Control of Black-Box Embedded Systems by Integrating Automaton Learning and Supervisory Control Theory of Discrete-Event Systems

Huimin Zhang, Lei Feng, *Member, IEEE*, and Zhiwu Li, *Fellow, IEEE*

***Abstract*— The paper presents an approach to the control of black-box embedded systems by integrating automaton learning and supervisory control theory (SCT) of discrete-event systems (DES), where automaton models of both the system and requirements are unavailable or hard to obtain. First, the system is tested against the requirements. If all the requirements are satisfied, no supervisor is needed and the process terminates. Otherwise, a supervisor is synthesized to enforce the system to satisfy the requirements. To apply SCT and automaton learning technologies efficiently, the system is abstracted to be a finite-discrete model. Then, a C^* learning algorithm is proposed based on the classical L^* algorithm to infer a Moore automaton describing both the behavior of the system and the conjunctive behavior of the system and the requirements. Subsequently, a supervisor for the system is derived from the learned Moore automaton and patched on the system. Finally, the controlled system is tested again to check the correctness of the supervisor. If the requirements are still not satisfied, a larger Moore automaton is learned and a refined supervisor is synthesized. The whole process iterates until the requirements hold in the controlled system. The effectiveness of the proposed approach is manifested through two realistic case studies.**

***Note to Practitioners*—Supervisory control theory of DES can synthesize maximally permissive supervisory controllers to ensure the correctness of software-controlled processes. The application of supervisory control theory relies on automaton models of the plant and specifications; however, the required models are often unavailable and difficult to obtain for black-box embedded systems. Automaton learning is an effective method for inferring models of black-box systems. This paper integrates the two technologies so that the supervisory control theory is applicable to the development of black-box embedded software systems. The proposed approach is implemented in a toolchain that connects automaton learning algorithms, SCT, and testing algorithms via scripts. The obtained supervisor is implemented as a software patch to monitor and control the original system online.**

***Index Terms*— Automaton learning algorithm, black-box embedded system, software testing, supervisory control theory.**

I. Introduction

IN THE development of complicated embedded systems, third-party components or legacy subsystems are reused to reduce the development costs. The source code of the reused components is often unavailable. If some requirements of the new application are not satisfied in the reused component, it is impossible to amend the system by modifying its source code. Although fault identification methods [1], [2] can detect the violations, they cannot correct the errors online. An alternative approach is to use a "supervisor" component to monitor the reused component and correct it, if necessary. The supervisory component can be designed by the *supervisory control theory* (SCT) of *discrete-event systems* (DES) [3].

In SCT, both the plant and requirements are specified in formal models, such as *finite state automata* [3], [4] and *Petri nets* [5]–[7]; however, the formal models of the reused components are often unknown in realistic systems. Since the source code of the reused components is unavailable, it is difficult to obtain a logical model for the system by analyzing its source code [4]. This paper considers the reused component as a black-box system [8] of which only input and output sequences are observable. Moreover, user requirements are often described as textual description, unified modeling languages (UMLs), structured natural languages, logical formulas, and so on. It is elusive to build appropriate automaton models for textual requirements [4]. Due to the lack of formal models, SCT can hardly be used for the verification and control of black-box systems. Fortunately, *automaton learning algorithms* can infer automaton models for systems and check the validity of requirements from the input and output trajectories of the system [9]–[12]. Thanks to these learning methods, this

Manuscript received January 14, 2019; revised June 6, 2019; accepted July 13, 2019. This paper was recommended for publication by Associate Editor P. Chiacchio and Editor Y. Ding upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant 61873342, Grant 61472295, and Grant 61562015, in part by the National Key R&D Program of China under Grant 2018YFB1700104, in part by the Recruitment Program of Global Experts, and in part by the Macao Special Administration Region (MSAR) through the Science and Technology Development Fund under Grant 0012/2019/A1. The work of H. Zhang was supported in part by Guangxi Natural Science Foundation under Grant 2018GXNSFAA294052 and in part by Guangxi Science and Technology Planning Project under Grant AB18126063. The work of L. Feng was supported in part by KTH Excellence in Production Research (XPRES). *(Corresponding authors: Lei Feng; Zhiwu Li.)*

H. Zhang is with the College of Computer Science and Information Engineering and the College of Software, Guangxi Normal University, Guilin 541004, China, and also with the School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China (e-mail: hmzhang@stu.xidian.edu.cn).

L. Feng is with the Department of Machine Design, KTH Royal Institute of Technology, 10044 Stockholm, Sweden (e-mail: lfeng@kth.se).

Z. Li is with the Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China, and also with the School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China (e-mail: zhwli@xidian.edu.cn).

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

Digital Object Identifier 10.1109/TASE.2019.2929563

paper proposes an approach to synthesize a supervisor for a system by integrating automaton learning algorithms and SCT together to make requirements satisfiable when the automaton models of both the system and requirements are unavailable.

Studies integrating automata learning methods and SCT to synthesize supervisors for the plant are investigated in [13]–[20]. When the model of the system is confined to a finite look-ahead window [21], an optimal supervisor is learned by an adapted L^* learning algorithm [13], [14]. In the case that the plant model is completely unavailable, the L^* algorithm is modified to infer supervisors by observing uncontrollable illegal strings [15]–[17]. When the model of the system is known but that of the requirement is not, Hiraishi [18] proposes the K^* algorithm to construct a reduced supervisor for the system. We have been studying supervisor synthesis for black-box systems in recent years. In [19], a deterministic finite automaton (DFA) model is inferred for a black-box system by LBTest [22]. The requirements specified in linear temporal logic (LTL) are converted into automata by tools such as LTL2BA [23]. Finally, a supervisor is computed using SCT. In [20], an S^* learning algorithm is proposed to directly infer a supervisor for the system. If the learned supervisor is blocking, SCT is used to compute a nonblocking supervisor in a further step.

In SCT, the first step for computing a supervisor is to calculate the *synchronous product* of the automaton models of the plant and the requirement [3], [24]. Then, a supervisor is derived on the synchronous product automaton by removing blocking and uncontrollable states iteratively. Motivated by this, this paper proposes a learning algorithm to infer a Moore automaton describing both the behavior of the system and the conjunctive behavior of the system and requirements. The learned Moore automaton is used by an SCT algorithm to synthesize the supervisor. The new approach is more general than our previous method in [19], where the automaton models of the plant and the requirement are obtained separately. If the requirement cannot be described by a regular language but the intersection of the system and requirement can, the method in [19] cannot learn the automaton model of the requirement. Then a supervisor cannot be computed by SCT using the previous approach. On the contrary, the new approach in this paper can still find the supervisor if the intersection of the system and the requirement is regular.

Integrating automaton learning algorithms and SCT, this paper presents a novel approach to the supervisory control of black-box embedded systems, where the automaton models of both the systems and requirements are not directly available or hard to obtain. First, the system is tested against the requirements. If all the requirements are satisfied, the approach terminates. Otherwise, the system is abstracted to be discrete if its state space is continuous or extremely large. Subsequently, the C^* algorithm adapted from the L^* learning algorithm is proposed to construct a Moore automaton describing both the behavior of the system and the conjunction behavior of the system and requirements. Then, a supervisor for the system is computed based on the learned automaton by SCT. Next, the supervisor is implemented as a patch to monitor and control the original system. Then, the controlled system is tested again to check the reliability of the supervisor. The procedures of automaton learning, supervisor computing, and system testing are performed in an iterative manner until the requirement holds. The C^* algorithm is implemented based on LearnLib [25], [26]. Scripts connecting LearnLib, TCT [27], and LBTest [22] are developed to automate the process. This paper provides two main contributions.

1) An approach for the supervisory control of black-box embedded systems is presented, where both the automaton models of the system and requirements are not directly available or hard to obtain.
2) A C^* learning algorithm based on the L^* algorithm is proposed to infer a Moore automaton describing both the behavior of the system and the conjunction behavior of the system and requirements such that the supremal nonblocking supervisor of the problem can be synthesized from the learned automaton by SCT.

The organization of the remaining paper is as follows. Section II concisely describes the primary knowledge of automata and SCT and gives a rough sketch of the L^* learning algorithm. Section III states the proposed approach in detail. A small example is given in Section IV to illustrate the implementation detail of the proposed approach. Experimental studies on realistic systems are performed in Section V. Finally, the paper is concluded in Section VI.

II. Preliminaries

A. Basics of Automata

An *alphabet* Σ is a nonempty finite set of symbols. All finite strings over Σ including the empty string ϵ form Σ^*. Let $\sigma \in \Sigma$ and $s \in \Sigma^*$. The symbol $\#\sigma(s)$ denotes the number of occurrences of σ in s. A *language* L over Σ is a subset of Σ^*. The concatenation of strings w_1 and w_2 in Σ^* is represented by $w_1.w_2$ or w_1w_2 for shorthand. The concatenation of two languages L_1 and L_2 is $L_1.L_2 = \{l_1l_2 | l_1 \in L_1 \ \& \ l_2 \in L_2\}$. Given two strings s and s' in Σ^*, if there is a string u in Σ^* such that $s'.u = s$, s' is said to be a *prefix* of s. The *prefix closure* $\overline{L}$ of a language L consists of all the prefixes of all the strings in L. If $L = \overline{L}$, L is *prefix-closed*. If there exist strings u and s' in Σ^* such that $u.s' = s$, s' is a *suffix* of s. All the suffixes of all the strings in L form the *suffix closure* of L. If L is equivalent to its suffix closure, then L is *suffix-closed*.

A DFA $\mathbf{A}$ over an alphabet Σ is a quintuple

$$\mathbf{A} = (Q, \Sigma, \delta, q_0, Q_m)$$

where

Q	set of states;
Σ	alphabet;
δ: $Q \times \Sigma \to Q$	transition function;
$q_0 \in Q$	initial state;
$Q_m \subseteq Q$	set of marker states.

Function δ is considered to be partial in a DFA, i.e., for $\sigma \in \Sigma$, $\delta(q, \sigma)$ is not always defined. The extension of the function δ from domains $Q \times \Sigma$ to $Q \times \Sigma^*$ is defined recursively: $\delta(q, \varepsilon) = q$; $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$ if $\delta(q, s)$ and $\delta(\delta(q, s), \sigma)$ are defined for $\sigma \in \Sigma$, $q \in Q$ and $s \in \Sigma^*$.

$L(\mathbf{A}) = \{s \in \Sigma^* | \delta(q_0, s) \text{ is defined } \}$ and $L_m(\mathbf{A}) = \{s \in L(\mathbf{A}) | \delta(q_0, s) \in Q_m\}$ are the *closed* and *marked languages* of $\mathbf{A}$, respectively.

A Moore automaton $\mathbf{M}$ over an alphabet Σ is a six-tuple $\mathbf{M} = (Q, \Sigma, O, \delta, \lambda, q_0)$, where Q, Σ, and q_0 have the same meanings as in a DFA. The finite set O consists of output symbols. $\lambda : Q \to O$ represents the output function defined on states. The transition function δ and λ in a Moore automaton are completely defined. If $O = \{\text{true}, \text{false}\}$ and $\lambda(q) = \text{true} \leftrightarrow q \in Q_m$, the Moore automaton $\mathbf{M}$ is isomorphic to a DFA $\mathbf{A}$.

B. Supervisory Control Theory

SCT is a control technique to enforce a plant to satisfy its requirement in the DES framework [3], [28]. Assume that an uncontrolled DES, namely a plant, is modeled by a DFA $\mathbf{G}$ with an alphabet Σ, where Σ is partitioned into two disjoint subsets Σ_c and Σ_u, i.e., $\Sigma = \Sigma_c \bigcup \Sigma_u$. Set Σ_c consists of *controllable events* that can be disabled, and set Σ_u consists of *uncontrollable events* that cannot be prevented from happening. In SCT, the requirement that the plant must satisfy is modeled by a finite automaton $\mathbf{H}$, which has the same alphabet as $\mathbf{G}$.

The premise is that the behavior of $\mathbf{G}$ may violate the requirement and must be modified through feedback control. In order to alter the behavior of $\mathbf{G}$, a supervisor implements the control function $V : L(\mathbf{G}) \to \Gamma$ is introduced, where $\Gamma = \{\gamma \in 2^{\Sigma} | \Sigma_u \subseteq \gamma \subseteq \Sigma\}$ is the set of *control patterns*. The *controllable language* is a fundamental concept in SCT [28].

Definition 1: A language $K \subseteq \Sigma^*$ is *controllable* with respect to a prefix-closed language $L \subseteq \Sigma^*$ and an uncontrollable event subset $\Sigma_u \subseteq \Sigma$ if

$$(\forall s \in \Sigma^*, \sigma \in \Sigma_u)\, s \in \overline{K} \,\&\, s\sigma \in L \Rightarrow s\sigma \in \overline{K}.$$

Given a language $M \subseteq L$, the set of all sublanguages of M that are controllable with respect to L is

$$C(M, L) = \{K \subseteq M | K \text{ is controllable w.r.t } L\}.$$

The supremal element of $C(M, L)$ is $\sup C(M, L)$ [28].

C. Essential Description of the L^* Algorithm

The L^* algorithm is a classical *active automaton learning* algorithm for identifying unknown regular languages [9]. Suppose that a set U is the target language defined over an alphabet Σ. The L^* algorithm infers U by constructing a canonical DFA $\mathbf{A}$ such that $L_m(\mathbf{A}) = U$.

In the learning process, *membership queries* and *equivalence queries* are generated by the *learner*. The membership query asks whether a string $s \in \Sigma^*$ belongs to U and the equivalence query asks whether the learned language is equal to U. A *minimal adequate teacher* (MAT) is assumed to answer the two types of queries. An *observation table* (S, E, T) is the key structure of the learning algorithm, where the set S is nonempty prefix-closed, the set E is nonempty suffix-closed, and T is a function $T : (S \cup S.\Sigma).E \to \{0, 1\}$. $T(s) = 1$ if $s \in U$. Otherwise, $T(s) = 0$. row(s) is a vector denoting the row of the observation table corresponding to a string s in $(S \cup S.\Sigma)$. Initially, $S = E = \{\epsilon\}$.

Definition 2: The observation table is *closed* if for each $t \in S.\Sigma$, there is a string $s \in S$ such that row(t) = row(s).

Definition 3: The observation table is *consistent* if for any $s_1, s_2 \in S$, row$(s_1) = row(s_2)$ implies row$(s_1.\sigma)$ = row$(s_2.\sigma)$ for all $\sigma \in \Sigma$.

Definition 4: If the observation table (S, E, T) is closed and consistent, a DFA conjecture $\mathbf{A}(S, E, T) = (Q, \Sigma, \delta, q_0, Q_m)$ is defined where

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

After a conjecture is learned, an equivalence query is asked to check whether the conjecture equals to the target. If the answer is positive, the L^* algorithm terminates. Otherwise, a *counterexample* is identified by the MAT to refine the conjecture. All the prefixes of the counterexample are added to the set S to refine the conjecture. Then a new conjecture is constructed. The learning process iterates until no counterexample is found.

Let the number of states of the canonical DFA that accepts U be m and the upper bound on the length of any counterexample provided by the MAT be n. Angluin [9] proves that the total running time of the L^* algorithm is bounded by a polynomial with respect to m and n.

III. Learning and Control Approach

Given an embedded system and its requirements, our approach provides a complete solution including testing and supervisor synthesis to correct the system online. The approach is effective when the following assumptions hold.

1) The input and output of the system are observable.
2) The membership queries to the requirement can be answered by a computer program through analyzing input and output signals.
3) With proper abstraction, the system can be represented by a DFA.
4) The conjunction of the system and the requirement is representable by a regular language.

Fig. 1 illustrates the procedure of the proposed approach. The inputs of the approach are a black-box system and a requirement. First, the system is tested against the requirement. If the requirement is satisfied, the procedure terminates. Otherwise, the approach designs a supervisor to prevent the system online from reaching unacceptable states that violate the requirement. Since a realistic system may involve large or infinite domains of input and output, it is necessary to abstract the system to be discrete such that automaton learning algorithms and SCT can work effectively. When the automaton models of the system and requirement are unavailable, the C^* algorithm is proposed to infer a Moore automaton capturing the conjunction behavior of the system and the requirement. Then, a supervisor is computed from the learned automaton by SCT in the fourth step. Finally, the supervisor is supplemented to the system to ensure its

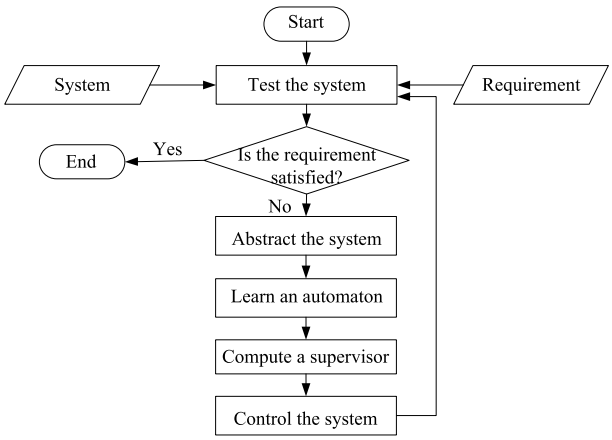

Fig. 1. Framework for the proposed approach.

correctness. To assure that the controlled system satisfies the requirement, the system is tested again. If the requirement is still not satisfied in the controlled system, a larger automaton is inferred in the next learning process. A new supervisor is computed and patched to the original system. The whole procedure iterates until the controlled system satisfies the requirement. Sections III-A–III-E elaborate on the technical details of these steps.

A. Testing the System

The embedded system is tested against the requirement. If the system satisfies the requirement, no supervisor is needed. Then, the approach terminates. There are plenty of technologies for testing black-box systems, such as boundary value analysis, equivalence partitioning, error guessing, and so on [29]. This paper applies two different testing methods to the two case studies. Section V-A applies learning-based testing [30] to a brake-by-wire (BBW) system and Section V-B applies random testing to a vehicle platooning program.

B. System Abstraction

The purpose of the system abstraction is to convert the continuous input and output domain as discrete ones. In embedded control systems, the input signals to the embedded control software are typical reference values determined by the human or other decision functions and sensor measurements. The output signals of the embedded control system are typically sensor measurements and system states actively sent out by the software. These signals are naturally observable.

Suppose a system with m input signals x_1, ..., x_m and n output signals y_1, ..., y_n, whose domains are X_i $(i = 1, \ldots, m)$ and Y_j $(j = 1, \ldots, n)$, respectively. Define the Cartesian products $X = X_1 \times \cdots \times X_m$ and $Y = Y_1 \times \cdots \times Y_n$. Evaluations of the input and output signals are represented by vectors $\mathbf{x} \in X \subseteq \mathbb{R}^m$ and $\mathbf{y} \in Y \subseteq \mathbb{R}^n$, where $\mathbb{R}$ is the set of real numbers.

In realistic systems, the domains of input and output may be continuous or finite but extremely large. Since the automaton learning algorithm and SCT are applicable only for finite-state problems, it is necessary to abstract the real input and output to discrete ones. To this end, two functions f_{in} and f_{out} are defined to abstract the concrete input and output to abstract ones, respectively.

We intend to obtain an abstract system such that the desired requirement can be analyzed while hiding irrelevant details. Generally, the abstraction of input and output of the embedded system is realized by partitioning the continuous or large finite domains into finite grids. Based on the description of the requirement, abstract output symbols can be first identified. Then, the output abstract function f_{out} is defined. Assume that there are p output symbols $\omega_1, \ldots, \omega_p$ considered in the requirement, of which the corresponding domains are Ω_1, ..., Ω_p. An evaluation of the abstract output is a vector $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_p]$ with domain $\Omega = \Omega_1 \times \cdots \times \Omega_p$. Let Σ be the set of all abstract input events.

Definition 5: The *output abstract function* of the system is $f_{\text{out}} : Y \to \Omega$, such that $f_{\text{out}}(\mathbf{y}) = [\omega_1, \ldots, \omega_p]$.

Definition 6: The *input refinement function* is defined as $f_{\text{in}} : \Sigma \to X$, such that $f_{\text{in}}(\sigma) = [x_1, \ldots, x_m]$.

The input refinement function is a dual of the output abstract function, because it converts an abstract input event to a concrete evaluation of the input vector $\boldsymbol{x}$ and sends the vector to the black-box system.

Converting continuous input and output signals into abstract input and output events is both important and challenging for finding an approximate discrete model of the real system. A bad selection of the output abstract function and input refinement function misses critical information of the system. The decision in practice requires a significant amount of prior knowledge about the system and iterative tests. Normally, the finer is the discretization step, the larger is the conjecture model. A guideline of selecting proper discretization resolution is to use the lowest resolution that allows the user to synthesize an effective supervisor for the system. In the future, we shall also study more advanced automaton learning methods to reduce the complexity of the conjecture model.

C. Learning Moore automata

Suppose that the marked and closed languages of the system are L and $\overline{L}$, respectively. Let H be the prefix-closed language of the requirement. In SCT, the nonblocking supremal supervisor is calculated by $\sup C(L \cap H, \overline{L})$. Therefore, it is necessary to determine the sets $L \cap H$ and $\overline{L}$ to compute the supervisor for the system. To this end, the set Σ^* is partitioned into three disjoint sets.

1) $L \cap H$ represents the marked system behaviors that also satisfy the requirement.
2) $\overline{L} - (L \cap H)$ represents all system behaviors that do not belong to the first subset.
3) $\Sigma^* - \overline{L}$ represents all behaviors that are not acceptable by the system.

The C^* algorithm is proposed to infer a Moore automaton that distinguishes the foregoing three types of languages.

1) Membership Query: The L^* algorithm relies on a MAT to correctly answer membership queries. This paper develops a software program to answer these queries. The program

sends the event sequences to the input of the black-box system, monitors the corresponding output sequences, and then determines the validity of the input event sequences. Formally, a membership query function is defined to answer membership queries.

Definition 7: The *membership query function* is defined as $f : \Sigma^* \to \{0, 1, 2\}$, where

$$f(s) = \begin{cases} 2, & s \vDash L \text{ and } s \vDash H \\ 1, & s \vDash \overline{L} \text{ but } f(s) \neq 2 \\ 0, & s \nvDash \overline{L}. \end{cases}$$

Let $L_i = \{s \in \Sigma^* | f(s) = i\}$ for $i = 0, 1, 2$. Evidently, $L_2 = L \cap H$ and $L_1 \cup L_2 = \overline{L}$. It is trivial to prove the following proposition.

Proposition 1: $\sup C(L \cap H, \overline{L}) = \sup C(L_2, L_1 \cup L_2)$.

2) Observation Table: Answers of membership queries are maintained in the observation table (S, E, T) during the learning process. In the table (S, E, T), $T(s.e)$ is the entry value of row s in $(S \cup S.\Sigma)$ and column e in E, where

$$T(s.e) = f(s.e). \tag{1}$$

For any $s \in (S \cup S.\Sigma)$, row(s) is a row vector of length $|E|$, i.e., row$(s) \in \{0, 1, 2\}^{|E|}$. For strings s and t in $S \cup S.\Sigma$, we denote row$(s) =$ row(t) if $(\forall e \in E)T(s.e) = T(t.e)$.

Definition 8: Suppose that the observation table (S, E, T) is both closed and consistent. A conjecture $\mathbf{M}(S, E, T)$ is defined as a Moore automaton $\mathbf{M} = (Q, \Sigma, O, \delta, \lambda, q_0)$, where

$$\begin{aligned} Q &= \{\text{row}(s) | s \in S\}; \\ O &= \{0, 1, 2\}; \\ q_0 &= \text{row}(\epsilon); \\ \delta(\text{row}(s), \sigma) &= \text{row}(s\sigma); \\ \lambda(\text{row}(s)) &= T(s). \end{aligned}$$

Proposition 2: The Moore automaton $\mathbf{M}(S, E, T)$ defined in Definition 8 is closed in Q and deterministic.

The proof of this proposition is similar to that of [20, Proposition 4].

Proposition 3: The Moore automaton $\mathbf{M}(S, E, T)$ defined in Definition 8 is minimal.

The proof of this proposition is similar to that of [20, Proposition 6].

3) Equivalence Checking: Let the learned automaton be $\mathbf{M} = (Q, \Sigma, O, \delta, \lambda, q_0)$. Equivalence queries ask whether the conjecture $\mathbf{M}$ is isomorphic to the target automaton model. In the L^* algorithm, a MAT is assumed to answer equivalence queries. In practice, however, the MAT is unavailable, because the target automaton model is unknown. It is impossible to directly check the isomorphism of $\mathbf{M}$ and the target structurally. We perform equivalence checking by verifying the behavioral equivalence between $\mathbf{M}$ and the target. The behavior of the target is described by the membership query function f defined in Definition 7. The procedure for equivalence checking is illustrated in Fig. 2.

Definition 9: If there is a string s in Σ^* such that $f(s) \neq \lambda(\delta(q_0, s))$, s is a *counterexample* between $\mathbf{M}$ and the system.

The challenge of the procedure in Fig. 2 is how to efficiently generate test strings and when to terminate the procedure. If a counterexample is found, the checking procedure terminates; if no counterexample is found after checking many test strings, one cannot decide whether the conjecture is indeed equivalent to the target or more test strings should be examined. The difficulty is similar to increasing the test coverage in software testing.

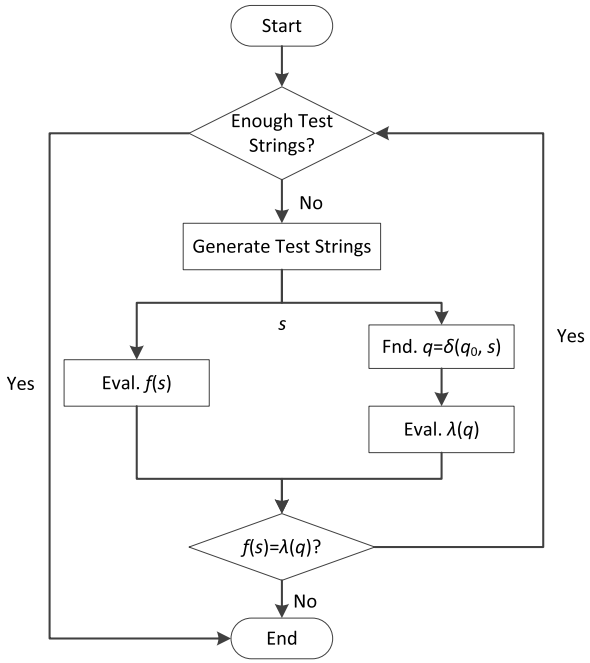

Fig. 2. Procedure for equivalence checking.

In the field of software testing, methods such as W-method [31], Wp-method [32], complete exploration, random testing, etc., are usually used to generate test strings for finite-state automata. The selection of these methods depends upon the specific systems to be learned. In Section V-A, the length of counterexamples found in the learning process of the BBW system increases gradually. Hence, the Wp-method is adopted. In Section V-B, however, the length of counterexamples of the platooning program is relatively long. Hence random testing is more suitable.

Random testing method generates random test strings in Σ^* of length between 0 and an upper bound. The test process stops if either a counterexample is found or a time upper bound is reached. The Wp-method generates test strings as follows. Assume that the numbers of states of the conjecture $\mathbf{M}$ and the target automaton are m and n, respectively. Let $V = \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \cdots \cup \Sigma^{(n-m)}$. Denote P, U and W as the *transition cover set*, *state cover set*, and *characterization set* of $\mathbf{M}$, respectively, whose detailed definitions are given in [32]. The state cover set is a subset of the transition cover set, i.e., $U \subseteq P$. At every reachable state $q_i \in Q$, the Wp-method defines the *identification set* $W(q_i) \subseteq W$ [32].

The Wp-method consists of two steps. Step 1 tests all strings in the set $U.V.W$. If no counterexample is found, step 2 continues to test all strings in the set

$$\{\text{pvw} \in \Sigma^* | p \in P - U, q_i = \delta(q_0, p), v \in V, \ w \in W(q_i)\}.$$

Under the assumption that the state number n of the target automaton can be correctly estimated, Fujiwara et al. [32] prove that the conjecture $\mathbf{M}$ is isomorphic to the target if there is no counterexample found by the Wp-method. In practice, however, the state number of the black-box software system can hardly be estimated. Equivalence query is essentially software testing and hence cannot guarantee the equivalence between the learned conjecture and the target even though no counterexample is found. This is an inherent limit of the learning-based method.

4) C^* Learning Algorithm: Suppose that the conjunctive behaviors of the black box system and the requirements can be represented by a Moore automaton $\mathbf{M}_c = (Q, \Sigma, O, \delta, \lambda, q_0)$, where $L_i = \{s \in \Sigma^* | \lambda(\delta(q_0, s)) = i\}$ for $i = 0, 1, 2$. We call the Moore automaton the *target* of the learning algorithm. To infer the target Moore automaton $\mathbf{M}_c$, this paper proposes the C^* learning algorithm in Algorithm 1, which is adapted from the classical L^* algorithm [9]. The differences between the L^* and C^* algorithms are illustrated as follows.

1) A DFA is learned by the L^* algorithm, but a Moore automaton with output set $O = \{0, 1, 2\}$ is constructed by the C^* algorithm.
2) The L^* algorithm assumes that an MAT answers membership queries, but our C^* algorithm checks whether the queried strings are acceptable by the system and the requirements via executing the system.
3) The L^* algorithm assumes that the MAT always correctly answers the equivalence queries, but our C^* algorithm applies software testing technologies (e.g., random testing and Wp-method) to check the equivalence between the conjecture automaton and the real system.

Algorithm 1: C^* Learning Algorithm

Input: An alphabet Σ and an estimated upper bound of the number of states of the target $\mathbf{M}_c$
Output: A Moore automaton $\mathbf{M}$

```
1  Let S = {ε} and E = {ε};
2  Construct the initial observation table (S, E, T) by
   performing f for ε and all σ ∈ Σ;
3  repeat
4      while (S, E, T) is not consistent or not closed do
5          if (S, E, T) is not consistent then
6              Find s, t ∈ S, e ∈ E and σ ∈ Σ such that
               row(s) = row(t), and T(sσe) ≠ T(tσe);
7              Add σ.e to E;
8          end
9          if (S, E, T) is not closed then
10             Find s ∈ S and σ ∈ Σ such that row(s.σ) is
               different from row(t) for all t ∈ S;
11             Add s.σ to S;
12         end
13         Filled the empty entries in (S, E, T) by performing
           f;
14     end
15     Construct a conjecture M(S, E, T) by Def. 8;
16     Perform equivalence checking between M and M_c;
17     if t is a counterexample then
18         Add all the prefixes of t to S;
19         Filled the empty entries in (S, E, T) by performing
           f;
20     end
21 until No counterexample exists;
22 Output M;
```

Algorithm 1 starts by assigning $S = E = \{\epsilon\}$. Then, the initial observation table (S, E, T) is constructed by executing the system and evaluating the membership query function f. Subsequently, both consistent and closed properties of (S, E, T), as in Definitions. 2 and 3, are checked in lines 4 to 14. If (S, E, T) is not consistent, it is modified to be consistent in lines 5–8. If (S, E, T) is not closed, it is modified to be closed in lines 9 to 12. When (S, E, T) is both consistent and closed, a conjecture $\mathbf{M}$ is constructed by Definition 8 at line 15. At line 16, a testing method is called to find a counterexample between $\mathbf{M}$ and the target. If t is a counterexample, all its prefixes are added to the set S. The observation table is enlarged, and the conjecture is refined. If no counterexample exists, the algorithm returns the conjecture $\mathbf{M}$ and terminates.

Proposition 4: The C^* algorithm terminates.

The proof of this proposition is similar to that of [20, Proposition 7].

Assume that the number of states of the target $\mathbf{M}_c$ is n and the maximum length of all the counterexamples provided by the equivalence checking procedure is m. In Algorithm 1, both S and E have one element ϵ initially. There is at least one state in the initial conjecture. If the observation table is not consistent, one element is added to the set E. If the observation table is not closed, one element is added to the set S. After the observation table becomes consistent and closed again, the number of states of the conjecture increases at least by 1. Therefore, the number for checking the consistency and closed properties of the C^* algorithm is at most $n-1$. There are $n-1$ counterexamples at most. For each counterexample, at most m strings are added to the set S. Therefore, the complexity of the C^* is bounded by a polynomial function of m and n.

An example is given in Section IV-A for illustrating the C^* algorithm.

D. Computing Supervisors

Let $\mathbf{M} = (Q, \Sigma, O, \delta, \lambda, q_0)$ be a Moore automaton constructed by the C^* algorithm. The state set Q is partitioned as three disjoint sets Q_0, Q_1, and Q_2, where $Q_i = \{q \in Q | \lambda(q) = i\}$, $i = 0, 1, 2$. Evidently, the set Q_0 contains all dump states.

Definition 10: The automaton $\mathbf{M}' = (Q', \Sigma, \delta', q_0', Q_m')$ is a DFA derived from the Moore automaton $\mathbf{M}$, where

$Q' = Q - Q_0$;
$\delta'(q, \sigma) = \delta(q, \sigma)$ if $\delta(q, \sigma) \notin Q_0$ for $q \in Q'$ and $\sigma \in \Sigma$; otherwise, $\delta'(q, \sigma)$ is not defined;
$q_0' = q_0$ if $q_0 \in Q'$;
$Q_m' = Q_2$.

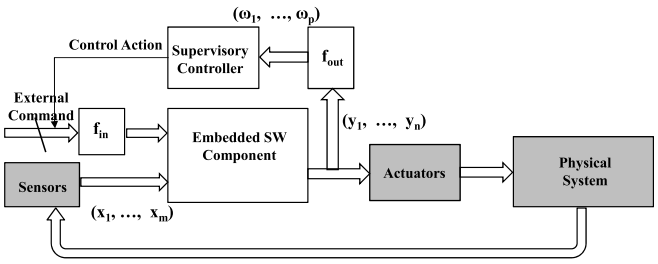

Fig. 3. Feedback loop of the supervisor.

In Definition 10, if $q_0 \notin Q'$, then $q_0 \in Q_0$ and $\epsilon \notin \overline{L}$. The closed language $\overline{L}$ of the system is empty, which is rare for real systems. The languages $L(\mathbf{M}')$ and $L_m(\mathbf{M}')$ describe the closed behavior of the system and the conjunction of the marked behavior of the system and the requirements, respectively.

Theorem 1: If the conjecture Moore automaton is isomorphic to the target, i.e., $\mathbf{M} \cong \mathbf{M}_c$, then $\sup C(L_m(\mathbf{M}'), L(\mathbf{M}')) = \sup C(L \cap H, \overline{L})$.

Proof: If $\mathbf{M} \cong \mathbf{M}_c$, $L_m(\mathbf{M}') = L_2$ and $L(\mathbf{M}')) = L_1 \cup L_2$. Therefore, $\sup C(L_m(\mathbf{M}'), L(\mathbf{M}')) = \sup C(L \cap H, \overline{L})$ holds by Proposition 1. ∎

An illustrative example for computing supervisors is presented in Section IV-B.

E. Supervisor Implementation

The supervisor constructed in Section III-D is implemented as a patch to force the behavior of the system to follow the requirement. Fig. 3 shows the detailed implementation of the feedback control. The entities of the physical world (e.g., the system, actuators, and sensors) are denoted by gray blocks. There are two types of input signals transmitting to the embedded software (SW) components. The first type comes from the sensor measurement of the physical system, which is uncontrollable. The other one is the control commands from the embedded controller, which may be uncontrollable or controllable. The supervisor implements the control function by overriding the control commands from the embedded control software. The whole control mechanism works in a feedback loop. As a result, the supervisor enforces the behavior of the system to act as desired.

The combination of the original system and the supervisor is regarded as a new system. The procedure iterates to Section III-A to test if the requirement is satisfied. If the requirement is still not satisfied, a larger conjecture is inferred and a new supervisor is calculated. The processes of automaton learning, supervisor computing, and system testing iterate until the requirement is satisfied.

IV. Illustrative Example

The proposed approach is illustrated by a simple example from [4], assuming that the automaton models of the plant and requirements are not available. The system contains two identical machines, which repeatedly perform the cycle of $a_i b_i$, $i = 1, 2$. The system is at a marker state when both machines have identical numbers of a_i and b_i. The alphabet of the system is $\Sigma = \{a_1, b_1, a_2, b_2\}$.

Requirement: After an occurrence of event b_1, b_1 shall not occur again until event b_2 occurs at least once.

Formally, a string $s \in \Sigma^*$ satisfies the requirement if either $\#b_1(s) \leq 1$ or

$$(\forall u, v, w \in \Sigma^*) s = ub_1vb_1w \wedge \#b_1(v) = 0 \Rightarrow \#b_2(v) \geq 1.$$

It can be easily checked that the system does not satisfy the requirement. Since the alphabet of the system is already discrete, we do not need to abstract it.

TABLE I
OBSERVATION TABLE I

T_1	ϵ
ϵ	2
a_1	1
a_2	1
b_1	0
b_2	0

TABLE II
OBSERVATION TABLE II

T_2	ϵ
ϵ	2
a_1	1
b_1	0
a_2	1
a_1a_2	1
a_1b_1	2

A. Learning Moore Automata

The first step is to construct the target Moore automaton with respect to the system and requirement by the proposed C^* learning algorithm. In the initial observation table (S_1, E_1, T_1), $S_1 = E_1 = \{\epsilon\}$ and $S_1.\Sigma = \{a_1, b_1, a_2, b_2\}$. The entries $T_1(s.e)$ in the observation table are filled by evaluating the membership query function $f(s.e)$. The initial observation table is shown in Table I.

In Table I, since $\text{row}(a_1) = 1$ and $\text{row}(b_1) = 0$ are not equivalent to the only row of set S_1, T_1 is not closed. Then, S_1 is enlarged to $S_2 = \{\epsilon, a_1, b_1\}$ and then $S_2.\Sigma - S_2 = \{a_2, b_2, a_1.a_1, a_1.a_2, a_1.b_1, a_1.b_2, b_1.a_1, b_1.a_2, b_1.b_1, b_1.b_2\}$. The updated observation table is shown in Table II. For brevity, if $s \in S_2.\Sigma - S_2$ and $row(s)$ is a zero vector, then the row is omitted. Table II is both closed and consistent. Correspondingly, a conjecture $\mathbf{M}_1 = (Q^1, \Sigma, O^1, \delta^1, \lambda^1, q_0^1)$ shown in Fig. 4 is constructed by Definition 8. Each row of a string in S_2 of the observation table denotes a state of the conjecture. The outputs of the states are labeled in the circles. If two strings in S_2 have the same row vector, they represent the same state in Q^1. If the row of a string is the zero vector, such as b_1 in T_2, the state denoted by the string is a dump state. For simplicity, the dump states of all automaton models in this paper are not shown in the pertinent figures.

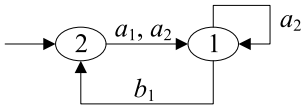

Fig. 4. Conjecture $\mathbf{M}_1$.

TABLE III
OBSERVATION TABLE III

T_3	ϵ
ϵ	2
a_1	1
b_1	0
a_1a_2	1
$a_1a_2b_1$	1
$a_1a_2b_1b_2$	2
a_2	1
a_1b_1	2
$a_1a_2b_2$	1
$a_1a_2b_1a_1$	1
$a_1a_2b_1b_2a_1$	1
$a_1a_2b_1b_2a_2$	1

TABLE IV
OBSERVATION TABLE IV

T_4	ϵ	a_2	b_1
ϵ	2	1	0
a_1	1	1	2
b_1	0	0	0
a_1a_2	1	0	1
$a_1a_2b_1$	1	0	0
$a_1a_2b_1b_2$	2	1	0
a_2	1	0	0
a_1b_1	2	1	0
$a_1a_2b_2$	1	1	2
$a_1a_2b_1a_1$	1	0	1
$a_1a_2b_1b_2a_1$	1	1	2
$a_1a_2b_1b_2a_2$	1	0	0

To evaluate whether the automaton shown in Fig. 4 is equivalent to the system, an equivalence checking is performed by random testing. Since $f(a_1a_2b_1b_2) = 2$, namely, the string is accepted by both the system and the requirement, but $\lambda^1(a_1a_2b_1b_2) = 0$, string $a_1a_2b_1b_2$ is a counterexample. All prefixes of $a_1a_2b_1b_2$ are added to S_2 of the observation table T_2. Then $S_3 = \{\epsilon, a_1, b_1, a_1a_2, a_1a_2b_1, a_1a_2b_1b_2\}$. The updated observation table T_3 is shown in Table III.

In Table III, $\text{row}(a_1) = \text{row}(a_1a_2)$ but $T_3(a_1a_2) \neq T_3(a_1a_2a_2)$, and $\text{row}(a_1a_2) = \text{row}(a_1a_2b_1)$ but $T_3(a_1a_2b_1) \neq T_3(a_1a_2b_1b_1)$, Table III is not consistent. Then strings a_2 and b_1 are added to the set E_3. The updated observation table T_4 has $S_4 = S_3$ and $E_4 = \{\epsilon, a_2, b_1\}$, as shown in Table IV. The table is also closed. The conjecture $\mathbf{M}_2 = (Q^2, \Sigma, O^2, \delta^2, \lambda^2, q_0^2)$ shown in Fig. 5 is constructed correspondingly. For clarity, the state is labeled by the row vector in T_4. The output of each state is represented by the first number of the vector at the state.

The equivalence query by random testing finds a counterexample $c = a_1b_1a_1b_1a_1a_2b_1b_2$. In Moore automaton $\mathbf{M}_2$, the string c reaches the initial state and the corresponding output is 2, namely, $\lambda^2(c) = 2$; however, the string c violates the requirement because event b_1 occurs three times before event b_2 occurs. Therefore, $f(c) = 1$. Adding all prefixes of c into set S_4, the set is enlarged to $S_5 = S_4 \cup \overline{\{c\}}$. There is no change on the column set: $E_5 = E_4$. From the observation table (S_5, E_5, T_5), we repeat the steps described in Algorithm 1 until no more counterexamples are found. The final Moore automaton model is shown in Fig. 6 with the dump state omitted. Then, DFA $\mathbf{M}'$ is derived from the Moore automaton $\mathbf{M}_3$ by Definition 10, which has the same state transition function as $\mathbf{M}_3$ and has two marker states illustrated in Fig. 6.

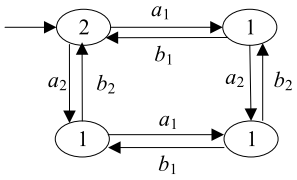

Fig. 5. Conjecture $\mathbf{M}_2$.

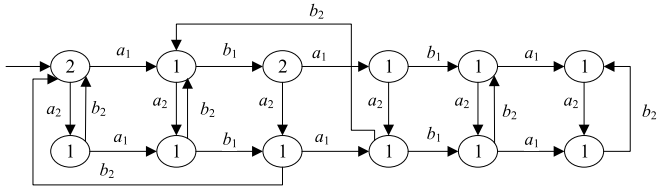

Fig. 6. Conjecture $\mathbf{M}_3$.

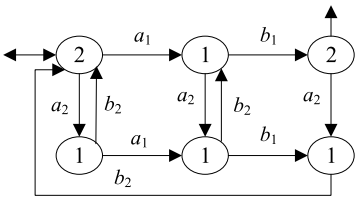

Fig. 7. Suppremal nonblocking supervisor.

B. Supervisor Computing

The supremal nonblocking supervisor can be derived from the DFA $\mathbf{M}'$ based on Theorem 1 and the standard supervisor synthesis algorithm. If we take the unobservable event set as $\Sigma_u = \{b_1, b_2\}$, the supremal nonblocking supervisor is shown in Fig. 7. One can easily verify that the supervisor is isomorphic to the one obtained directly from the given DFA models of the plant and the requirement.

C. Supervisor Implementation

The supervisor is developed as an executed program according to the logical structure shown in Fig. 7. Then, the supervisor runs synchronously with the system to implement control actions on line.

V. EXPERIMENTAL STUDY

A BBW system [33] and a platooning program [44] are studied on a personal laptop running Windows 10 with

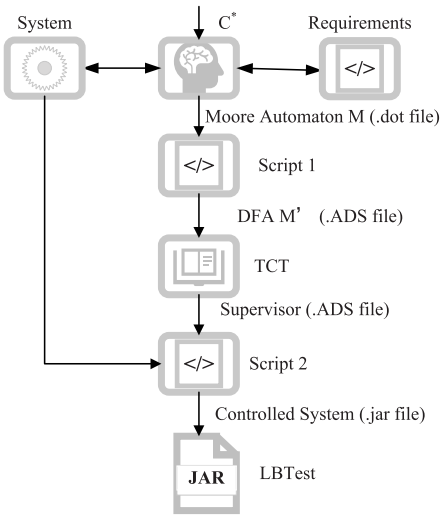

Fig. 8. Toolchain of the proposed approach.

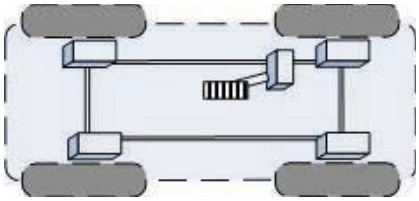

Fig. 9. Hardware model of the BBW system.

8-GB RAM and an Intel Core i7 4712MQ CPU at 2.30 GHz to illustrate the feasibility of the proposed approach on realistic systems. The C^* algorithm is implemented based on the LearnLib framework. A Java program is developed for checking satisfiability of requirements. LBTest [22] is used for testing requirements and TCT [27] is used for computing supervisors. The toolchain of the proposed approach is implemented with some scripts connecting LearnLib, TCT, and LBTest together as depicted in Fig. 8. The output format of the learned conjecture is a dot file. The format of the input and output files of TCT is either the text (.ADS) file or the binary (.DES) file. The supervisor is used to control the realistic systems. Scripts 1 and 2 are developed to transform dot files to ADS files and ADS to jar files, respectively.

A. BBW System

The Brake-by-Wire (BBW) system is a hard real-time controller for automobiles [33]. Recently, many researchers explore the BBW as case studies for code validation [34], resource usage [35], requirements verification [36] and testing [33]. In this paper, the BBW system is an executable jar file developed by Feng et al. [33] and the requirements are described as natural languages. Automaton models of both the system and requirements are unavailable.

Fig. 9 describes the hardware model of the BBW. Cubes denote the five electronic control units (ECUs). The four wheels are represented by gray round-corner rectangles. The gas and brake pedals are connected with the central ECU, while the other four wheels are attached to the other four ECUs. The input of the system is the position percentages of the gas and brake pedals. The software on the central ECU reads the input, calculates the global drive/brake torque, and distributes the torque to the four wheels. Then, the software on the four-wheel ECUs carries out the anti-lock braking system (ABS) function, which releases the brake actuator when the wheel is slipping.

1) Abstracting and Testing the System: The domains of the input and output of the BBW system are continuous. It is necessary to abstract the continuous system to a discrete one such that the proposed approach can be applied effectively. To distinguish concrete and abstract signals, the first letter of a concrete output signal is the lower case but the first letter of the abstract output is capital. The values of the abstract output are lower case.

Three requirements considered in [33] are described as follows.

Requirement 1: If the brake pedal is pressed and the wheel speed is greater than zero, the value of brake torque applied to the wheel by the corresponding ABS component shall eventually be greater than 0.

Requirement 2: If the brake pedal is pressed, the actual speed of the vehicle is larger than 10 km/h and a wheel is slipping, the corresponding brake torque at the wheel shall be zero.

Requirement 3: If both the brake and gas pedals are pressed, the actual vehicle speed shall be decreased.

For simplicity, the safety of the rear right wheel is studied as an example, and functions f_{in} and f_{out} with respect to Requirement 2 are introduced in detail. The output signals of the concrete system is: vehicle speed (*veh_Speed*), rotational speed of the rear right wheel (*speed_RR*), and torque value on the rear right wheel (*torque_RR*). Requirement 2 concerns three abstract output signals: vehicle status (*VehStatus*), brake torque on the wheel (TorqueRR), and wheel shipping status (SlipRR). Correspondingly, three output abstract functions f_{out_1}, f_{out_2}, and f_{out_3} are defined. Let $\mathbf{y}$ represent the concrete output vector.

The range of the symbol *VehStatus* is $\{\text{moving}, \text{still}\}$. The component output abstract function of *VehStatus* is

$$f_{\text{out}_1}(\mathbf{y}) = \begin{cases} \text{still}, & \textit{veh\_Speed} \leq 10 \\ \text{moving}, & \textit{veh\_Speed} > 10. \end{cases} \tag{2}$$

The domain of the symbol TorqueRR is $\{\text{nonzero}, \text{zero}\}$. The component output abstract function of TorqueRR is

$$f_{\text{out}_2}(\mathbf{y}) = \begin{cases} \text{zero}, & \text{torque\_RR} = 0 \\ \text{nonzero}, & \text{torque\_RR} > 0. \end{cases} \tag{3}$$

The range of abstract output SlipRR is $\{\text{slip}, \text{noslip}\}$. Let $P = 2 \times \text{veh\_Speed}/3.6$ and $T = 10 \times (\text{veh\_Speed}/3.6 - \text{wSpeed\_RR} \times \text{whl\_Radius})$, where *w_SpeedRR* and *whl_Radius* are angular velocity of the wheel and wheel radius. The component output abstract function of SlipRR is

$$f_{\text{out}_3}(\mathbf{y}) = \begin{cases} \text{noslip}, & \textit{veh\_Speed} < 10 \text{ or } T <= P \\ \text{slip}, & \textit{veh\_Speed} \geq 10 \text{ and } T > P. \end{cases} \tag{4}$$

The input of the concrete system is the position percentages of the brake and gas pedals in the range [0,100], denoted

by [*braPos*, *gasPos*]. The abstract alphabet of the input is $\Sigma = \{\text{brake}, \text{acc}, \text{idle}\}$. The input refinement function is

$$f_{in}(\sigma) = \begin{cases} [100, 0], & \sigma = \text{brake} \\ [0, 100], & \sigma = \text{acc} \\ [0, 0], & \sigma = \text{idle}. \end{cases} \quad (5)$$

LBTest is a software testing tool integrating learning-based testing methods with model checking technology and can check both safety and liveness properties [22]. Requirements 1–3 are tested by LBTest as presented in [33]. When testing Requirements 1–3, no counterexample appears after performing 300 testing iterations. When Requirement 2 is tested, a violation appears at the fifth testing round. The counterexample is "*acc,acc,acc,acc,brake,brake*." The corresponding sequence of abstract outputs is "[*still, zero, noslip*], [*still, zero, noslip*], [*moving, zero, noslip*], [*moving, zero, noslip*], [*moving, zero, noslip*], [*moving, zero, slip*], and [*still, nonzero, slip*]." The counterexample shows that a violation appears at the second *brake* event. When the vehicle is moving and the wheel is slipping, the brake torque at the wheel is *nonzero*.

2) Inferring Moore Automata: The C^* algorithm is used to infer the target Moore automaton with respect to the system and Requirement 2 so that a supervisor can be synthesized to make the system satisfy Requirement 2. The Wp-method introduced in Section III-C3 is used for the equivalence checking. The correctness and efficiency of the Wp-method rely on the accurate estimate of the state size n of the true automaton model of the black-box system. In real applications, however, the estimation value is often hard to obtain. If the estimate is smaller than the real value, the Wp-method may not find the counterexample and the conjecture is wrong. If the estimate is too large, the Wp-method may not terminate within an acceptable period of time.

We propose an iterative approach to estimate n on the basis of the state size of the conjecture automaton [20]. At the ith iteration of the C^* algorithm, if the state size of the $(i-1)$th conjecture automaton is m_{i-1}, then the estimated state size of the target automaton is $\hat{n}_i = m_{i-1} + l$, where l is a nonnegative integer. Thus, the set V required by the Wp-method becomes $V = \Sigma^0 \cup \Sigma \cup \cdots \cup \Sigma^l$. The following experiments reveal that if a counterexample exists, it can be found with a small value of l.

Recall Wp-method in Section III-C3. A test string $s \in \Sigma^*$ in both the steps consists of three substrings u, v, and w: $s = uvw$, where $u \in U$ (step 1) or $P - U$ (step 2), $v \in V$, and $w \in W$ (step 1) or $W(q_i)$ (step 2). We call them "prefix string," "middle string," and "suffix string," respectively. The lengths of the prefix and suffix strings are determined by the conjecture automaton and hence irrelevant to the state estimate $\hat{n}_i$. According to the definition of V, the maximal length of the middle string is dependent on $\hat{n}_i$ or l. Fig. 10 shows the lengths of the three substrings of counterexamples during the learning iterations. The lengths of the middle strings almost remain constant 1 except being 5 at the first iteration. Therefore, we take $l = 5$ for the Wp-method applied to the BBW system.

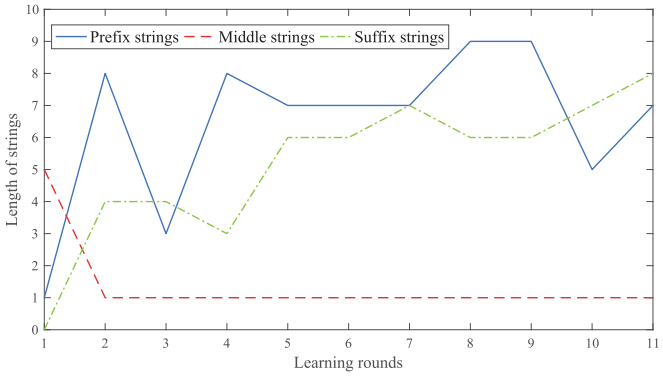

Fig. 10. Lengths of prefix, middle, and suffix strings of counterexamples.

TABLE V
SIZE OF THE LEARNED AUTOMATA AND SUPERVISORS AND TIME CONSUMPTION

Iteration (i)	1	2	3	4	5	6	7	8	9
#State(M)	1	19	22	36	63	222	348	401	663
#Trans(M)	3	57	66	108	189	666	1044	1203	1989
Learn. Time(s)	0	1.3	2.4	5.6	13.8	85.8	208.3	307.2	649.0
Equv. Time(s)	0	2.1	2.1	2.2	3.1	4.7	8.4	12.1	22.8
#State(S)	0	18	21	35	62	221	347	400	660
#Trans(S)	0	46	55	95	172	620	976	1119	1866

To obtain a reliable supervisor, a series of Moore automata approximating the target with respect to the BBW system and Requirement 2 are constructed by increasing the number of maximal learning iterations. The numbers of states (#State(M)) and transitions (#Trans(M)) of the learned automata and the time consumption on constructing the observation tables and performing equivalence checking in the ith learning iteration are listed from row 2 to row 5 in Table V, where i ranges from 1 to 9.

3) Computing Supervisors, Controlling, and Testing the System: According to the basic principle of antilock braking, if the wheel is slipping, the BBW controller may ignore the *brake* request. Therefore, $\Sigma_c = \{\text{brake}\}$ is the controllable event set. Supervisors are calculated by Theorem 1. The numbers of states (#State(S)) and transitions (#Trans(S)) of supervisor sup_i derived on the corresponding learned automata are listed in the last two rows of Table V.

The first nontrivial supervisor is sup_2, which is implemented as a patch to control the system. Then, the integrated system, including BBW and sup_2, is tested by LBTest. A *Warning* verdict is given at the 20th testing iteration. A counterexample "*brake, acc, brake, idle, acc, brake*" is identified. The corresponding sequence of outputs is "[*still, zero, noslip*], [*still, zero, noslip*], [*moving, zero, noslip*], [*still, nonzero, noslip*], [*still, zero, slip*], [*moving, zero, slip*], [*still, nonzero, noslip*]." The violation appears at the third "*brake*" event in the counterexample. Thus, sup_2 is not a correct supervisor for the system.

Subsequently, sup_3 is used to control the BBW system. The controlled system is tested again by LBTest. Since LBTest is a testing tool, the nonexistence of counterexample does not imply that the requirement holds. In our recent study, an approximate method based on the *output pattern* definition

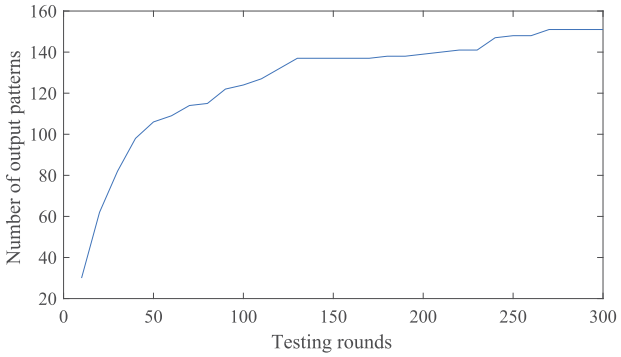

Fig. 11. Numbers of output patterns varying with testing rounds with control of sup_3 when testing Req. 2.

TABLE VI
TESTING VERDICTS OF THE SYSTEM CONTROLLED BY SUP_i

Sup i	1	2	3	4	5	6	7
Req. 1	–	–	No	–	No	No	Yes
Req. 2	–	No	Yes	No	Yes	Yes	Yes
Req. 3	–	–	Yes	–	Yes	Yes	Yes

for estimating whether the test is enough in LBTest when there is no counterexample identified is proposed [19]. Fig. 11 shows that the output pattern of the system controlled by sup_3 varies with the testing rounds. The number of output patterns increases relatively slowly after 120 testing rounds, and almost remains unchanged after 260 rounds. We conclude that the test is sufficient.

It is necessary to test whether the satisfiability of Requirements 1 and 3 is influenced in the controlled system. Thus, Requirements 1 and 3 are tested by LBTest. A counterexample is produced at the 131st testing iteration when testing Requirement 1, which is "*idle, acc, acc, acc, brake, brake*." Since Requirement 1 only concerns two abstract outputs *VehStatus* and *TorqueRR*, the corresponding output of the controlled system is: "[*still, zero*], [*still, zero*], [*still, nonzero*], [*still, nonzero*], [*still, nonzero*], [*moving, zero*], [*still, nonzero*], [*still, nonzero*]." Then, supervisor_3 is not an appropriate supervisor, which does not ensure Requirement 1.

To obtain a supervisor that satisfies all requirements, supervisors listed in Table V are tested one by one. The testing results indicating the satisfiability of Requirements 1–3 are listed in Table VI. All tests are performed by LBTest and the maximal number of testing iterations is 300.

Table VI shows that all the three requirements are satisfied in the controlled system when sup_7 is used. We further examine the quality of the testing result by plotting the numbers of output patterns [19] of the testing process for sup_7 and the three requirements in Fig. 12. Evidently, the numbers of output patterns of the controlled system become almost invariant after 250 testing iterations for all three requirements. Finally, we conclude that sup_7 is a feasible supervisor that enforces the system to satisfy all the requirements.

B. Adaptive Cruise Control in a Platooning Program

The proposed approach can also be applied to the platooning system which has proved to be effective for increasing traffic efficiency and decreasing automobile fuel

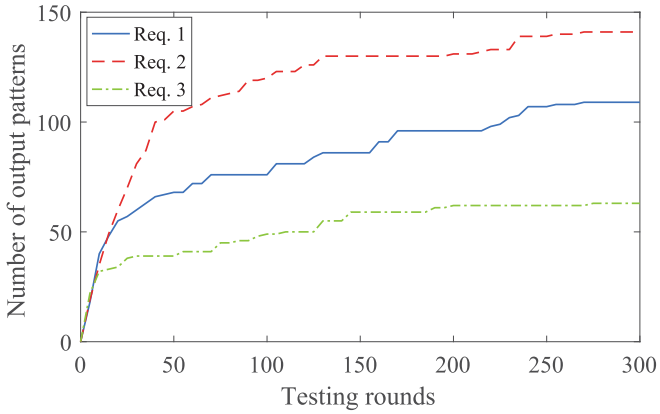

Fig. 12. Numbers of output patterns for testing sup_7 and Requirements 1–3.

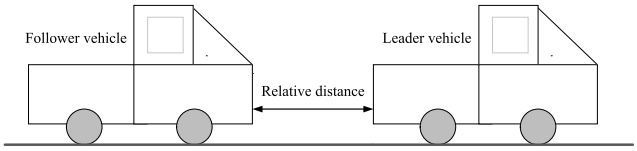

Fig. 13. Platooning with two vehicles.

consumptions [38]–[41]. This paper studies a simple platooning problem with a leader and a follower, as illustrated in Fig. 13. The leader is driven by a human driver as usual. The follower is controlled by an adaptive cruise control (ACC) system that automatically controls the follower's speed to keep a short distance from the follower and avoid collision [42]. The follower vehicle measures its relative distance to the leader by radar and laser sensors. The two vehicles also communicate via a wireless network [43]. The ACC adjusts the distance between the follower and the leader by controlling the throttle and brake commands of the follower.

Generally, there are two kinds of policies used to regulate the distance between the vehicles in an ACC: constant time-gap policy (CTP) and constant range policy (CRP). CTP maintains a constant time gap during which the follower vehicle reaches the leader vehicle's present position, while CRP maintains a constant relative distance between the two vehicles. Daniel and Athanasios [44] implemented the ACC algorithm presented in [45] and [46] as an executable Java program which is considered as the black-box system to be controlled in this paper. Furthermore, Daniel and Athanasios [44] have found in experiments that the proportional-derivative (PD) controller proposed in [45] using CRP cannot avoid collisions in the stop-and-go scenario. Therefore, we want to eliminate the collision by adding a supervisor on the ACC using the proposed approach.

1) Abstracting the System: Safety is paramount in platooning.

Requirement 4: The two vehicles shall not crash.

A sufficient condition to meet the requirement is to always keep a safe distance between the two vehicles. The output of the ACC software describes the relative distance between the leader and the follower vehicles at discrete time samples. The range of safe distance between the two vehicles is defined as follows [45]. The relative distance is a positive value denoted by x_r. Let $x_{\min}$ and $x_{\max}$ denote the reference minimum and

maximum inter-vehicle safe distances between the follower and the leader vehicles, respectively. Error represents the percentage of allowed error. The relative minimum and maximum safe distances between the vehicles are calculated by $(x_{\min} - (x_{\min} \times \text{Error})$ and $(x_{\max} + (x_{\max} \times \text{Error})$, respectively. As suggested by Daniel and Athanasios [44], we set $x_{\min} = x_{\max} = 5m$ and Error $= 0.1$ in the experiments. Thus, the relative safe distance ranges in [4.5, 5.5]m.

To analyze the relative distance qualitatively, an output abstract function f_{out} is defined as

$$f_{\text{out}}(x_r) = \begin{cases} \text{toofar}, & x_r > 5.5 \\ \text{good}, & x_r \geq 4.5 \text{ and } x_r \leq 5.5 \\ \text{tooclose}, & x_r > 0 \text{ and } x_r < 4.5 \\ \text{crash}, & x_r \leq 0.0. \end{cases} \tag{6}$$

The input signals of the leader are commands of acceleration and deceleration by pressing the gas and brake pedals, which are indicated by gasPos_l and braPos_l in the range from 0% to 100%. The input of the follower is calculated by ACC, which is either acceleration or deceleration represented by $[\text{gasPos}_f, \text{braPos}_f]$, respectively.

Let $\boldsymbol{x} = [\text{braPos}_l, \text{gasPos}_l]$ be the input vector of the leader vehicle. In the experiment, the set of abstract input events is $\Sigma_l = \{0, l\_a_i, l\_b_i | i = 1, \ldots, 10\}$. The input refinement function of the leader is defined as

$$f_{\text{in}}(\sigma) = \begin{cases} [10 \times i, 0], & \sigma = l\_b_i \\ [0, 10 \times i], & \sigma = l\_a_i \\ [0, 0], & \sigma = 0. \end{cases} \tag{7}$$

The acceleration request acc_f of the follower vehicle is calculated by a PD controller [45], where

$$\text{acc}_f = (K_d \times (v_l - v_f))/h - K_p(x_{\min} - (x_l - x_f)). \tag{8}$$

In (8), K_p and K_d are the gain parameters of the proportional and derivative parts, v_l and v_f denote the velocity of the leader and follower, x_l and x_f represent the position of the leader and follower, $x_{\min}$ is the minimal safe relative distance between the two vehicles, and h is a constant time headway. If $\text{acc}_f > 0$, the input of the follower is an acceleration. If $\text{acc}_f < 0$, the input of the follower is a deceleration. The current velocity of the follower keeps constant if $\text{acc}_f = 0$.

2) Testing the System: In the experiments, we set parameters $K_p = 3.59$, $K_d = 0.08$, $x_{\min} = 5.0$ and $h = 0.98$. Since the leader vehicle runs independently without adapting to the follower, its sampling period is 5 s. The follower vehicle has to quickly regulate its speed to keep the safe distance. The sampling period of the ACC function is 5 ms.

The ACC system is tested in two common scenarios in traffic [44]. We perform similar tests in our experiments to show that Requirement 4 is not satisfied by the given ACC. The first scenario is the stop-and-go situation. A worst case of this situation is that the leader vehicle decelerates quickly when running at a high speed. The second scenario is that the leader speeds up with smooth acceleration and switches to the cruise mode. Many test cases are executed to test the behavior of the platooning under the two traffic scenarios. Test case 1 represents the leader performing sudden accelerations and two sudden brakes. As a result of the abrupt braking, the leader almost comes to a stand-still. Test case 2 denotes the leader performing smooth acceleration and going into cruising gradually.

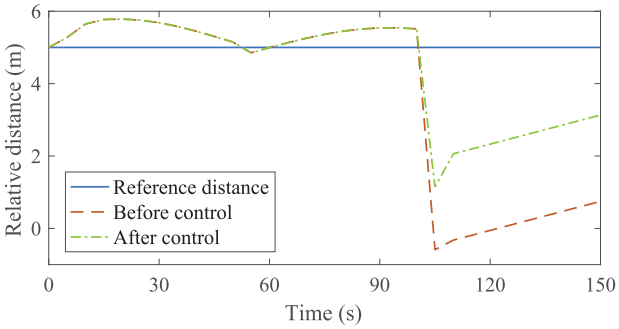

Fig. 14. Relative distance when performing Test case 1.

Test case 1: $l\_a_3, l\_a_4, l\_a_1, 0, 0, 0, 0, 0, 0, 0, l\_a_2, l\_a_5, 0, 0, 0, 0, 0, 0, 0, l\_b_6, l\_b_5, 0, 0, 0, 0, 0, 0, 0, 0$.

Test case 2: $l\_a_1, l\_a_1, l\_a_1, l\_a_1, l\_a_1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$.

When Test case 1 is performed, plot labeled with "Before control" describing the relative distance between the leader and follower vehicles is shown in Fig. 14. In Fig. 14, a collision happens at 105 s, when the leader vehicle brakes l_b_6 suddenly. The collision reveals that the ACC system is not safe in the first scenario. When Test case 2 is tested, no collision happens. The ACC works well in the second scenario.

3) Learning Moore Automata: In the platooning example, the external input of the system is positions of brake and gas pedals of the leader vehicle. If ACC of the follower vehicle cannot prevent a collision, a supervisor of the ACC system should override the output of ACC by commanding emergency brake. The supervisor cannot control the actions of the leader vehicle. Therefore, the events in set $\Sigma_l = \{0, l\_a_i, l\_b_i | i = 1, \ldots, 10\}$ are uncontrollable.

The input of the follower vehicle is calculated by (8). We introduce two controllable events f_a and f_b to control the input value of the follower vehicle, where f_a means that the ACC sends the computed decision value to the follower and f_b means that the supervisor overrides ACC's decision and applies the emergency brake. The alphabet of the follower is $\Sigma_f = \{f\_a, f\_b\}$. The alphabet of the platooning program is $\Sigma = \{l\_a_i, l\_b_i, 0, f\_a, f\_b\}$.

Since the sampling periods of the input of leader and follower are 5 s and 5 ms, 1000 speed control requests are calculated for the follower before the leader receives the next input signal. Consequently, there are 1000 states between two input events of the leader. The state space is extremely large and we just infer automaton models for the follower under specific test cases, e.g., Test case 1. Even so, the number of states becomes huge when the length of a test case is a bit long (e.g., there are 30 000 states in the complete model of the system under Test case 1). Thus, we just construct approximate supervisors for the system in this study.

Fig. 15 shows that state L_i transfers to state L_{i+1} in the model of the leader vehicle in Test case 1, when the leader vehicle receives event l_b_6. In the duration that state L_i transfers to state L_{i+1}, 1000 speed control decisions are

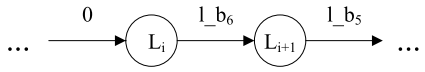

Fig. 15. Partial model of the leader vehicle.

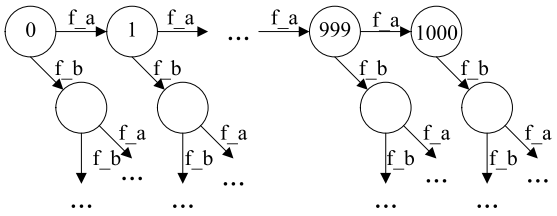

Fig. 16. Partial model of the follower vehicle.

calculated in the follower vehicle. A partial model of the follower vehicle is constructed in Fig. 16.

4) Computing Supervisors, Controlling and Testing the System: In this experiment, there are 30 000 states in the learned Moore automaton of the first scenario. In the supervisor, the event f_b is executed from states 20 150 to 20 825, which sends the highest request to the brake pedal ($\text{acc}_f = -20$). The supervisor is patched to the ACC system. The controlled system is tested again. The plot describing the relative distance between the leader and the follower vehicles is shown in Fig. 14 labeled with "After control." In Fig. 14, it is shown that the collision at 105 s is avoided in the controlled system under Test case 1. The controlled ACC system still works well with Test case 2. As a result, it can be concluded that the proposed approach works effectively in the platooning program.

As far as we know, this is the first attempt of applying SCT to platooning systems. In further studies, we shall find a general supervisor for the system with a scenario where the leading vehicle stops unexpectedly at different time.

VI. Conclusion

We propose an approach to improve the quality of black-box embedded systems when the formal models of the system and requirement are not directly available. The approach integrates automaton learning algorithms and SCT of DES. The system is tested against the requirement. If the requirement is not satisfied, the C^* algorithm is proposed to infer a Moore automaton. Then a supervisor is derived by SCT on the learned automaton and patched onto the system to correct erroneous behavior of the system. Finally, the controlled system is tested again to verify the correctness of the supervisor. The procedure iterates until the requirement holds in the controlled system. The proposed approach is implemented automatically. Experiments are performed on the BBW system and a platooning program to illustrate the feasibility of the approach to realistic systems.

References

[1] G. Zhu, Z. Li, N. Wu, and A. Al-Ahmari, "Fault identification of discrete event systems modeled by Petri nets with unobservable transitions," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 49, no. 2, pp. 333–345, Feb. 2019.

[2] G. H. Zhu, Z. W. Li, and N. Q. Wu, "Model-based fault identification of discrete event systems using partially observed Petri nets," *Automatica*, vol. 96, pp. 201–212, Oct. 2018.

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

[4] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. New York, NY, USA: Springer-Verlag, 2007.

[5] Y. Chen, Z. Li, and M. Zhou, "Optimal supervisory control of flexible manufacturing systems by Petri nets: A set classification approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 549–563, Apr. 2014.

[6] H. F. Chen, N. Q. Wu, Z. W. Li, and T. Qu, "On a maximally permissive deadlock prevention policy for automated manufacturing systems by using resource-oriented Petri nets," *ISA Trans.*, vol. 89, pp. 67–76, Jun. 2019.

[7] G. Y. Liu, P. Li, Z. W. Li, and N. Q. Wu, "Robust deadlock control for automated manufacturing systems with unreliable resources based on petri net reachability graphs," *IEEE Trans. Syst., Man Cybern., Syst.*, vol. 49, no. 7, pp. 1371–1385, Jul. 2019.

[8] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," *J. Automata, Lang. Combinatorics*, vol. 7, no. 2, pp. 225–246, 2002.

[9] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Aug. 1987.

[10] M. Shahbaz, and R. Groz, "Analysis and testing of black-box component-based systems by inferring partial models," *Softw. Test., Verification Rel.*, vol. 24, no. 4, pp. 253–288, Jun. 2014.

[11] F. Aarts, B. Jonsson, J. Uijen, and F. Vaandrager, "Generating models of infinite-state communication protocols using regular inference with abstraction," *Formal Methods Syst. Des.*, vol. 46, no. 1, pp. 1–41, Feb. 2015.

[12] B. Caldwell, R. Cardell-Oliver, and T. French, "Learning time delay mealy machines from programmable logic controllers," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 1155–1164, Apr. 2016.

[13] M. Lemmon, P. Antsaklis, X. Yang, and C. Lucisano, "Control system synthesis through inductive learning of Boolean concepts," *IEEE Control Syst. Mag.*, vol. 15, no. 3, pp. 25–36, Jun. 1995.

[14] X. Yang, M. Lemmon, and P. Antsaklis, "Inductive inference of logical DES controllers using the L^* algorithm," in *Proc. Amer. Control Conf.*, Seattle, WA, USA, Jun. 1995, pp. 3163–3167.

[15] J. Dai and H. Lin, "A learning-based synthesis approach to decentralized supervisory control of discrete event systems with unknown plants," *Control Theory Technol.*, vol. 12, no. 3, pp. 218–233, Aug. 2014.

[16] B. Wu and H. Lin, "Counterexample-guided distributed permissive supervisor synthesis for probabilistic multi-agent systems through learning," in *Proc. Amer. Control Conf. (ACC)*, Boston, MA, USA, Jun. 2016, pp. 5519–5524.

[17] J. Dai, A. Benini, H. Lin, P. J. Antsaklis, M. J. Rutherford, and K. P. Valavanis, "Learning-based formal synthesis of cooperative multi-agent systems with an application to robotic coordination," in *Proc. 24th Medit. Conf. Control Automat. (MED)*, Jun. 2016, pp. 1008–1013.

[18] K. Hiraishi, "Synthesis of supervisors using learning algorithm of regular languages," *Discrete Event Dyn. Syst.*, vol. 11, no. 3, pp. 211–234, Jul. 2001.

[19] H. Zhang, L. Feng, N. Wu, and Z. Li, "Integration of learning-based testing and supervisory control for requirements conformance of black-box reactive systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 1, pp. 2–15, Jan. 2018.

[20] H. M. Zhang, L. Feng, and Z. W. Li, "A learning-based synthesis approach to the supremal nonblocking supervisor of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 63, no. 10, pp. 3345–3360, Oct. 2018.

[21] C. Gu, Z. Li, N. Wu, M. Khalgui, T. Qu, and A. Al-Ahmari, "Improved multi-step look-ahead control policies for automated manufacturing systems," *IEEE Access*, vol. 6, pp. 68824–68838, 2018.

[22] K. Meinke and M. A. Sindhu, "LBTest: A learning-based testing tool for reactive systems," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation*, Mar. 2013, pp. 447–454.

[23] (2019). *LTL 2 BA: Fast Translation From LTL Formulae to Büchi Automata*. [Online]. Available: http://www.lsv.fr/ gastin/ltl2ba/

[24] C. Gu, X. Wang, and Z. Li, "Synthesis of supervisory control with partial observation on normal state-tree structures," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 984–997, Apr. 2019.

[25] (2019). *Learnlib: A Framework for Automata Learning*. [Online]. Available: http://learnlib.de/

[26] M. Isberner, F. Howar, and B. Steffen, "The open-source LearnLib," in *Proc. Int. Conf. Comput. Aided Verification*, Jul. 2015, pp. 487–495.

[27] L. Feng and W. M. Wonham, "TCT: A computation tool for supervisory control synthesis," in *Proc. 8th Int. Workshop Discrete Event Syst.*, Jul. 2006, pp. 388–389.
[28] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optim.*, vol. 25, no. 3, pp. 637–659, 1987.
[29] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
[30] K. Meinke, "CGE: A sequential learning algorithm for mealy automata," in *Proc. Int. Colloq. Grammatical Inference*, Sep. 2010, pp. 148–162.
[31] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Softw. Eng.*, vol. 4, no. 3, pp. 178–187, May 1978.
[32] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, pp. 591–603, Jun. 1991.
[33] L. Feng, S. Lundmark, K. Meinke, F. Niu, M. A. Sindhu, and P. Y. H. Wong, "Case studies in learning-based testing," in *Proc. 25th Int. Conf. Test. Softw. Syst.*, Nov. 2013, pp. 164–179.
[34] R. Marinescu, M. Saadatmand, A. Bucaioni, C. Seceleanu, and P. Pettersson, "A model-based testing framework for automotive embedded systems," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2014, pp. 38–47.
[35] R. Marinescu and E. P. Enoiu, "Extending EAST-ADL for modeling and analysis of system's resource-usage," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2012, pp. 532–537.
[36] E.-Y. Kang and P.-Y. Schobbens, "Schedulability analysis support for automotive systems: From requirement to implementation," in *Proc. 29th Annu. ACM Symp. Appl. Comput.*, Mar. 2014, pp. 1080–1085.
[37] K. Meinke and M. Sindhu, "Correctness and performance of an incremental learning algorithm for finite automata," in *Proc. 3rd Asian Conf. Mach. Learn.*, Nov. 2011, pp. 1–19.
[38] C. Bonnet and H. Fritz, "Fuel consumption reduction in a platoon: experimental results with two electronically coupled trucks at close spacing," SAE Tech. Paper 2000-01-3056, 2000. [Online]. Available: https://www.sae.org/publications/technical-papers/content/2000-01-3056/
[39] A. Alam, "Fuel-efficient heavy-duty vehicle platooning," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, SE, USA, 2014.
[40] O. Karoui, M. Khalgui, A. Koubâa, E. Guerfala, Z. Li, and E. Tovar, "Dual mode for vehicular platoon safety: Simulation and formal verification," *Inf. Sci.*, vol. 402, pp. 216–232, Sep. 2017.
[41] O. Karoui *et al.*, "Performance evaluation of vehicular platoons using Webots," *IET Intell. Transp. Syst.*, vol. 11, no. 8, pp. 441–449, 2017.
[42] A. Alam, B. Besselink, V. Turri, J. Mårtensson, and K. H. Johansson, "Heavy-duty vehicle platooning for sustainable freight transportation: A cooperative method to enhance safety and efficiency," *IEEE Control Syst. Mag.*, vol. 35, no. 6, pp. 34–56, Dec. 2015.
[43] X. Zhang, X. Song, L. Feng, L. Chen, and M. Törngren, "A case study on achieving fair data age distribution in vehicular communications," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)* Pittsburgh, PA, USA, Apr. 2017, pp. 307–318.
[44] T. Daniel and T. Athanasios, "Athanasios, adaptive cruise control implementation with constant range and constant time-gap policies," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2017.
[45] R. S. Kakade, "Automatic cruise control system," Ph.D. dissertation, Eng. Indian Inst. Technol., Mumbai, India, 2017.
[46] K. Liang, "Linear quadratic control for heavy duty vehicle platooning," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2011.

Huimin Zhang received the B.S. and M.S. degrees in information and computing sciences and computer application technology from the Guilin University of Electronic Technology, Guilin, China, in 2005 and 2008, respectively, and the Ph.D. degree in control theory and control engineering from Xidian University, Xi'an, China, in 2018.

In 2019, she joined the College of Computer Science and Information Engineering and the College of Software, Guangxi Normal University, Guilin. Her current research interests include supervisory control theory of discrete-event systems, automata theory, and automata learning theory.

Lei Feng (M'07) received the B.S. and M.S. degrees from the Department of Mechanical and Electronic Engineering, Xi'an Jiaotong University, Xi'an, China, in 1998 and 2001, respectively, and the Ph.D. degree from the Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, in 2007.

He joined the Mechatronics and Embedded Control System Division, Department of Machine Design, KTH Royal Institute of Technology, Stockholm, Sweden, in 2012, where he is currently an Associate Professor. His main research interests include formal methods for the verification and control synthesis of cyber-physical systems, energy management control of mechatronic systems, and supervisory control of discrete-event systems.

Zhiwu Li (M'06–SM'07–F'16) received the B.S. degree in mechanical engineering, the M.S. degree in automatic control, and the Ph.D. degree in manufacturing engineering from Xidian University, Xi'an, China, in 1989, 1992, and 1995, respectively.

He was a Visiting Professor with the University of Toronto, Toronto, ON, Canada; the Technion–Israel Institute of Technology, Haifa, Israel; the Martin-Luther-University of Halle-Wittenburg, Halle, Germany; Conservatoire National des Arts et Métiers, Paris, France; and Meliksah Universitesi, Kayseri, Turkey. In 1992, he joined Xidian University. He is currently with the Institute of Systems Engineering, Macau University of Science and Technology, Macau, China. His current research interests include Petri net theory and applications, supervisory control of discrete-event systems, work-flow modeling and analysis, system reconfiguration, game theory, and data and process mining.

Dr. Li chairs the Discrete-Event Systems Technical Committee of the IEEE Systems, Man, and Cybernetics Society and served as a member of IFAC Technical Committee on Discrete-Event and Hybrid Systems, from 2011 to 2014. He was a recipient of the Alexander von Humboldt Research Grant, Alexander von Humboldt Foundation, Germany. He is listed in *Marquis Who's Who in the World* (27th ed., 2010). He is the Founding Chair of Xi'an Chapter of IEEE Systems, Man, and Cybernetics Society. He serves as a frequent reviewer for more than 90 international journals, including *Automatica*, a number of IEEE Transactions, and many international conferences.