be the local event sets associated with natural projection P_i, $i \in I$. A language $K \subseteq L(G)$ is decomposable (with respect to G and $\{P_i\}_{i\in I}$) if and only if there exists a group of languages $K_i \subseteq P_i(L(G)), i \in I$ such that $K = L(G) \cap (\bigcap_{i\in I} P_i^{-1}K_i)$.

Remark 2: If we extend $L(G) = \Sigma^*$, then the condition for language decomposability in view of Lemma 1 is reduced to the existence of $\{K_i \subseteq P_i(\Sigma^*) = \Sigma_i^*\}$ that satisfies $K = \bigcap_{i\in I} P_i^{-1}K_i$, which is exactly the property called language separability in [20], and will be discussed in Section VII.

Based on Lemma 1, the following theorem provides a necessary and sufficient condition for the existence of decentralized supervisors based on language decomposability.

Theorem 1: [6] Let G be a plant, Σ be the global event set, and $\Sigma_i \subseteq \Sigma, i \in I$ be the local event sets. Let P_i be the natural projection from Σ to Σ_i. Then for a given non-empty, prefix-closed specification language $K \subseteq L(G)$, decentralized (local) supervisors $\{S_i, i \in I\}$ exist such that $L(\{\tilde{S}_i\}_{i\in I}, G) = K$ if and only if

$$K = L(G) \cap (\bigcap_{i\in I} P_i^{-1}(\inf \underline{PCO}_i(P_i(K))))$$

where $\tilde{S}_i$ is the extended version of S_i for the global system, and $\inf \underline{PCO}_i(P_i(K))$ denotes the infimal prefix-closed, controllable (with respect to $\Sigma_{i,uc}$ and $P_i(L(G))$) and observable (with respect to P_i and $P_i(L(G))$) superlanguage of $P_i(K)$

According to Lemma 3.2 in [20], when the global plant is of concurrent nature, then if the control specification $K \subseteq L(G)$ is prefix-closed and controllable (with respect to Σ_{uc} and $L(G)$), $P_i(K) \subseteq P_i(L(G))$ and $P_i(K)$ is also prefix-closed, and controllable (with respect to $\Sigma_{i,uc}$ and $P_i(L(G))$). In general, we can have the following corollary from Theorem 1.

Corollary 1: Let G be a plant, Σ be the global event set, and $\Sigma_i \subseteq \Sigma, i \in I$ be the local event sets. Let P_i be the natural projection from Σ to Σ_i. Then for a given non-empty, prefix-closed specification language $K \subseteq L(G)$, decentralized (local) supervisors $\{S_i, i \in I\}$ exist such that $L(\{\tilde{S}_i\}_{i\in I}, G) = K$ if and only if K is Σ_{uc}-controllable and $\{P_i\}_{i\in I}$-decomposable.

When the given specification K is not controllable or decomposable, a supervisor synthesis problem arises that we need to find decentralized supervisors to achieve a controllable and decomposable sublanguage of K. However, it has been pointed out that decomposability is not closed under unions [13], hence it is not guaranteed that the set of controllable and decomposable sublanguages of a given prefix-close language contains a unique supremal element, which implies that no optimal solution (in the sense of maximal permissiveness) exists for the decentralized supervisory control problem. An alternative option to obtain a sub-optimal solution takes advantage of the following co-normality property.

Definition 2: A language $K \subseteq L(G)$ is said to be *co-normal* with respect to $\{P_i\}_{i\in I}$ if $K = L(G) \cap (\bigcup_{i\in I} P_i^{-1}P_i(K))$

The following theorem provides a sufficient condition for the existence of decentralized supervisors in terms of co-normality.

Theorem 2: Let G be a plant, Σ be the global event set, and $\Sigma_i \subseteq \Sigma, i \in I$ be the local event sets. Let P_i be the natural projection from Σ to Σ_i. Then for a given non-empty, prefix-closed specification language $K \subseteq L(G)$, if K is Σ_{uc}-controllable and $\{P_i\}_{i\in I}$-co-normal, then there exist decentralized supervisors $\{S_i, i \in I\}$ such that $L(\{\tilde{S}_i\}_{i\in I}, G) = K$.

Proof Since co-normality always implies co-observability [18], then the existence of decentralized supervisors can be further guaranteed by controllability and prefix-closedness. ■

It is proved that co-normality is preserved under arbitrary unions and is a stronger property than co-observability [13] [18]. Since controllability and prefix-closedness of regular languages are also preserved under union, which implies that the supremal prefix-closed, controllable and co-normal sublanguage of a given prefix-closed language K, denoted as $\sup CCN(K)$, does exist.

Note that for a non-empty, prefix-closed language $K \subseteq L(G)$, the supremal normal language of K with respect to $L(G)$ and projection P, denoted as $\sup N(K)$, can be computed using the following formula [2]

$$\sup N(K) = L(G) - [P^{-1}P(L(G) - K)]\Sigma^* \qquad (1)$$

Based on (1), we propose the following formula for the computation of supremal co-normal language of a given language K with respect to $\{P_i\}_{i\in I}$, denoted as $\sup CN(K)$.

Theorem 3: For a language $K \subseteq L(G)$, the supremal co-normal sublanguage of K is given by

$$\sup CN(K) = L(G) - [\bigcup_{i\in I} P_i^{-1}P_i(L(G) - K)]\Sigma^* \qquad (2)$$

Proof The proof of Theorem 3 follows immediately the proof of the correctness of Eq. (1) in [2], with simply replacing P by $\bigcup_{i\in I} P_i^{-1}P_i$ in the proof. ■

V. Apply L^* learning for the synthesis of decentralized supervisors

In the previous work of Yang et al. [21] [22], they applied L^*-based algorithm for supervisor synthesis, and in their framework, the knowledge of the plant behaviors is confined to a limited lookahead window, which was first introduced by Chung and Lafortune [5]. However, this assumption is difficult in realization. Due to this drawback of the limited lookahead windows, in this section, we derive a modified L^* learning algorithm to synthesize the supervisor with an totally unknown plant model.

Recall that a language K is controllable if $\overline{K}\Sigma_{uc} \cup L(G) \subseteq \overline{K}$, therefore it is difficult to verify the controllability of the given specification language due to lack of knowledge of the plant, hence modification to the existing L^* learning procedure is required. We solve this difficulty by using dynamical membership queries that is capable of learning the supremal controllable sublanguage of the given specification (note that since the specification language is prefix-closed, its supremal controllable sublanguage is also prefix-closed), and hence a supervisor is synthesized.

A. Learning for controllability

We start from solving the following basic centralized supervisor synthesis problem (BSSP), and next we will take advantage of the results and apply them for decentralized supervisor synthesis.

Problem 1: Given non-empty, prefix-closed specification language $K \subseteq L(G)$, with the unknown plant G, find a supervisor S such that $L(S||G) = \sup C(K)$, where $\sup C(K)$ is the supremal controllable sublanugage of K.
We aim at solving BSSP by modifying the L^* membership queries and counterexample classes so that it can learn $\sup C(K)$ of a given prefix-closed specification K.

A string s generated by G is said to be *legal* with respect to K if $s \in K$, and s is said to be *illegal* if $s \notin K$. In this paper, the dynamical membership queries presented to the oracle in the modified L^* is based on the observed illegal behaviors generated by the system along with the given specification language. A plant behavior $st \in \Sigma^*$ is said to be *uncontrollably illegal* if s is legal, $t \in \Sigma_u^*$ and $st \notin K$. Let C denote the collection of observed uncontrollably illegal behaviors. We define the operator

$$D_u(C) = \{s \in L(G) : \exists t \in \Sigma_{uc}^* \text{ such that } st \in C\}$$

for C to represent the collection of the strings formed by discarding the uncontrollable suffixes of strings in C, and let C_i denote the set of uncontrollably illegal behaviors after the i-th iteration, then if a new uncontrollably illegal behavior s_i (a new counterexample) is generated by the oracle, we update C_i to $C_{i+1} = \{s_i\} \cup C_i$. Finally, we propose the following membership oracle T_i for $i \in \mathbb{N}$. For $t \in \Sigma^*$, let $T(t)$ denote the membership Boolean function, initially,

$$T_1(t) = \begin{cases} 0, & \text{if } t \notin K \\ 1. & \text{otherwise} \end{cases} \tag{3}$$

For $i > 1$

$$T_i(t) = \begin{cases} 0, & \text{if } T_{i-1}(t) = 0 \text{ or } t \in D_u(C_i)\Sigma^* \\ 1. & \text{otherwise} \end{cases} \tag{4}$$

Note that different from conventional L^* discussed in Section III, the dynamical membership queries used in (3) and (4) are dynamical.

The L^*-based synthesis algorithm for BSSP is given as Algorithm 2.

Algorithm 2 L^* synthesis algorithm for BSSP.

```
1:  Set $S = \epsilon$ and $E = \epsilon$.
2:  Use the membership oracle to form the initial observation table $T_i(S, E, T)$ where $i = 1$
3:  while $T_i(S, E, T)$ is not completed do
4:      if $T_i$ is not consistent then
5:          find $s_1, s_2 \in S$, $\sigma \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $T(s_1\sigma e) \neq T(s_2\sigma e)$;
6:          Add $\sigma e$ to $E$;
7:          extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries (3) and (4).
8:      end if
9:      if $T_i$ is not closed then
10:         find $s_1 \in S$, $\sigma \in \Sigma$ such that $row(s_1\sigma)$ is different from $row(s)$ for all $s \in S$;
11:         Add $s_1\sigma e$ to $S$;
12:         extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries (3) and (4).
13:     end if
14: end while
15: Once $T_i$ is completed,let $M_i = M(T_i)$ as the acceptor; make the conjecture that $M_i$ is the correct supervisor
16: if the counterexample oracle declares that the conjecture to be false and a counterexample(illegal behavior) $t \in \Sigma^*$ is generated then
17:     Add $t$ and all its prefixes into $S$;
18:     update $T_i$ to $T_{i+1}$ by using the counterexample $t$;
19:     extend $T_i$ to $(S \cup S\Sigma)E$ using membership queries to the oracle;
20: end if
21: Set $i = i + 1$, reset the conjectured supervisor and return to while until the oracle declares that the conjecture is true.
22: return $M_i$.
```

Next we establish the convergence and correctness properties of the Algorithm 2 by construction.

First, the following lemma is presented to give an iterative method to compute $\sup C(K)$.

Lemma 2: If K and $L(G)$ are regular languages, then $\sup C(K)$ can be computed iteratively as the following [8]:

$$K_1 := K, \tag{5}$$

$$K_{i+1} := K_i - [(L(G) - K_i)/\Sigma_{uc}]\Sigma^* \tag{6}$$

If there exists $m \in \mathbb{N}$ such that $K_{m+1} = K_m$, then $\sup C(K) = K_m$.

Compare (5), (6) with the dynamical membership oracle $T_i(t)$ in (3) and (4). Initially, $T_1(t)$ is consistent with $K_1 = K$; and when the iteration step $i \geq 2$, the counterexamples occur and form the set C. We use the operator $D_u(C)$ to compute the languages formed by discarding the suffixes made up with uncontrollable events of strings in C, then at the ith step of the iteration, we obtain the result in the form of $K - D_u(C_i)\Sigma^*$. On the other hand, if K is prefix-closed, then Lemma 1 suggests that $\sup C(K)$ is also prefix-closed, and the iterations (5) and (6) can be reduced to

$$\sup C(K) = K - [(L(G) - K)/\Sigma_{uc}^*]\Sigma^* \tag{7}$$

When the iteration step i goes up until no more counterexamples are generated from the counterexample oracle (provided that the algorithm is convergent), the collection $\{C_i\}$ will eventually equal to $L(G) - K$, which is the collection of all the illegal strings and hence the collection of all the potential illegal strings. At the ith step, the result we

$$K - D_u(C_i)\Sigma^* = K - [D_u(L(G) - K)]\Sigma^*$$

which coincides with the results obtained in (6) and is indeed $\sup C(K)$.

Next we will show that by using the L^* with the dynamical membership queries given in (3) and (4), the learning procedure will converge within a finite number of iteration steps. To illustrate the finite convergence and correctness of Algorithm 2, we first introduce the following two lemmas.

Lemma 3: Assume that the observation table (S, E, T) is closed and consistent and suppose that the corresponding acceptor, $M(S, E, T)$, has n states. Then for any other acceptor M' that is consistent with T that has n or fewer states, M' is isomorphic to M. [1]

Lemma 4: Assume that the observation table (S, E, T) is complete and that the corresponding acceptor $M(S, E, T)$ has n states. If a string $s \in \Sigma^*$ is a counterexample and is added to update $M(S, E, T)$ using Algorithm 1. Assume that the updated and completed observation table is (S', E', T') with the corresponding acceptor M', then M' must have at least $(n + 1)$ states. [22]

There are two kinds of "counterexamples" that are used the modified L^*: counterexamples used to complete observation table T_i and counterexamples generated by the counterexample oracle to update the new observation table T_{i+1}. We denote D as the set of those "counterexamples" that are used to complete T_i after a new illegal string s is detected and added to update S of the current observation table (S, E, T).

Consider the complete observation table $T = (S_j, E_j, T_i)$ associated with the acceptor $M(S_j, E_j, T_i) = M$, where S_j and E_j are the pre-defined S and E after the jth counterexample from D has been added to make T complete, and T_i denote the current membership oracle after i times of iterating Algorithm 2. Let n_{ij} denote the number of states of M, n_i denotes the number of states of the acceptor of T_i, and $n^\uparrow$ denotes the number of states of the acceptor $M(\sup C(K))$. Then by Lemma 2 and 3, we know that$\{n_i\}$ and $\{n_{ij}\}$ are both monotonically increasing sequences (with respect to i and j, respectively.) and $n_{ij} \leq n_i$ for any fixed i and all $j \in \mathbb{N}$. Hence we can conclude that by using L^* learning procedure, the iteration using membership oracle (3) and (4) are convergent. The following theorem summarizes the correctness and convergence properties.

Theorem 4: Assume that $K \subseteq L(G)$ is prefix-closed, then the modified L^* learning procedure using membership queries (3) and (4) converges to a supervisor S, such that $L(S||G) = \sup C(K)$. Furthermore, this iteration procedure of synthesizing S will be done in a finite number of counterexample tests.

B. Learning for Co-normality

1) Learning for Co-normality: We now consider using L^* to learn $\sup CCN(K)$. To compute $\sup CCN(K)$ using L^* learning procedure, we first consider how to deal with the co-normality and local observation projections. For $j \geq 1$, define recursively

$$K_1 = K, \tag{8}$$

$$C_{cn}(K_j) = \left\{ s \in \overline{K}_j : \exists s' \in L(G), \bigcup_{i\in I} P_i^{-1}P_i(s) = \bigcup_{i\in I} P_i^{-1}P_i(s'), s' \notin \overline{K}_j \right\}, \tag{9}$$

$$\tilde{K}_j = K_j - C_{cn}(K_j) \tag{10}$$

to denote the collection of indistinguishable (with respect to $\bigcup_{i\in I} P_i^{-1}P_i$ and K) behaviors. It follows immediately from Theorem 3 that by using the $C_{cn}(\cdot)$ operator, the above iteration will converge to the supremal co-normal sublanguage of the given prefix-closed specification $K \subseteq L(G)$ within a finite number of steps.

2) Modified membership queries: To capture the illegal behavior generated by the unknown plant under partial observations in the decentralized control structure, we modify C in previous section to be

$$\tilde{C} = \left\{ st \in \bigcup_{i\in I} P_i^{-1}P_i(L(G)) : s \in \bigcup_{i\in I} P_i^{-1}P_i(\tilde{K}), t \in \Sigma_{uc}^*, st \notin \bigcup_{i\in I} P_i^{-1}P_i(\tilde{K}) \right\} \tag{11}$$

Then, we define the following membership queries $\tilde{T}_j, j \in \mathbb{N}$ as follows:

$$\tilde{K}_1 = K - C_{cn}(K)$$

$$\tilde{T}_1(t) = \begin{cases} 0, & \text{if } t \notin \bigcup_{i\in I} P_i^{-1}P_i(\tilde{K}_1)) \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

$$K_j = L(M(T_{j-1})), j > 1.$$

If $t \in C_{cn}(K_j)$, remove t from K_j to obtain $\tilde{K}_j$, and,

$$\tilde{T}_j(t) = \begin{cases} 0, & \text{if } \tilde{T}_{j-1}(t) = 0 \text{ or } t \in D_u(\tilde{C}_j)\Sigma_o^* \\ 1, & \text{otherwise} \end{cases} \tag{13}$$

The algorithm of supervisor synthesis is summarized as Algorithm 3.

Algorithm 3 L^* for learning $\sup CCN(K)$.

```
1: Set $S = \epsilon$ and $E = \epsilon$.
2: Use the membership oracle to form the initial observation table $\tilde{T}_j(S, E, \tilde{T})$ where $j = 1$
3: while $\tilde{T}_j(S, E, \tilde{T})$ is not completed do
4:     if $\tilde{T}_j$ is not consistent then
5:         find $s_1, s_2 \in S$, $\sigma \in \Sigma_o$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $\tilde{T}(s_1\sigma e) \neq \tilde{T}(s_2\sigma e)$;
6:         Add $\sigma e$ to $E$;
7:         extend $\tilde{T}_j$ to $(S \cup S\Sigma)E$ using membership queries (12) and (13).
8:     end if
9:     if $\tilde{T}_j$ is not closed then
10:        find $s_1 \in S$, $\sigma \in \Sigma_o$ such that $row(s_1\sigma)$ is different from $row(s)$ for all $s \in S$;
11:        Add $s_1\sigma e$ to $S$;
12:        extend $\tilde{T}_j$ to $(S \cup S\Sigma)E$ using membership queries (12) and (13).
13:    end if
14: end while
15: Once $\tilde{T}_j$ is completed,let $\tilde{M}_j = M(\tilde{T}_j)$ as the acceptor; make the conjecture that $\tilde{M}_j$ is the DFA that recognizes $\sup CCN(K)$
16: if the counterexample oracle declares that the conjecture to be false and a counterexample (indistinguishable behavior) $t \in C_{cn}(\tilde{K}_j)$ is generated then
17:     remove $t$ from $\tilde{K}_j$.
18: end if
19: Obtain $\tilde{K}_{j+1}$.
20: if the counterexample oracle declares that the conjecture to be false and a counterexample(illegal behavior) $t \in \Sigma^*$ is generated then
21:     Add $P(t)$ and all its prefixes into $S$;
22:     update $\tilde{T}_j$ to $\tilde{T}_{j+1}$ by using the counterexample $t$;
23: end if
24: Set $i = i + 1$, reset the conjectured supervisor and return to while until the oracle declares that the conjecture is true.
25: return $M_i$.
26: Let $S_i$ be a DFA over $\Sigma_i$ such that $L(S_i) = P_i(M_n)$, then the obtained $S_i, i \in I$ are the decentralized supervisors.
```

3) Correctness and convergence: The following theorem states the convergence and correctness of the modified L^* by using membership queries (12) and (13).

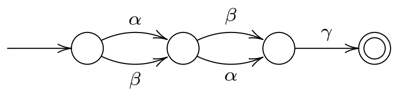
Theorem 5: Let $K \subseteq L(G)$ be a non-empty and prefix-closed specification, then L^* with dynamical membership queries (12) and (13) converges to decentralized supervisors $S_i, i \in I$, such that $L(\{S_i\}_{i\in I}, G) = supCCN(K)$. Furthermore, this iteration procedure of synthesizing S will be done in a finite number of counterexample tests.

Proof The convergence property of Algorithm 3 can be shown using the similar approach of Theorem 4 and is omitted here. We show that the obtained language from Algorithm 3 is $\sup CCN(K)$. First, we claim that the obtained language $\tilde{K}$ is co-normal, which can be proved by contradiction. For the $i-$th step of iteration, assume that $\tilde{K}_i$ is not co-normal, then there exists a string $s \in \overline{\tilde{K}_i}$ and another string $t \in L(\overline{G})$ such that $\bigcup_{i\in I} P_i^{-1}P_i(s) = \bigcup_{i\in I} P_i^{-1}P_i(t)$ but $t \notin \overline{\tilde{K}_i}$; then by the definition of $C_{cn}(K)$, it is clear to find that $s \in C_{cn}(K_i)$ and should be eliminated from $\tilde{K}_i$. Thus we get the contradiction and $\tilde{K}_i$ is a co-normal and so is $\tilde{K}$. Next we show that $\tilde{K} = \sup CCN(K)$. In fact, by comparing dynamical membership queries (12) and (13) with (3) and (4), respectively, we alternatively compute the supremal co-normal sublanguage and the supremal controllable sublanguage in each iteration, hence it is clear to show that the obtained language $\tilde{K} = \sup CCN(K)$. ■

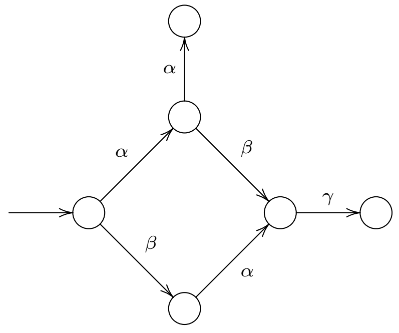
4) Illustrative Example: The effectiveness of Algorithm 3 is illustrated through the following simple example.

Example 2: Consider the global event set $\Sigma = \{\alpha, \beta, \gamma\}$. The local controllable events and observable events are given by $\Sigma_{1,c} = \{\alpha\gamma\}$, $\Sigma_{2,c} = \{\beta\gamma\}$, $\Sigma_{1,o} = \{\alpha\}$ and $\Sigma_{2,o} = \{\beta\}$, respectively. The specification is given as $K = \overline{\alpha\alpha + (\alpha\beta + \beta\alpha)\gamma}$ and the language generated by the plant is $L(G) = \overline{(\alpha + \beta)(\alpha + \beta)\gamma}$. Note that $L(G)$ is not known to the supervisors. Both $L(G)$ and K are depicted as the following.

$G:$



$K:$

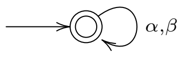


We start from the first observation table, and set $S = E = \{\epsilon\}$ for Algorithm 1. The first complete observation table with its corresponding acceptor is given as the following table (note that we only care about the rows whose first entries are 1's).

TABLE I
T_1 IN EXAMPLE

T_1		ϵ
S	ϵ	1
$S\Sigma - S$	α	1
	β	1

$M(T_1):$

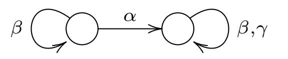


We detect that the string $\alpha\beta\beta \in M(T_1) - P(K_1)$, hence it is a counterexample and we use Algorithm 1 to update and complete the observation table.

TABLE II
T_2 IN EXAMPLE

T_2		ϵ	α
S	ϵ	1	1
	α	1	0
	$\alpha\beta$	1	0
$S\Sigma - S$	$\alpha\delta$	1	0
	β	1	0
	$\alpha\beta\gamma$	1	0

$M(T_2)$:

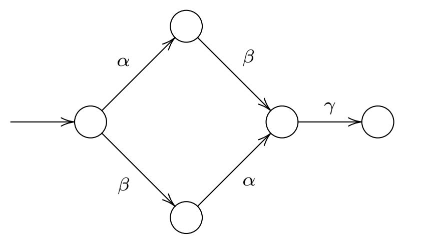


We use $M(T_2)$ as the supervisor to control the plant behaviors, the string $\beta\alpha\gamma\gamma \in M(T_2) - P(K_2)$ is a counterexample, then we add the prefixes of $\alpha\delta\delta\delta$ into S and update the observation table to T_3.

TABLE III
T_3 IN EXAMPLE

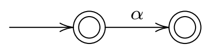
T_2		ϵ	α	β	γ
S	ϵ	1	1	1	0
	α	1	0	1	0
	$\alpha\beta$	1	0	0	1
	β	1	1	0	0
	$\beta\alpha$	1	0	0	1
	$\beta\alpha\gamma$	1	0	0	0
$S\Sigma - S$	$\alpha\beta\gamma$	1	0	0	0

$M(T_3)$:

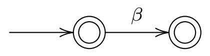


In this case no more counterexamples are detected, and we can conclude that $\sup CCN(K) = \overline{(\alpha\beta + \beta\alpha)\gamma}$, the local(decentralized) supervisors can then be obtained such that $S_i = P_i(\sup CCN(K)), i = 1, 2$, which are depicted respectively as follows.

S_1 :



S_2 :



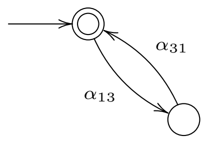
VI. LOCAL SYNTHESIS OF THE DECENTRALIZED SUPERVISORS

A. Discussion and alternation of Algorithm 3

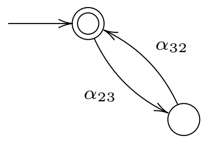
In the previous section, we discuss the synthesis problem for decentralized supervisors, and Algorithm 3 is proposed to illustrate the learning-based synthesis approach in detail. However, the approach adopted by Algorithm 3 still has some drawbacks, and can be illustrated by revisiting the motivating example mentioned in Section II.

Example 3: (Motivating example revisited) For the specification $K = \overline{(\alpha_{13}\alpha_{31}\alpha_{23}\alpha_{32})^*}$, since $\Sigma_c = \Sigma = \{\alpha_{13}, \alpha_{31}, \alpha_{23}, \alpha_{32}\}$, i.e., all the events are globally controllable, therefore, a monolithic supervisor that drives the global system to achieve K always exists; however, if the local plant model of Robots 1 and 2 are given by the following, then we can find that, by applying Algorithm 3 one shall fail to get a non-trivial (non-empty) solution.

$Robot1(G_1)$:



$Robot2(G_2)$:



The drawbacks of Algorithm 3 are twofold:

- *Centralized synthesis for decentralized supervisors* although the synthesis algorithm returns the DFAs that jointly achieve the supremal controllable and co-normal sublanguage of the given non-empty, prefix-closed specification $K \subseteq L(G)$, the algorithm itself is of centralized nature and uses the information of controllability and observability of all the global events as if it learns a monolithic supervisor.
- *Conservativeness of the result* Algorithm 3 provides an approach to actively learn $\sup CCN(K)$ when prior knowledge of the plant is unaccessible; however, as the following proposition implies, $\sup CCN(K)$ is conservative in some cases.

Proposition 1: If $K \subseteq L(G)$ is co-normal with respect to $\Sigma_i, i \in I$, then K is normal with respect to each Σ_i, respectively.

Proof The inclusion $K \subseteq P_i^{-1}P_i(K) \cap L(G)$ is always satisfied for any language $K \subseteq L(G)$. Therefore it is sufficient to prove that $K \supseteq P_i^{-1}P_i(K) \cap L(G)$. In fact, K is co-normal with respect to $\Sigma_i, i \in I$ implies that $K = L(G) \cap (\bigcup_{i \in I} P_i^{-1}P_i(K))$, thus

$$\begin{aligned} K &= L(G) \cap (\bigcup_{i \in I} P_i^{-1}P_i(K)) \\ &= \bigcup_{i \in I} [P_i^{-1}P_i(K)) \cap L(G)] \\ &\supseteq P_i^{-1}P_i(K)) \cap L(G), \forall i \in I \end{aligned}$$

Hence, $K = P_i^{-1}P_i(K) \cap L(G)$, which implies that K is normal with P_i, $i \in I$. ■

Intuitively, Proposition 1 states that, if a language $K \subseteq L(G)$ is co-normal, then it is normal with all the local observation mappings, i.e., if there exists $i \in I$ such that K is not normal with respect to P_i, then we can conclude that K is not co-normal. This proposition implies that all the decentralized supervisors can pass the authority to any single one of them. The following proposition is an immediate corollary of Proposition 1.

Proposition 2: For a non-empty and prefix-closed specification language $K \subseteq L(G)$, $\sup CCN(K) \subseteq \sup CN_i(K)$ for all $i \in I$, where $\sup CN_i(K)$ denotes the supremal controllable (with respect to Σ_{uc} and $L(G)$) and normal (with respect to P_i and $L(G)$) sublanguage of K).

Propositions 1 and 2 provide some insights on the derivation of an alternative approach to obtain a "better" synthesis solution than $\sup CCN(K)$, and can be implemented by using the following manner.

- *Step 1* Compute $\sup CN_i(K)$ for each $i \in I$ by using the global uncontrollable event set Σ_{uc} and the local observable events in $\Sigma_{i,o}$.
- *Step 2* Find $\sup_i(\sup CN_i(K))$ and return it as the solution for the synthesis problem.

For simplicity, we consider the problem of learning $\sup CN_i(K)$ for a fixed $i \in I$; and one needs to modify the membership queries (12) and (13) in Algorithm 3 to take normality into account.

The following lemma provides an iterative method of computing the supremal normal sublanguage of a given language.

Lemma 5: If $K \subseteq L(G)$, then the supremal normal sublanguage of K with respect to $\Sigma_{i,o}$, denoted as $\sup N_i(K)$, can be obtained using the following iteration. [8]

$$K_0 := K, \tag{14}$$

$$K_{j+1} := K_i - [P_i^{-1}P_i(L(G) - K_j)]\Sigma^* \tag{15}$$

If there exists $m \in \mathbb{N}$ such that $K_{m+1} = K_m$, then $K_m = \sup N_i(K)$.

Now that if K is prefix-closed, so is $\sup N_i(K)$. Therefore, the iteration in (14) and (15) can be reduced to

$$\sup N_i(K) = K - [P_i^{-1}P_i(L(G) - K)]\Sigma^* \tag{16}$$

To compute $\sup N_i(K)$ using L^* learning procedure, similar to $C_{cn}(\cdot)$, we define

$$K_1 = K, \tag{17}$$

$$C_o(K_j) = \{s \in K_j : \exists s' \in L(G), P_i(s) = P_i(s'), s' \notin K_j\}, \tag{18}$$

$$\tilde{K}_j = K_j - C_o(K_j) \tag{19}$$

to denote the collection of "locally indistinguishable" (with respect to P_i and K) behaviors generated by the controlled plant. Moreover, we define the modified C to be

$$\tilde{C} = \left\{st \in P_i(L(G)) : s \in P_i(\tilde{K}), t \in \Sigma_{uc}^*, st \notin P_i(\tilde{K})\right\}$$

Then, for the partial observed plant, we define the following membership queries $\tilde{T}_j, j \in \mathbb{N}$ as follows:

$$\tilde{K}_1 = K - C_o(K)$$

$$\tilde{T}_1(t) = \begin{cases} 0, & \text{if } t \notin P_i(\tilde{K}_1)) \\ 1, & \text{otherwise} \end{cases} \tag{20}$$

$$K_j = L(M(T_{j-1})), j > 1.$$

If $t \in C_o(K_j)$, remove t from K_j to obtain $\tilde{K}_j$, then,

$$\tilde{T}_j(t) = \begin{cases} 0, & \text{if } \tilde{T}_{j-1}(t) = 0 \text{ or } t \in D_u(\tilde{C}_j)\Sigma_{i,o}* \\ 1, & \text{otherwise} \end{cases} \tag{21}$$

When we use Algorithm 3 as the supervisor synthesis algorithm except that we use membership queries (20) and (21) to replace (12) and (13), respectively, the following theorem guarantees that the resulting learning procedure can still converge to $\sup CN_i(K)$ for a fixed $i \in I$ within a finite number of iterations.

Theorem 6: Assume $K \subseteq L(G)$ is non-empty and prefix-closed, then the modified L^* learning procedure in Algorithm 3 by using membership queries (20) and (21) for each $i \in I$ converges to a local supervisor S_i, such that $L(S_i||G) = \sup CN_i(K)$. Furthermore, this iteration procedure of synthesizing S_i will be done in a finite number of counterexample tests.

Proof The proof of Theorem 6 can be performed in exactly the same way as the proof of Theorem 5, and is omitted here.■

B. Local synthesis for decentralized supervisors: language separability and online control

Although by using Algorithm 3 with membership queries (20) and (21), one can synthesize decentralized supervisors that can achieve less conservative solutions than $\sup CCN(K)$ when given specification K. However, Algorithm 3 is still of centralized nature when using membership queries (20) and (21) since it requires the global controllability of all the events; furthermore, Algorithm 3 involves partial observation of local supervisors, which increases the computational complexity (as suggested in [4], in the worst case synthesizing a supervisor under partial observation is an NP-complete problem) and therefore it is difficult to implement the algorithm in an on-line manner. In this section, we go beyond Algorithm 3 and aim at proposing a fully decentralized learning-based algorithm to synthesize local supervisors. In particular, in this section, we assume the global plant is a concurrent discrete-event system, i.e., $G = ||_{i\in I}G_i$, where G_i is the subplant over the local alphabet Σ_i, and $L(G) = ||_{i\in I}L(G_i) = \bigcap_{i\in I} P_i^{-1}L(G_i)$ according to Proposition 3.1 in [20]. Furthermore, we assume that the local supervisor S_i can observe all the local events, i.e., $\Sigma_i = \Sigma_{i,o}$ and that any events shared by more than one local plants agree on the status of controllability, i.e.,

$$\forall i, j \in I, i \neq j, \Sigma_{i,cu} \cap \Sigma_{j,c} = \varnothing$$

which is equivalent to the condition that the local supervisor S_i controls all the controllable events that are local, i.e., $\Sigma_{i,c} = \Sigma_c \cap \Sigma_i$ [6]. Thus, we can conclude that $S_i = \tilde{S}_i$.

As mentioned in Remark 2, language separability introduced by Willner and Heymann [20] plays a key role in decentralized supervisory control of concurrent systems, and is formally defined as follows.

Definition 3: Given local event sets $\{\Sigma_i\}_{i\in I}$ and the global event set $\Sigma = \cup_{i\in I}\Sigma_i$, a language $K \subseteq \Sigma^*$ is separable (with respect to $\{\Sigma_i\}$) if for each $i \in I$ there exists $K_i \subseteq \Sigma_i^*$ such that $K = ||_{i\in I}K_i = \bigcap_{i\in I} P_i^{-1}(K_i)$.

The authors of [20] also pointed out that the verification of language separability can be performed by testing the local projection of the language, which states as follows.

Proposition 3: A language $K \subseteq \Sigma^*$ is separable with respect to $\{\Sigma_i\}$ if and only if $K = ||_{i\in I}P_i(K) = \bigcap_{i\in I} P_i^{-1}P_i(K)$.

According to Theorem 1 and Corollary 1, decentralized superviors exist if and only if the specification is prefix-closed, controllable and decomposable; note that for concurrent systems, $L(G) = ||\bigcap_{i\in I} P_i^{-1} L(G_i)$The following theorem provides a necessary and sufficient condition for the existence of the local supervisors S_i such that the collective behaviors of all the local plants satisfy the global specification.

Theorem 7: [20] [6]Consider the concurrent system $G = ||_{i\in I} G_i$. Let Σ be the global event set, and $\Sigma_i \subseteq \Sigma, i \in I$ be the local event sets. Let P_i be the natural projection from Σ to Σ_i, $\Sigma_{i,c} \subseteq \Sigma_i$ be the local controllable event set for the i-th agent, $\Sigma_c = \bigcup_{i\in I} \Sigma_{i,c}$ be the global controllable event set and $\Sigma_u c = \Sigma - \Sigma_c$ be the global uncontrollable event set. If $\Sigma_{i,c} = \Sigma_c \cap \Sigma_i$, then for a given non-empty, prefix-closed specification language $K \subseteq L(G)$, local supervisors $\{S_i, i \in I\}$ exist such that $\bigcap_{i\in I} P_i^{-1}(L(S_i||G_i)) = K$ if and only if K is Σ_{uc}-controllable and $(\{\Sigma_i\})$-separable. Moreover, if K is separable but not Σ_{uc}-controllable, then the local supervisors S_i can be synthesized such that

$$\begin{aligned} &||_{i\in I} L(S_i||G_i) \\ &\quad = ||_{i\in I} \sup C_{\Sigma_{i,uc}}(P_i(K)) \\ &\quad = \sup C_{\Sigma_{uc}}(K) \end{aligned} \tag{22}$$

We introduce the concept of modular controllability to characterize the properties of the specification required by Theorem 7:

Definition 4: Consider the local plant language $\{L_i \subseteq \Sigma_i^*\}_{i\in I}$ and the local uncontrollable events $\Sigma_{i,uc} \subseteq \Sigma_i$, a language $K \subseteq \Sigma^*$ is said to be *modularly controllable* if there exists $\{K_i \subseteq \Sigma_i^*\}$ such that $K = ||_{i\in I} K_i$ and K_i is locally controllable with respect to L_i and $\Sigma_{i,uc}$.

Based on Definition 4 and Theorem 7, it is clear that for a concurrent system $G = ||_{i\in I} G_i$ and a prefix-closed specification $K \subseteq L(G)$, the local supervisors that can achieve K exist if and only if K is modularly controllable.

From Lemma 3.2 in [20] along with Proposition 3, the following proposition claims the relationship between modular controllability and global controllability.

Proposition 4: If $K \subseteq L = ||_{i\in I} L_i$ is modularly controllable, then K is globally controllable with respect to L and Σ_{uc} and K is separable with respect to $\{\Sigma_i\}_{i\in I}$.

Remark 3: Proposition 4 states that, modular controllability is a sufficient condition for the combination of separability and global controllability, i.e., if a specification can be achieved by the joint work of the local supervisors, it can always be achieved by a centralized monolithic supervisor, as we expect.

Under the assumption that shared events have the same controllability status, modular controllability coincides with controllability and separability combined. This is stated as the following corollary:

Corollary 2: Consider $K \subseteq L = ||_{i\in I} L_i$, where $L_i \subseteq \Sigma_i^*$ and $\forall i, j \in I, i \neq j, \Sigma_{i,cu} \cap \Sigma_{j,c} = \varnothing$, then K is modularly controllable if and only if K is globally controllable with respect to L and Σ_{uc} and K is $\{\Sigma_i\}$ separable.

Theorem 7 and Corollary 2 provide shed light on our development of the algorithm for local synthesis of the supervisors. The following steps can help us to implement the local synthesis procedure.

- Local specification generation Given the non-empty,prefix-closed global specification language $K \subseteq L(G)$, let $K_i = P_i(K)$, as suggested by Lemma 3.2 in [20], $K_i \subseteq P_i(L(G)) \subseteq L(G_i)$.
- Check the separability of K Check whether or not $K = ||_{i\in I} K_i$, if so, then K is separable from Proposition 3, otherwise, find $K_i' \subseteq \Sigma_i^*$ such that $K = ||_{i\in I} K_i'$; if no such K_i' exists, then find $K' \subseteq K$ such that K' is separable. Set $K_i = K_i'$, $K' = K$.
- Local synthesis With the local specification K_i and the local controllability information $\Sigma_{i,uc}$, one can use Algorithm 2 to learn a centralized supervisor S_i such that $L(S_i||G_i) = \sup C(K_i)$.
- Return the solution Return S_i as the decentralized supervisors

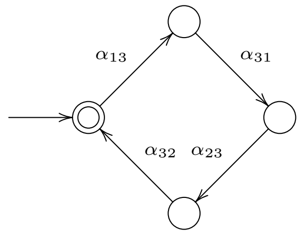
Since the complexity synthesizing a centralized monolithic supervisor S_i under complete observation by using Algorithm 2 is only polynomial with respect to the number of the states of S_i and the length of the counterexample queries as mentioned in Section III, we can conclude that the aforementioned synthesis procedure can be implemented in an on-line manner.

C. Illustrative example

We now apply the aforementioned local synthesis procedure for the motivating example proposed in Section II.

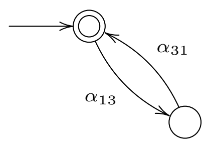
Example 4: (Motivating example revisited (cont'd)) For $i \in I = \{1, 2\}$, define α_{i3} and α_{3i} to be the events representing that Robot i is transporting from Rail i to Rail 3 and back from Rail 3 to Rail i, respectively. Then the local event sets are $\Sigma_1 = \{\alpha_{13}, \alpha_{31}\}$ and $\Sigma_2 = \{\alpha_{23}, \alpha_{32}\}$. Without loss of generality, we assume that all the local events are locally controllable, i.e., $\Sigma_{uc} = \Sigma_{1,uc} = \Sigma_{2,uc} = \varnothing$. From above, the specification for the two robots is given by $K = \overline{(\alpha_{13}\alpha_{31}\alpha_{23}\alpha_{32})^*}$, and a DFA that recognizes K is given as follows.

K :

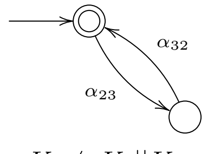


We start from Step 2, and obtain (initial) local specifications as follows: $K_1 = P_1(K) = \overline{(\alpha_{13}\alpha_{31})^*}$ and $K_2 = P_2(K) = \overline{(\alpha_{23}\alpha_{32})^*}$, which are depicted, respectively.

K_1 :



K_2 :



However, we find that $K \neq K_1||K_2$, which requires us to reconfigure K_1 and K_2.

Consider the two-robot scenario, we assume that each robot has adequate sensors to help detect the which rail the other robot is currently located, however, the two robots cannot control the behaviors of the other's. Accordingly, Σ_1 and Σ_2 are augmented to be $\Sigma_1 = \{\alpha_{13}, \alpha_{31}, \alpha_{32}\}$ and $Sigma_1 = \{\alpha_{23}, \alpha_{32}, \alpha_{31}\}$, respectively. Moreover, we set that $\Sigma_{1,uc} = \{\alpha_{32}\}$ and $\Sigma_{2,uc} = \{\alpha_{31}\}$. In this case, we obtain the local specifications to be $K_1 = \overline{(\alpha_{32}^*\alpha_{13}\alpha_{32}^*\alpha_{31}\alpha_{32})^*}$ and $K_2 = \overline{(\alpha_{31}\alpha_{23}\alpha_{31}^*\alpha_{32}\alpha_{31}^*)^*}$, in this case we find that $K = K_1||K_2$, which permits us to go one step further.

Now we apply Algorithm 2 in Section IV for local supervisor synthesis. It can be learnt by using membership queries (3) and (4) that both K_1 and K_2 are locally controllable with the corresponding local controllable events. Therefore, we can find the local supervisors S_1 and S_2 to help achieve the global specification K.

VII. CONCLUSIONS

In this paper, the decentralized supervisory control and synthesis problem with no prior knowledge of the plant is investigated. By using the modified membership queries, the L^* can learn the supremal controllabel and co-normal sublanguage of a given prefix-closed specification language, and an illustrative example is also provided to show the effectiveness of the proposed algorithm. Moreover, the local synthesis of decentralized supervisors is also investigated, and based on the property of modular controllability, one can learn the local supervisors by using only local information and the specification can also be achieved by the joint work of the decentralized supervisors. Future work will be focused on synthesis problems when the given specification is not prefix-closed or not modularly controllable.

REFERENCES

[1] D. Angluin,"Learning regular sets from queries and counterexamples," *Int. J. Inform. and Computation*, vol. 75, no. 1, pp. 87-106, 1987.

[2] C. G.Cassandras, S. Lafortune, *Introduction to Discrete Event Systems*, USA: Springer, 2008.

[3] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Valaiya, "Supervisory Control of Discrete-event Process with Partial Observation," *IEEE Trans. Autom. Control*, vol. 33, no. 3, pp. 249-260, 1988.

[4] M. Heymann and F. Lin, "On-line control of partially observed discrete event systems," *Discrete Event Dynamical Systems*, vol. 4, no. 3, pp. 221-236, 1994.

[5] S. Chung. S. Lafortune, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1921-1935, 1992.

[6] S. Jiang and R. Kumar, "Decentralized control of discrete event systems with specializations to local control and concurrent systems." *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*,vol. 30, no. 5, pp. 653-660, 2000.

[7] P. Kozak W. M. Wonham, "Fully decentralized solutions of supervisory control problems," *IEEE Trans. Autom. Control*, vol. 40, no. 12, pp. 2094-2097, 1995.

[8] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Boston: Kluwer, 1995.

[9] F. Lin, W. M. Wonham, "On observability of Discrete-event Systems," *Inform. Sci.*, vol. 44, pp. 173-198, 1988.

[10] F. Lin, "Robust and Adaptive Supervisory Control of Discrete Event Systens," *IEEE Trans. Autom. Control*, vol. 38, no. 12, pp. 1848-1852, 1993.

[11] —, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 1330C1337, Dec. 1990.

[12] F. Liu and H. Lin, "Reliable supervisory control for general architecture of decentralized discrete event systems," *Automatica*, vol. 46, no. 9, pp. 1510-1516, 2010.

[13] A. Overkamp and J. H. van Schuppen, "Maximal solutions in decentralized supervisory control," *SIAM J. Control Optim.*, vol. 39, no. 2, pp. 492-511, 2000.

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

[15] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. of the IEEE*, vol. 77, no. 1, pp. 81-98, 1989.

[16] R. L. Rivest and R. E. Schapire. "Inference of finite automata using homing sequences." *Machine Learning: From Theory to Applications*, Springer Berlin Heidelberg, pp. 51-73, 1993.

[17] K. Rohloff and S. Lafortune. "On the computational complexity of verification of modular discrete event systems," *Proc. IEEE Conference on Decision and Control*, Las Vegas, NV, 2002.

[18] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Autom. Control*, vol. 37, no. 11, pp. 1692C1708, Nov. 1992.

[19] Q. Wen, R. Kumar, J. Huang and H. Liu, "A Framework for Fault-Tolerant Control of Discrete Event Systems," *IEEE Tran. Autom. Control*, vol. 53, no. 8, pp. 1839-1849, 2008.

[20] Y. Willner and M. Heymann, "Supervisory control of concurrent discrete-event systems," *Int. J. Contr.*, vol. 54, no. 5, pp. 1143C1169, 1991.

[21] X. Yang, M. D. Lemmon, and P. Antsaklis, "Inductive inference of optimal controllers for uncertain logical discrete event systems," *Proc. American Control Conference*, Seattle, Jun. 1995, vol. 5, pp. 3163-3167.

[22] X. Yang, M. D. Lemmon, and P. Antsaklis, "Inductive inference of logical DES controllers using the L^* algorithm," *Proc. 1995 IEEE International Symposium on Intelligent Control*, pp. 585-590, 1995.