IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 17, 2020, accepted February 24, 2020, date of publication March 24, 2020, date of current version April 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2983074

Learning a Partially-Known Discrete Event System

IRA WENDELL BATES, II[1], ALI KARIMODDINI[1], (Senior Member, IEEE), AND MOHAMMAD KARIMADINI[2]

[1]North Carolina A&T State University, Greensboro, NC 27411, USA
[2]Department of Electrical Engineering, Arak University of Technology, Arak 3818146763, Iran

Corresponding author: Ali Karimoddini (akarimod@ncat.edu)

The work of Ira Wendell Bates, II, was supported in part by the Air Force Research Laboratory and OSD under Grant FA8750-15-2-0116, and in part by the Title III HBGI Grant from the U.S. Department of Education. The work of Ali Karimoddini was supported in part by the National Science Foundation under Award 1832110, and in part by the Air Force Research Laboratory and OSD under Grant FA8750-15-2-0116.

ABSTRACT There are many cases in which our understanding of a system may be limited due to its complexity or lack of access into the entire system, leaving us with only partial system knowledge. This paper proposes a novel systematic active-learning method for realizing a partially-known Discrete Event System (DES). The proposed technique takes the available information about the system into account by tabularly capturing the known data from the system, and then, discovers the unknown part of the system via an active-learning procedure. For this purpose, a series of tables will be constructed to first infer the information about the system from the available data, and if unavailable, the developed algorithm collects the information through basic queries made to an oracle. It is proven that the developed technique returns a language-equivalent finite-state automaton model for the system under identification after a finite number of iterations. A real-world illustrative example is provided to explain the details of the proposed method.

INDEX TERMS Discrete event systems, partially-known systems, active-learning, complex systems, systems identification, automata theory, manufacturing systems, automotive industries.

I. INTRODUCTION

The complexity of engineered systems has significantly increased over the years, creating a considerable need for multiresolution methods of analysis, design, and testing in order to control low-level dynamics to achieve fine requirements while supervising the high-level behaviors to fulfill the logical specifications. The Discrete Event System (DES) framework, in which the high-level behaviors of the system are captured by sequences of events (abrupt and distinct changes in the system and its operation modes), has gained much attention due to its innate ability to represent the logical behavior of complex systems in an abstract, yet effective way. Many different methods have been studied to model DES such as Petri Nets [1]–[8], Process Algebra [9], [10], and Automata Theory [11], [12]. The benefit of modeling a DES as an automaton is that it provides an intuitive and visual representation of the system that is amenable to composition operations as well as analysis [13], [14].

The associate editor coordinating the review of this manuscript and approving it for publication was Shouguang Wang.

Analytical tools have been developed to analyze the aforementioned modeling methods such as observers [15]–[17], diagnosers [18]–[20], and supervisors [21]–[24]. Most of these methods assume perfect knowledge about the system, which, realistically, may not always be readily accessible for analysis.

Learning unknown DES has a wide field of applications in literature ranging from robotics and control systems to data mining, and many other interesting applications [25], [26]. Generally speaking, learning algorithms fall into two different types of categories [27]: *active* [25], [28] and *passive* [29], [30]. Unlike passive learning techniques, active-learning algorithms engage in choosing examples that are believed to provide sufficient information by actively making queries regarding the existence (or nonexistence) of strings in the language of the DES system being assessed.

Different active-learning techniques for automata in literature have been considered when addressing learning DES such as L^* [28], NL^* [31], and Homing-Sequences [32]. All of these techniques assume initially completely unknown systems which, in practice, might not always be the case. However, with these methods, it is not possible to utilize the

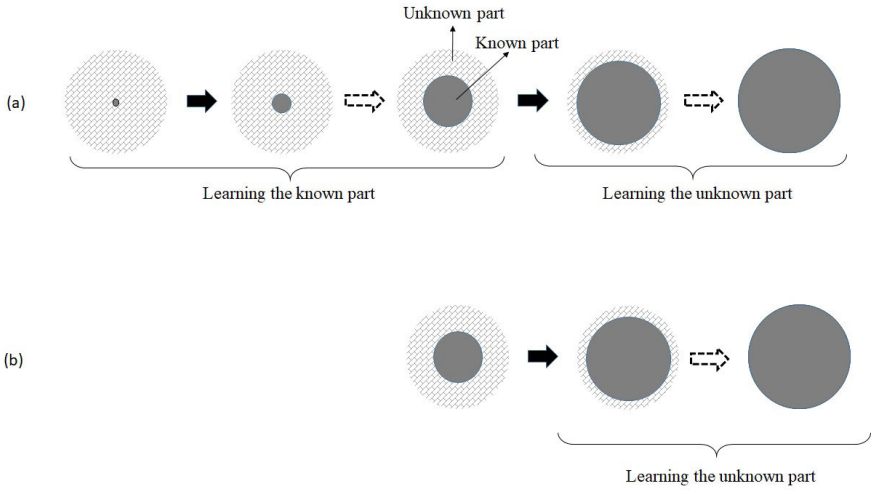


FIGURE 1. Symbolic representation of learning strategies for a partially known system: (a) ignoring the known information and applying a learning technique for both the known and unknown parts, and (b) taking into account the known information about the system, and then begin to explore and learn only the unknown part.

already known information about the system. In other words, employing [28], [31], [32] in order to identify a partially-known system, one should ignore the already known information and inefficiently treat the system as a completely unknown system, imposing more computation and interactions (queries to the oracle) to the active learning process (see Figure 1.a for symbolic visualization of this approach). Compared to [28], [31], [32], we develop a novel method that can be applied to partially known systems, by efficiently taking into account the known part, followed by identifying the unknown portion (see Figure 1.b for symbolic visualization of the proposed approach).

Therefore, this paper proposes a novel approach for learning partially-known systems. Inspired by L^*-Algorithm [28], [33], we develop an active-learning technique to explore the unknown portion of a given partially-known system. The proposed technique, first examines the rich set of information readily provided from the known part of the system and captures this information in the form of a Boolean table where the rows and columns are associated with the existence of different strings. Then, to explore the unknown part of the system, if the information cannot be inferred from the available data, the proposed technique makes basic queries to an oracle to assess the existence of particular strings that can assist in completing the information about the unknown part. These queries aid in the further construction of a series of tables until an automaton is built which is language-equivalent to the original partially-known system under identification. We have proven that such a language-equivalent automaton can be constructed in a finite number of iterations. Illustrative examples are provided to detail the steps of the developed algorithms.

The organization of the remainder of this paper is as follows: Section II formulates the problem of learning a partially-known DES system and provides preliminary definitions and notations. In Section III, we develop an effective technique for learning a partially-known DES system. Section IV provides a simplified real-world illustrative example to explain the details and different steps of the proposed algorithms. Section V concludes the paper.

II. PROBLEM FORMULATION

Consider the system modeled by a nondeterministic automaton G as follows:

$$G = (X, \Sigma, \delta, x_0) \tag{1}$$

where X is the state space, Σ is the event set, $\delta : X \times \Sigma \to 2^X$ is the transition relation, and x_0 is the initial state. The sequence of events forms a *string*, and a set of strings forms a *language*. With the abuse of notation, we use $e \in s$ to say that the event e belongs to the string s if the event e is one of the events forming the string s. The length of a string s is shown by $|s|$. The concatenation of the strings s_1 and s_2 is denoted by $s_1.s_2$. Let $\Sigma^n = \{e_1.e_2 \cdots e_n \mid e_i \in \Sigma\}$ for $n \in \mathbb{N}$, where $\Sigma^0 = \varepsilon$. Then, Σ^* denotes all possible finite-length strings over Σ including the zero-length string ε.

The transition relation δ can be recursively extended to strings by revising its definition as $\delta : X \times \Sigma^* \to 2^X$ where $\delta(x, s.e) = \bigcup_{y \in \delta(x,s)} \delta(y, e)$. The language $\mathcal{L}(G(x)) = \{s \in \Sigma^* \mid \delta(x, s) \text{ is defined}\}$ includes the strings that can be executed by G from x. The language of G, denoted by $\mathcal{L}(G)$, is the sequence of strings that can be generated by G from x_0, i.e., $\mathcal{L}(G) = \mathcal{L}(G(x_0)) = \{s \in \Sigma^* \mid \delta(x_0, s) \text{ is defined}\}$. Automata G_1 and G_2 are language-equivalent if $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. Consider the string $s = s_p.u.s_s$, then s_p and s_s are called the prefix and suffix of the string s, respectively. We say that $s_1 \leq s_2$ if s_1 is a prefix of s_2. The sets $Pre(\mathcal{L})$ and $Suf(\mathcal{L})$ contain all prefixes and suffixes of $\mathcal{L}$, respectively. The language $\mathcal{L}$ is said to be prefix/suffix-closed if all

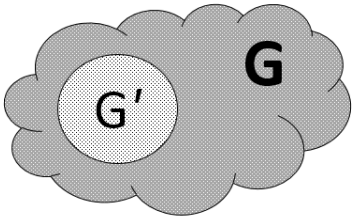


FIGURE 2. A partially known DES system G whose subautomaton G' is known.

prefixes/suffixes of all strings in the language $\mathcal{L}$ are also members of this language. The language $\mathcal{L}(G)/s = \{t \in \Sigma^* \mid s.t \in \mathcal{L}(G)\}$ is the set of all strings in $\mathcal{L}(G)$ that occur after $s \in \mathcal{L}(G)$.

Definition 1: $G_1 = (X_1, \Sigma_1, \delta_1, x_{0_1})$ *is a sub-automaton of* $G_2 = (X_2, \Sigma_2, \delta_2, x_{0_2})$, *shown by* $G_1 \sqsubseteq G_2$, *if and only if* $x_{0_1} = x_{0_2}$, $\Sigma_1 = \Sigma_2$, $X_1 \subseteq X_2$, *and for any* $x_1, x_2 \in X_1 \subseteq X_2$ *and for any* $e \in \Sigma_2$ *with* $x_2 \in \delta_1(x_1, e)$, *then* $x_2 \in \delta_1(x_1, e)$.

It can be verified that for $G_1 \sqsubseteq G_2$, we have $\delta_1 \subseteq \delta_2$ and $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$.

Problem 1: Consider a discrete event system G, whose sub-automaton G′ is known. Given the known part, G′, construct a language-equivalent model of G.

III. THE PROPOSED ACTIVE-LEARNING METHOD

Consider a partially-known system G as shown in Figure 2, which contains information and behavior we already know, the sub-automaton G', as well as *hidden* behavior we do not, the gray area of G. To identify such a partially-known system and address Problem 1, we first capture the information about the known part in Section III-A, and then in Section III-B, we explore the unknown part of the system.

A. CAPTURING THE INFORMATION ABOUT THE PARTIALLY-KNOWN PART OF THE SYSTEM

The proposed method seeks to take advantage of and utilize the information about the known portion of the system that is readily available to us. For this purpose, to extract the information from the known part of the system $G' = (X', \Sigma, \delta', x_0)$, we form a series of tables $V'_i = (S', E'_i, T')$, where E'_i (a nonempty, finite, suffix-closed set) consists of the labels of the table columns; S' (a nonempty, finite, prefix-closed set) and $S'.\Sigma - S'$ are two disjoint sets which include the labels for the rows of the table, and T' is a function mapping the set of $((S'.\Sigma \cup S').E'_i)$ to $\{0, 1\}$ determining the elements of the table. For any row label $s \in S'.\Sigma \cup S'$ and any column label $t \in E'_i$, we have:

$$T'(s.t) = \begin{cases} 1, & \text{if } s.t \in \mathcal{L}(G') \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

To show the elements of a row in the table in the form of an ordered pair, we define the function $row(s) = (T'(s.t_1), T'(s.t_2), \cdots, T'(s.t_{|E'_i|}))$, where $t_j \in E'_i$ is the label of the j-th column in E'_i.

Now, to form the Table $V'_i = (S', E'_i, T')$, we set $S' = Pre(\Sigma^\ell)$, where ℓ should be selected in a way that strings in S' will reach all states in G', which can be found as:

$$\ell = \max_{x \in X'} \min_{s \in S^x_{x_0}} |s| \tag{3}$$

where $S^x_{x_0} = \{u \mid u \in \mathcal{L}(G'),\ x \in \delta'(x_0, u)\}$.

We propose the following lemmas:

Lemma 1: All states in G′ are reachable by strings in $S' = Pre(\Sigma^\ell)$, *i.e.,* $X' = \{x \in \delta'(x, s) \mid s \in S'\}$.

Proof: For any $x \in X'$, there is a string $s = \arg\min_{u \in S^x_{x_o}} |u|$, where $x \in \delta'(x_0, s)$. By construction (See (3)), $|s| \leq \ell$, and hence, $s \in \Sigma^{|s|} \subseteq S'$. ■

For the first Table $V'_1 = (S', E'_1, T')$, we set $E'_1 = \{\varepsilon\}$, and then will fill the Table V'_1 using T' accordingly. We then check if Table V'_1 is consistent:

Definition 2: The Table $V'_i = (S', E'_i, T')$ *is said to be consistent iff*:

$$\forall s_1, s_2 \in S' \text{ with } row(s_1) = row(s_2) \Rightarrow row(s_1.\sigma) = row(s_2.\sigma), \forall \sigma \in \Sigma \tag{4}$$

If the Table $V'_i = (S', E'_i, T')$ is not consistent, it means that there are two strings $s_1, s_2 \in S'$, $t \in E'_i$, $\sigma \in \Sigma$ such that $row(s_1) = row(s_2)$ but $T'(s_1.\sigma.t) \neq T'(s_2.\sigma.t)$. To resolve this inconsistency, it is sufficient to add $\sigma.t$ to E'_{i+1} and then fill the new Table V'_{i+1} using T'. For example, in Figure 8.a, the Table V'_1 is not consistent due to the fact that $row(a) = 1$ and $row(aa) = 1$ while $row(a.a) = 1$ and $row(aa.a) = 0$. Therefore, to fix this problem, we set $E'_2 = E'_1 \cup \{a\}$ in V'_2 and then fill the Table V'_2 accordingly (see Figure 8.b). This process of finding and resolving inconsistencies in a table can be done for a finite number of times and will eventually result in a consistent table as it is shown in Lemma 4. Before that, we need to prove the following lemmas:

Lemma 2: In $V'_i = (S', E'_i, T')$, *if* $T'(s) = 0$, *then s is a zero-row, i.e.,* $row(s) = (0, \cdots, 0)$.

Proof: If $T'(s) = 0$, it means that $s \notin \mathcal{L}(G')$, which in turn means that $\delta'(x_0, s)$ is not defined. Therefore, for any other $t \in E'_i$, $\delta'(x_0, s.t) = \bigcup_{y \in \delta'(x_0, s)} \delta'(y, t)$ is not defined as $\delta'(x_0, s)$ is not defined. Hence, $row(s) = (T'(s.t_1), \cdots, T'(s.t_{|E'_i|})) = (0, \cdots, 0)$. ■

Lemma 3: In the Table $V'_i = (S', E'_i, T')$, *the number of distinguished non-zero rows in S′ is less than or equal to the number of states of G′.*

Proof: Consider a row label t for a non-zero row. Since t is a non-zero row, based on Lemma 2, we have $T'(t) \neq 0$, meaning that $\delta'(x_0, t)$ is defined, i.e., $t \in \mathcal{L}(G')$. For each $s \in S'$, $row(s) \neq row(t)$, there exists $u \in E'_i$ such that $T'(t.u) \neq T'(s.u)$, which is equivalent to $\delta'(x_0, t.u) \neq \delta'(x_0, s.u)$. This can only happen when there exists a state $x \in \{\delta(x_0, t) - \bigcup_{s \in S'} \delta(x_0, s)\}$, i.e., $x \in X'$ is a distinguished state, which is not reachable by $s \in S'$. This means that any pair of distinguished non-zero rows in S' corresponds to two distinguished states in G'. Since, there are only $|X'|$ states

in G', there will be at most $|X'|$ number of distinguished non-zero rows in S'. ■

Lemma 4: A Table $V'_i = (S', E'_i, T')$ can be made consistent within a finite number of iterations.

Proof: Consider the labels t and s for non-zero distinguished rows with $row(t) = row(s)$. Assume that the Table V'_i is not consistent as there exist $u \in E'_i$ and $\sigma \in \Sigma$ such that $T(t\sigma.u) \neq T(s\sigma.u)$. To fix this inconsistency, we should add $\sigma.u$ to E'_{i+1}, and then fill the Table $V'_{i+1} = (S', E'_{i+1}, T')$ using T'. In this updated table, since $T(t.\sigma u) \neq T(s.\sigma u)$, $row(t) = (T(t.e_1), \cdots, T(t.\sigma u))$ and $row(s) = (T(s.e_1), \cdots, T(s.\sigma u))$ are no longer the same. This process, therefore, adds at least one distinguished row to S'. However, based on Lemma 3, the number of distinguished non-zero rows of S' is bounded by $|X'|$. Therefore, one needs to resolve the inconsistency with this procedure at most for $|X'|$ times. ■

Next, we check if the Table $V'_i = (S', E'_i, T')$ is closed:

Definition 3: The Table $V'_i = (S', E'_i, T')$ is said to be closed iff:

$$\forall t \in (S'.\Sigma - S') \text{ with } row(t) \neq (0, \cdots, 0), \quad \exists s \in S' \text{ such that } row(s) = row(t) \tag{5}$$

If the Table $V'_i = (S', E'_i, T')$ is not closed, it means that there is a string $t \in (S'.\Sigma - S')$ where $row(t)$ does not match any $row(s)$ for all $s \in S'$. To make this table closed, the string t must be added to S', and the table must be extended and filled accordingly.

Lemma 5: The Table $V'_i = (S', E'_i, T')$ is closed.

Proof: By contradiction, assume that V'_i is not closed, i.e., there exists $t \in S'.\Sigma - S'$ such that $row(t) \neq (0, \cdots, 0)$ and $row(t) \neq row(s)$ for all $s \in S'$. Since $row(t) \neq 0$, then based on Lemma 2, we have $T'(t.\varepsilon) = T'(t) = 1$, concluding that $t \in \mathcal{L}(G')$. Since $row(t) \neq row(s)$ for all $s \in S'$, this means that for each $s \in S'$, there exists $u \in E'_i$ such that $T'(t.u) \neq T'(s.u)$, which is equivalent to $\delta'(x_0, t.u) \neq \delta'(x_0, s.u)$. This can only happen when there exists a state $x \in \delta(x_0, t) - \bigcup_{s \in S'} \delta(x_0, s)$, i.e., $x \in X'$ is not reachable by $s \in S'$. This contradicts Lemma 1. ■

Algorithm 1 details the process of capturing the information about the known part of the system, G', which starts by initialization of S' and E'_i by setting $S' = Pre(\Sigma^\ell)$ and $E'_1 = \varepsilon$, followed by constructing $V'_1 = (S', E'_1, T')$. Then, through an iterative process, the table is made consistent. Lemma 4 shows that the consistency *while*-loop in Algorithm 1 (lines 5-10) terminates after a finite number of iterations. At the end, Algorithm 1 returns the Table $V' = (S', E', T')$, which is consistent. Also, in Lemma 5, it was shown that $V' = (S', E', T')$ is closed. For the Table $V' = (S', E', T')$ which is complete (closed and consistent), we can construct the automaton $G(V') = (X'_v, \Sigma, \delta'_v, x_0{}'_v)$, where

$$\begin{aligned} X'_v &= \{row(s) \mid s \in S', T'(s) = 1\} \\ \delta'_v(row(s), \sigma) &= row(s.\sigma) \\ x_0{}'_v &= row(\varepsilon) \end{aligned} \tag{6}$$

Algorithm 1 Capturing the Partially-Known Information

1: **input:** $G' = (X', \Sigma, \delta', x_0)$, the partially known part of the automaton
2: **output:** The complete Table $V' = (S', E', T')$ which captures the known part G'
3: **initialization:** $S' = Pre(\Sigma^\ell)$ and $E'_1 = \varepsilon$
4: Fill the the Table $V'_1 = (S', E'_1, T')$
5: **while** $V'_i = (S', E'_i, T')$ is not consistent **do**
6: Find $s_1, s_2 \in S'$, $\sigma \in \Sigma$, and $e \in E'_i$ such that $row(s_1) = row(s_2)$ but $T'(s_1.\sigma.e) \neq T'(s_2.\sigma.e)$;
7: $E'_{i+1} = E'_i \cup \{\sigma.e\}$
8: Form the new Table $V'_{i+1} = (S', E'_{i+1}, T')$ and fill it up over $(S' \cup S'.\Sigma).E'_{i+1}$
9: Set $i = i + 1$;
10: **end while**
11: **return:** $V' = V'_i = (S', E'_i, T')$

For example, Figure 9 shows an automaton for the complete Table V'_3 in Figure 8.c.

The table $V' = (S', E', T')$ captures the information about G' as its corresponding automaton $G(V')$ and G' are language equivalent as it is stated in the following theorem:

Theorem 1: The automaton $G(V') = (X'_v, \Sigma, \delta'_v, x_0{}'_v)$, which corresponds to the complete Table $V' = (S', E', T')$, and $G' = (X', \Sigma, \delta', x_0)$ are language-equivalent.

Proof: Consider any string $s = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G')$. Let $s(i) = \sigma_1 \cdots \sigma_i$, $i = 1, \ldots, n$. In G', as a finite state automaton, for any string $s(i-1)$, there exists a minimal length string $u \leq \ell$ such that $\delta'(x_0, s(i-1)) = \delta'(x_0, u) = row(u) \in X'_v$, where $u \in S'$, as $u \in Pre(\Sigma^\ell)$. Therefore, for the transition $\delta'(\delta'(x_0, s(i-1)), \sigma_i) = \delta'(\delta'(x_0, u), \sigma_i)$, we have $\delta'_v(x_{0v}, s(i)) = \delta'_v(\delta'_v(x_{0v}, u), \sigma_i) = \delta'_v(row(u), \sigma_i) = row(u\sigma_i)$. Note that $row(u\sigma_i) = (T'(u\sigma_i), T'(u\sigma_i.e_2), \cdots, T'(u\sigma_i, e_{|E'|}))$, $e_i \in E'$, and $row(u\sigma_i) \neq (0, \cdots, 0)$ as $u\sigma_i \in \mathcal{L}(G')$ and hence $T'(u\sigma_i) = 1$. Moreover, since $|u| \leq \ell$, $row(u) \in X'_v$, as $u \in S' = Pre(\Sigma^\ell)$. Therefore, $u\sigma_i \in S'.\Sigma - S$ and $row(u\sigma_i) \in X'_v$, otherwise for $u\sigma_i$, there is no $s \in S'$ so that $row(s) = row(u\sigma_i)$, which contradicts the completeness of V'. Since $row(u\sigma_i) \in X'_v$, we can conclude that $\delta'(x_0, s(i)) = \delta'_v(row(u), \sigma_i) = row(u\sigma_i)$ is defined and $s(i) \in \mathcal{L}(G(V'))$. Conversely, and with a similar argument, it can be proven that for any $s = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G(V'))$, it is also in $\mathcal{L}(G')$. ■

B. ACTIVE-LEARNING OF THE UNKNOWN PART

Now that we have information about the known part of the system captured by V', it is time to explore the unknown part of G. For this purpose, inspired by [28], we propose an active-learning approach (Algorithm 2), that can reveal the information about the unknown part and construct an automaton G_t which is language-equivalent to G. This will be accomplished through basic queries proposed to a minimally adequate oracle that is only able to provide *yes* or *no* answers to two types of queries:

- Membership query: in which the algorithm asks whether a string s belongs to $\mathcal{L}(G)$.
- Equivalence query: in which the algorithm asks whether $\mathcal{L}(G) = \mathcal{L}(G_t)$. If not, the oracle returns a counterexample *cex* that is in the symmetric difference between G and G_t.

The collected and inferred information will be sorted in a series of Tables $V_i = (S_i, E_i, T)$. The Table $V_i = (S_i, E_i, T)$ has a structure similar to the previous section with the column label set E_i and the row label sets S_i and $S_i.\Sigma - S_i$. We fill the tables using T. For any row label $s \in (S_i.\Sigma \cup S_i)$ and column label $t \in E_i$, the table entry $T(s.t)$ can be found as:

- $T(s.t) = 1$ if $s.t \in \mathcal{L}(G)$
- $T(s.t) = 0$ if $s.t \notin \mathcal{L}(G)$

To determine the value of $T(s.t)$, the algorithm first explores if the value of $T(s.t)$ can be found in or inferred from the existing information. If not, then the algorithm treats $T(s.t)$ as a membership query to the oracle.

Now, to form $V_i = (S_i, E_i, T)$, $i \geq 1$, the algorithm first starts with $V_1 = V'$. Recall that the Table V' contains all information about the known part. If a transition $\delta'(x_0, s)$ was observed in G', we let $T'(s) = 1$ in V'. This will not change in V, i.e., $T(s) = 1$. The problem, however, is when there was no transition $\delta'(x_0, s)$ in G' and we let $T'(s) = 0$ based on the information from the known part. While such a transition did not exist in G', it may exist in G. Therefore, we need to check all the zero elements in V_1. The set of all zero elements in V_1 is called Z, which can be defined as:

$$Z = \{s.t \mid s \in (S_1 \cup S_1.\Sigma),\ t \in E_1,\ T'(s.t) = 0\} \tag{7}$$

To check zero elements in V_1, we should make membership queries $T(s)$ for all $s \in Z$ and update them in V_1. For this purpose, we form the set Z_M for the updated values:

$$Z_M = \{z \in Z \mid T(z) = 1\} \tag{8}$$

In addition to updating the table for elements in Z_M, any member of Z_M could serve as a counterexample. To make the process more efficient, we chose one of the strings with the shortest length in Z_M that is not already in S_1 as the counterexample, called z_{cex}. To include the information about a counterexample in the constructed tables, the counterexample and all of its prefixes should be added to S_2, i.e., $S_2 = S_1 \cup Pre(z_{cex})$. The table should then be extended across $(S_2 \cup S_2.\Sigma).E_2$ using the function T. For example, in Figure 10.a, Z is constructed from V_1 representing its elements $s.t$, $s \in S_1 \cup S_1.\Sigma$ and $t \in E_1$, where $T'(s.t) = 0$. Membership queries of the elements in Z are then performed to form Z_M. The shortest string in Z_M not in S_1 is *aaa*. The Table V_2, shown in Figure 10.b, is updated for the elements in Z_M and strings in $Pre(z_{cex} = aaa)$ are added to S_2, and the table is extended accordingly.

Every time that a counterexample is included in the table, it should be checked for consistency and closedness, defined in (4) and (5), respectively. If the Table $V_i = (S_i, E_i, T)$ is not consistent, it means that there are two strings $s_1, s_2 \in S_i$ and $t \in E_i$, $\sigma \in \Sigma$ such that $row(s_1) = row(s_2)$ but $T(s_1\sigma.t) \neq T(s_2\sigma.t)$. To make the table consistent, we should add $\sigma.t$ to E_i and then fill the table using T. If the Table $V_i = (S_i, E_i, T)$ is not closed then it means that there is a string $t \in (S_i.\Sigma - S_i)$ where $row(t)$ does not match any $row(s)$ for all $s \in S$. To make the observation table closed, the string t must be added to S_i and the table must be extended and filled accordingly. For example, in Figure 10.d, V_4 is not closed because $row(aab)$ does not match any row in S_4. To make it closed, we add $row(aab)$ to S_5 and extend the table over $(S_5 \cup S_5.\Sigma).E_5$ accordingly.

Lemma 6: A Table $V_i = (S_i, E_i, T)$ can be made closed within a finite number of iterations.

Proof: Assume that the Table V_i is not closed as there is a string $t \in (S_i.\Sigma - S_i)$ where $row(t) \neq 0$ and $row(t) \neq row(s)$ for all $s \in S_i$. To fix this problem, the string t should be added to S_i. This process therefore adds a new distinguished row t to S_i. Recall in Lemma 3 that for V', the number of distinguished non-zero rows of S' is bounded by $|X'|$. This same statement is similarly applicable to V where the number of distinguished non-zero rows of S is bounded by $|X|$. Therefore, the closedness needs to be checked with this procedure at most for $|X|$ times. ■

Once the Table $V_i = (S_i, E_i, T)$ is both consistent and closed, we are then able to use the table to build a corresponding automaton $G(V_i) = (X_i, \Sigma, \delta_i, x_{0i})$, where:

$$\begin{aligned} X_i &= \{row(s) \mid s \in S_i, T(s) = 1\} \\ \delta_i(row(s), \sigma) &= row(s.\sigma) \\ x_{0i} &= row(\varepsilon) \end{aligned} \tag{9}$$

Once the automaton $G(V_i)$ is constructed, the algorithm makes an equivalence query to the oracle. In the case where $\mathcal{L}(G(V_i)) \neq \mathcal{L}(G)$, the oracle provides a counterexample, *cex*. In this situation, the counterexample and all of its prefixes should be added to S_{i+1}, and the table must again be checked for consistency and closedness. Otherwise, if $\mathcal{L}(G(V_i)) = \mathcal{L}(G)$, the algorithm returns $G_t = G(V_i)$ as the learned DES system.

Algorithm 2 details the entire process for learning the unknown part of the system. The algorithm is initiated with V' (the information about the known part of the system), and then forms a series of Tables $V_i = (S_i, E_i, T)$. First the algorithm forms the set Z (the set of zero elements) and finds those elements that should be updated, Z_M. Next, z_{cex}, a string with the shortest length in Z_M, is found and then z_{cex} and all its prefixes are added to S_2. Then, the rest of information about the unknown part of the system is explored in two loops. The inner-loop checks for completeness (closedness and consistency) of the tables. Once a Table V_i is made complete, then an automaton $G(V_i)$ will be constructed and then in the outer-loop, an equivalence query is made to the oracle. If $\mathcal{L}(G(V_i)) \neq \mathcal{L}(G)$, the oracle returns a counterexample *cex*. The counterexample and all its prefixes should be added to S_{i+1} and the table should be checked again for completeness. If the oracle returns the equivalence query with "Yes", then

Algorithm 2 Active-Learning of the Unknown Information

input: The complete Table V' for the known part, and the event set Σ
output: The complete automaton G_t with $\mathcal{L}(G_t) = \mathcal{L}(G)$
Set $V_1 = (S_1, E_1, T) = V'$
Find $Z = \{s.t \mid s \in (S_1 \cup S_1.\Sigma),\ t \in E_1,\ T'(s.t) = 0\}$
Form $Z_M = \{z \mid T(z) = 1\}$ using membership queries and update the table with $T(z) = 1$ for elements $z \in Z_M$.
Find $z_{cex} = \arg\min_{z \in Z_M} |z| \mid z \notin S_1$
Set $S_2 = S_1 \cup Pre(z_{cex})$ and $E_2 = E_1$
Fill the Table V_2 over $(S_2 \cup S_2.\Sigma).E_2$ using membership queries
while $V_i(S_i, E_i, T)$ is not complete **do**
    if V_i is not consistent **then**
        Find $s_1, s_2 \in S_i$, $\sigma \in \Sigma$ and $e \in E_i$ such that $row(s_1) = row(s_2)$ but $T(s_1.\sigma.e) \neq T(s_2.\sigma.e)$;
        Set $E_{i+1} = E_i \cup \sigma.e$ and $S_{i+1} = S_i$;
        Set $i = i + 1$;
        Form the new Table V_i and fill it up over $(S_i \cup S_i.\Sigma).E_i$ using membership queries;
    end if
    if V_i is not closed **then**
        Find $s_1 \in S$ and $\sigma \in \Sigma$ such that $row(s_1.\sigma)$ is different from $row(s)$ for all $s \in S_i$;
        Set $S_{i+1} = S_i \cup \{s_1.\sigma\}$, $E_{i+1} = E_i$;
        Set $i = i + 1$;
        Form the new Table V_i and fill it up over $(S_i \cup S_i.\Sigma).E_i$ using membership queries;
    end if
end while
Construct the automaton $G(V_i)$
Ask equivalence query if $\mathcal{L}(G(V_i)) = \mathcal{L}(G)$.
if The oracle replies with the counterexample *cex* **then**
    Set $S_{i+1} = S_i \cup Pre(cex)$, $E_{i+1} = E_i$;
    Set $i = i + 1$;
    Form the new Table V_i and fill it up over $(S_i \cup S_i.\Sigma).E_i$ using membership queries;
    Go to *line* 8.
end if
return: $G_t = G(V_i)$

the constructed automaton G_t will be returned as a DES model which captures the information about the system G, as proven in next theorem:

Theorem 2: Algorithm 2 returns G_t after a finite number of iterations, such that $\mathcal{L}(G_t) = \mathcal{L}(G)$.

Proof: Algorithm 2 has two loops. In the inner-loop, the algorithm checks for completeness (closedness and consistency) of the table. In Lemmas 4 and 6, it was shown that the process for making a table consistent and closed will be terminated after a finite number of iterations. For the outer-loop, the algorithm checks for equivalence of the constructed model and the original system. If there is a mismatch, the algorithm returns a counterexample. The counterexample, along with all its prefixes, will be added to S_{i+1}, adding new non-zero rows in the table (states in G_t), so that the updated table can take into account the information about the counterexample. Each time that the outer-loop handles a counterexample, at least one distinguished non-zero row will be added to S_i. Based on Lemma 3, the number of distinguished non-zero rows in S_i is bounded. Hence, the outer-loop cannot continue returning the counterexamples forever, and after a finite number of iterations, the outer-loop terminates, returning the equivalence query with "Yes." ■

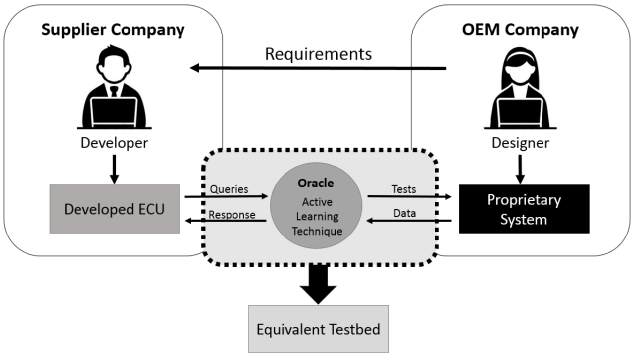


FIGURE 3. The process of testing ECU-OEM integration described in Section IV through interactions between the OEM and Supplier Companies.

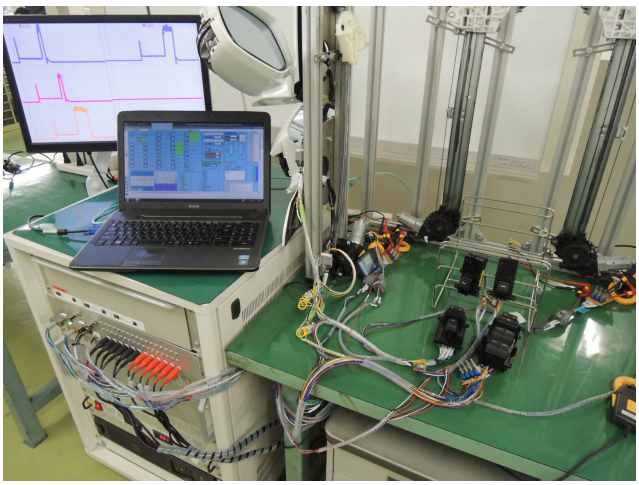


FIGURE 4. A test stand for testing integration of an ECU with an OEM system [34].

C. COMPUTATION REDUCTION

The proposed algorithm takes advantage of information about the known part of the system, and reduces the number of queries to the oracle to identify the unknown part. Therefore, we quantify the computation reduction resulted from the proposed algorithm in terms of the reduction in the number of queries. Before calculating the computation reduction, we need the following definition and lemmas.

Definition 4: The deterministic automaton $G' = (X', \Sigma, \delta', x_0)$ is the canonical recognizer of $L(G)$, if $L(G') = L(G)$ and for any other automaton $G''(X'', \Sigma, \delta'', x_0)$ with $L(G'') = L(G)$, we have $|X'| \leq |X''|$.

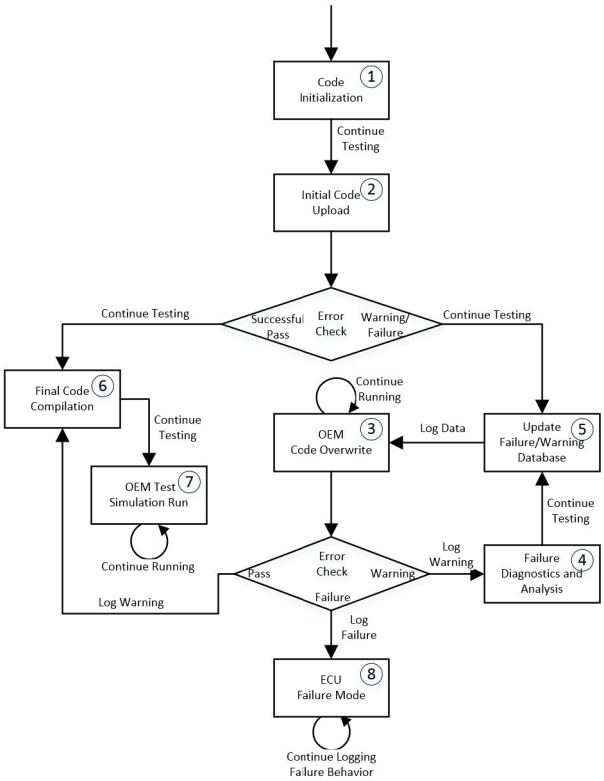


FIGURE 5. The flowchart describing the procedure for testing the integration of an ECU and an OEM system.

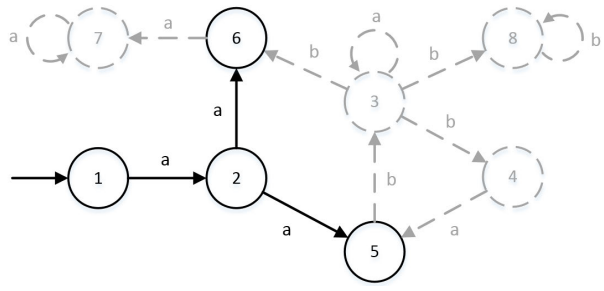


FIGURE 6. A partially-known automaton, G, where its known part is represented by the subautomaton G', shown by solid arrows and circles, and the unknown portion of the system is represented by gray dashed lines.

Lemma 7: Consider $G'' = (X'', \Sigma, \delta'', x_0)$ as the canonical recognizer of $L(G')$. The number of distinct rows in V', resulted from Algorithm 1, is at least $n'' = |X''|$.

Proof: By construction, G' is consistent with T', i.e., for any $s \in (S' \cup S'.\Sigma)$ and $e \in E'$, $\delta'(x_0, s.e)$ is defined if and only if $T'(s.e) = 1$. Since G'' and G' are language-equivalent, if $\delta'(x_0, s.e)$ is defined, then $\delta''(x_0, s.e)$ is also defined. Now, assume two distinguished states q_1 and q_2 in G'', that are reachable from x_0 by $s_1, s_2 \in Pre(\Sigma^\ell)$, i.e., $\delta''(x_0, s_1) = q_1$ and $\delta''(x_0, s_2) = q_2$. Since G'' is the canonical recognizer of $L(G')$, there exists $e \in \Sigma$ such that $\delta''(\delta''(x_0, s_1), e) \neq \delta''(\delta''(x_0, s_2), e)$, which is the equivalent of saying $\delta''(x_0, s_1.e) \neq \delta''(x_0, s_2.e)$. By construction, $S' = Pre(\Sigma^\ell)$ contains s_1 and s_2. Since, V' is consistent,

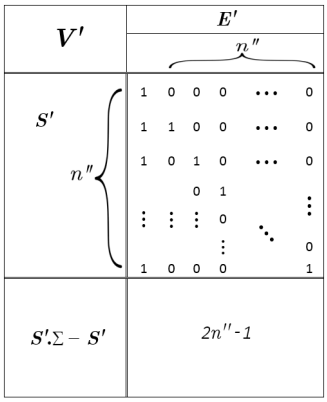


FIGURE 7. A Table V' containing n'' distinguished rows, which can be distinguished by at least $2n'' - 1$ 1-elements.

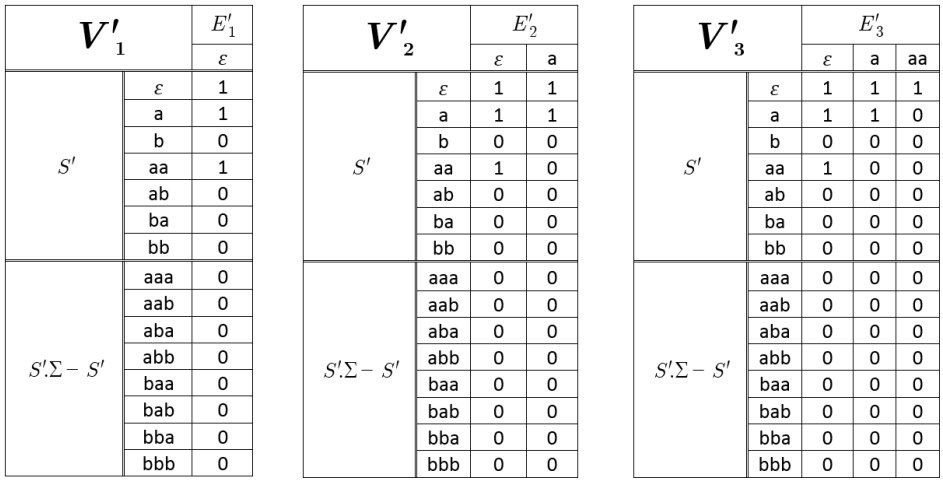
and since for $e \in \Sigma$ we have $\delta''(x_0, s_1.e) \neq \delta''(x_0, s_2.e)$, there exists $e' \in E'$ such that $\delta''(x_0, s_1.e') \neq \delta''(x_0, s_2.e')$, concluding that $row(s_1) \neq row(s_2)$, resulting in two distinct rows in S'. Therefore, V' has at least $n'' = |X''|$ distinct rows. ■

Theorem 3: Consider $G'' = (X'', \Sigma, \delta'', x_0)$ as the canonical recognizer of $L(G')$, where $n'' = |X''|$. Compared to the original L^* algorithm, Algorithms 1 and 2 reduce the number of membership queries by at least $2n'' - 1$.

Proof: Given a canonical recognizer G'', we can see from Lemma 3 that in the worst case scenario, there will be n'' distinguished non-zero rows in S'. In order to distinguish each distinct non-zero row with minimum number of 1-elements, at most n'' columns are needed in V', with each subsequent column after the first column containing a single 1-element (See Fig. 7). Hence, Table V' contains at least $2n'' - 1$ of 1-elements. According to Algorithms 1 and 2, to identify the unknown part of the system, we only need to inquire about the 0-elements, as the 1-elements are already considered known information. Therefore, equivalent to the number of 1-elements, i.e., $2n'' - 1$, the number of queries will be reduced from the original L^* algorithm, which inquires about all parts of the system (known and unknown). ■

IV. ILLUSTRATIVE EXAMPLE

The automotive industry has grown in a way that currently, approximately 90% of all recent vehicle innovations depend upon Electronic Control Units (ECUs) and their software [35]. With software becoming a major force of innovation, original equipment manufacturers (OEMs) and suppliers have had to grow and place more focus on system integration [36]. An example of such an occurrence would be where an ECU Supplier needs to test their ECU with an OEM's system. Due to intellectual property (IP) rights, OEMs are not in a position to reveal the complete model of their systems, but allow the ECU Supplier to perform black-box testing of the ECU while integrated with the OEM's system and then return the test results back to the Supplier's developers [37]. This process is shown in Figure 3, through which the Supplier Company's ECU software makes queries to the OEM's



V'_1		E'_1
		ε
S'	ε	1
	a	1
	b	0
	aa	1
	ab	0
	ba	0
	bb	0
$S'.\Sigma - S'$	aaa	0
	aab	0
	aba	0
	abb	0
	baa	0
	bab	0
	bba	0
	bbb	0

(a)

V'_2		E'_2	
		ε	a
S'	ε	1	1
	a	1	1
	b	0	0
	aa	1	0
	ab	0	0
	ba	0	0
	bb	0	0
$S'.\Sigma - S'$	aaa	0	0
	aab	0	0
	aba	0	0
	abb	0	0
	baa	0	0
	bab	0	0
	bba	0	0
	bbb	0	0

(b)

V'_3		E'_3		
		ε	a	aa
S'	ε	1	1	1
	a	1	1	0
	b	0	0	0
	aa	1	0	0
	ab	0	0	0
	ba	0	0	0
	bb	0	0	0
$S'.\Sigma - S'$	aaa	0	0	0
	aab	0	0	0
	aba	0	0	0
	abb	0	0	0
	baa	0	0	0
	bab	0	0	0
	bba	0	0	0
	bbb	0	0	0

(c)

FIGURE 8. Constructing a complete table for G' as the known part of system G, given in Figure 6: (a) V'_1 is initialized and constructed by setting $S' = Pre(\Sigma^{\ell})$ and $E'_1 = \varepsilon$ and then V'_1 is filled using T' accordingly, (b) V'_1 is not consistent, which is fixed by setting $E'_2 = E'_1 \cup \{a\}$, and then, V'_2 is filled accordingly, (c) V'_2 is not consistent, which is fixed by setting $E'_3 = E'_2 \cup \{aa\}$ and then V'_3 is filled accordingly.

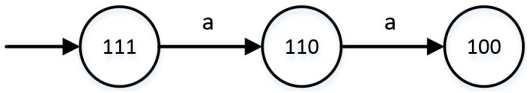


FIGURE 9. The automaton $G(V')$ constructed by Equation (6) from Table V'_3 located in Figure 8.c.

system and the OEM Company conducts relevant tests and replies to the queries.

In this section, partially knowing the software system model of the integrated system of an ECU connected to an OEM test system, we employ the proposed technique in this paper to actively learn the behavior of the entire integrated system through interactions with the OEM Company which answers the queries about the proprietary system.

A. THE STRUCTURE OF AN ECU INTEGRATED WITH AN OEM SYSTEM

Consider an ECU Supplier needing to test their ECU integrated with an OEM through a setup shown in Figure 4 in order to confirm accurate system integration. A simplified flowchart for ECU software system integration into an OEM's system is shown in Figure 5. Many OEMs test system integration in different ways. An example of an integrated testing process is described in Figure 5, which begins with the ECU Supplier initializing and uploading the code to the ECU (States 1 and 2, respectively). If the upload is successful, then the Supplier will immediately be able to compile the ECU testing simulation code (State 6) and begin to run the simulation tests (State 7). However, if a problem occurs with the upload of initial code creating a warning or failure, the system then logs that failure and updates the failure logging database (State 5) as the OEM overwrites their code to the ECU (State 3) in hopes of rectifying the failure. The code is then again checked for errors and if it passes, the system will proceed to testing and simulation (States 6 and 7). If the code raises a warning flag, the system logs the error for analysis (State 4) and proceeds to update the failure/warning database. If the system completely fails, the ECU defaults into a complete failure mode (State 8) and the system continues to monitor the faulty behavior of the ECU. The process in Figure 4 can be modeled by the automaton G, shown in Figure 6, with the event set $\Sigma = \{a, b\}$, where 'a' and 'b' represent "continue testing" and "log data/failure/warning," respectively.

Due to proprietary reasons, the ECU Supplier has no knowledge of the OEM's system and only knows their part of the ECU system, shown by solid circles (states) and arrows (transitions), captured by the subautomaton G'. The OEM's proprietary part (unknown to the ECU Supplier) is shown by gray dashed arrows and circles in Figure 6. The Supplier, however, has the opportunity to test the ECU system and make queries to the OEM company for the executed tests as described in Figure 3. For this purpose, we apply the proposed algorithm in this paper to capture the known part of the system and actively learn the unknown part, as described in the next subsections.

B. CAPTURING THE INFORMATION ABOUT THE KNOWN PART (ECU Side) OF THE ECU-OEM INTEGRATED SYSTEM

Starting with Algorithm 1, in order to capture the known information and using (3), we can find $\ell = 2$ as the length of the longest acyclic path in G', for which we can form $Pre(\Sigma^{\ell}) = \{\varepsilon, a, b, aa, ab, ba, bb\}$. Then, we initialize V'_1 by setting $S'_1 = Pre(\Sigma^{\ell})$ and $E'_1 = \{\varepsilon\}$, and then, we fill the Table V'_1 using T' accordingly. It can be verified that the Table V'_1 is not consistent as $row(a) = row(aa)$ but $row(aa) \neq row(aaa)$ as $T'(a.\varepsilon.a) \neq T'(aa.\varepsilon.a)$, and hence

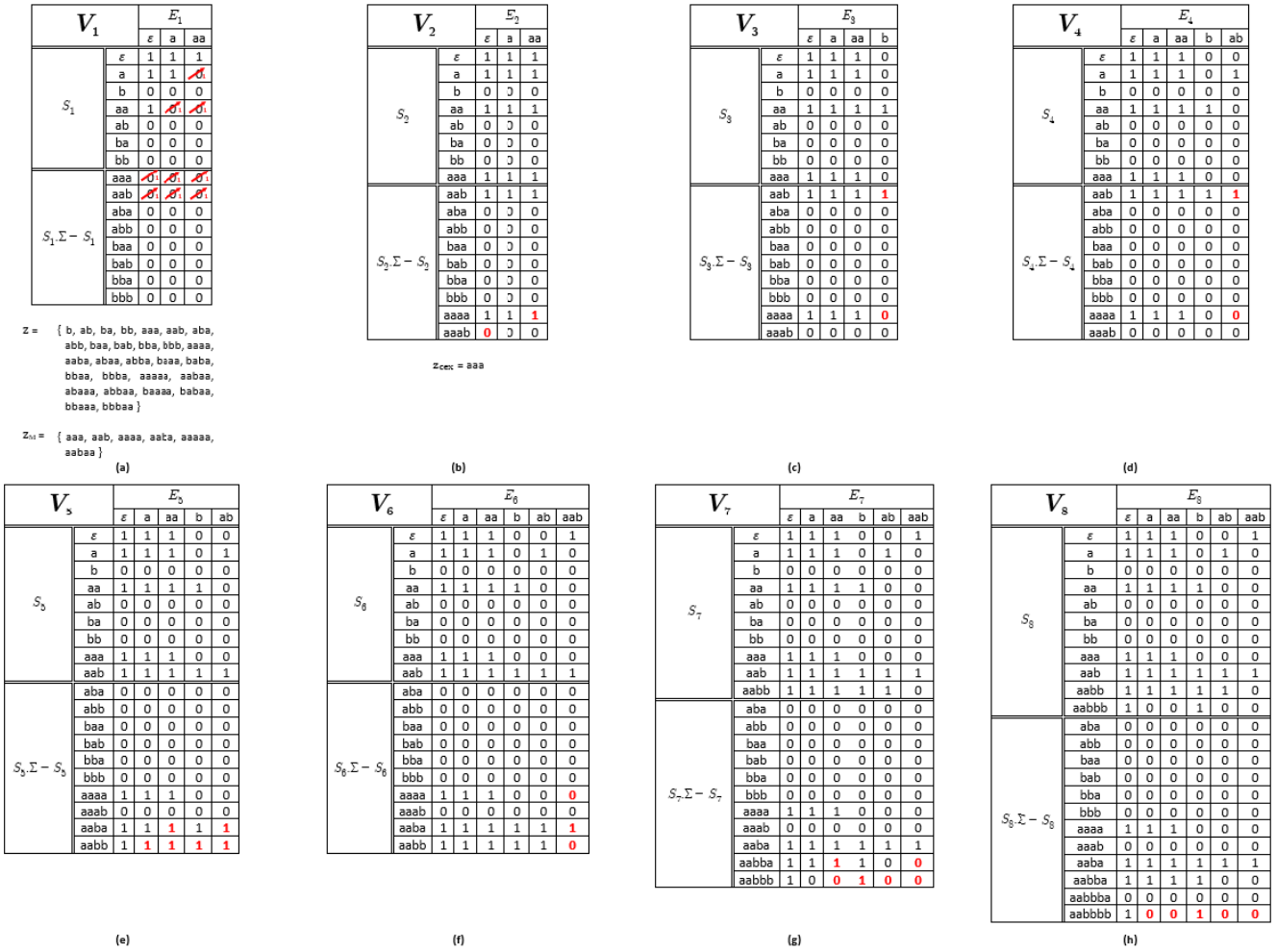


FIGURE 10. Constructing a complete table for the partially known system G, given in Figure 6: **(a)** $V_1 = V'$, Z includes the zero elements of V', membership queries of the elements in Z are then performed to form Z_M, and V_1 is updated, **(b)** z_{cex} is found to be $\{aaa\}$ so the strings in $Pre(z_{cex} = aaa)$ are added to S_2, and V_2 is updated, **(c)** V_2 is not consistent so in V_3, we set $E_3 = E_2 \cup \{b\}$ and then V_3 is filled accordingly, **(d)** V_3 is not consistent so in V_4, we set $E_4 = E_3 \cup \{ab\}$ and then V_4 is filled accordingly, **(e)** V_4 is not closed so in V_5, abb is added to S_5 and then V_5 is extended and filled accordingly, **(f)** V_5 is not consistent so in V_6, we set $E_6 = E_5 \cup \{aab\}$ and then V_6 is filled accordingly, **(g)** V_6 is not closed so in V_7, $aabb$ is added to S_7 and then V_7 is extended and filled accordingly, **(g)** V_7 is not closed so in V_8, $aabbb$ is added to S_8 and then V_8 extended and filled accordingly, which becomes a complete table. The membership queries are shown in red.

we set $E_2' = E_1' \cup \{a\}$ in V_2' and then fill the Table V_2' accordingly. V_2' is still not consistent as $row(\varepsilon) = row(a)$ but $row(\varepsilon.a.a) \neq row(a.a.a)$ as $T'(\varepsilon.a.a) \neq T'(a.a.a)$, and so we must set $E_3' = E_2' \cup \{aa\}$ in V_3' and then fill the Table V_3' accordingly. $V' = V_3'$ is now complete (closed and consistent). This process can be followed in Figure 8.

C. LEARNING THE UNKNOWN PART (OEM Side) OF THE ECU-OEM INTEGRATED SYSTEM

The Table V', containing the information regarding the known part of the system, can now be used for the initialization of the active-learning Algorithm 2 for the unknown part of the system G. We begin by setting $V_1 = V'$ and gathering all zero elements in V_1 to construct the set Z. Then, $Z_M = \{aaa, aab, aaaa, aaba, aaaaa, aabaa\}$ is constructed using membership queries on the Z elements. Then, V_2 is constructed by including $z_{cex} = aaa$, a string with the shortest length in Z_M, as a counterexample. For this purpose, $z_{cex} = aaa$ and its prefixes are added to S_2, forming $S_2 = S_1 \cup Pre(z_{cex})$, and then, Table V_2 is extended across $(S_2 \cup S_2.\Sigma).E_2$ using the function T. We can see that Table V_2 is not consistent as $row(a) = row(aa)$ but $row(a.b) \neq row(aa.b)$ as $T(a.b.\varepsilon) \neq T(aa.b.\varepsilon)$, which is fixed by setting $E_3 = E_2 \cup \{b\}$ in V_3 and then filling the Table V_3 accordingly. Table V_3 is not consistent since $row(a) = row(aaa)$ but $row(a.ab) \neq row(aaa.ab)$ as $T(a.a.b) \neq T(aaa.a.b)$, which is fixed by setting $E_4 = E_3 \cup \{ab\}$ in V_4 and then filling the Table V_4 accordingly. Table V_4 is not closed as $row(aab)$ in $(S_4.\Sigma - S_4)$ is not in S_4, which is fixed by setting $S_5 = S_4 \cup \{aab\}$ and then extending and filling S_5 accordingly. Table V_5 is not consistent since $row(\varepsilon) = row(aaa)$ but $row(\varepsilon.aab) \neq row(aaa.aab)$ as $T(\varepsilon.a.ab) \neq T(aaa.a.ab)$, which is fixed by setting $E_6 = E_5 \cup \{aab\}$ in V_6 and then filling the Table V_6 accordingly. Table V_6 is not closed as $row(aabb)$ in

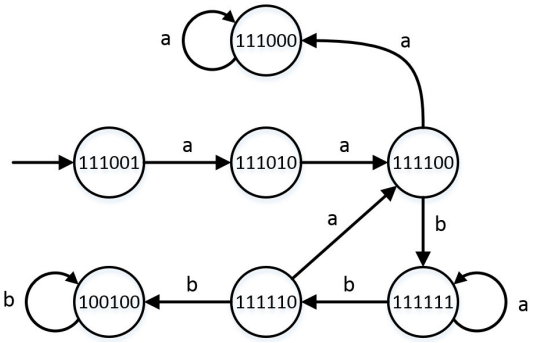


FIGURE 11. The automaton G_t, constructed for the complete Table V_8 in Figure 10.h. The automaton G_t is language-equivalent to the original system G, given in Figure 6.

$(S_6.\Sigma - S_6)$ is not in S_6, which is fixed by setting $S_7 = S_6 \cup \{aabb\}$ and then extending and filling S_7 accordingly. Table V_7 is not closed as $row(aabbb)$ in $(S_7.\Sigma - S_7)$ is not in S_7, which is fixed by setting $S_8 = S_7 \cup \{aabbb\}$ and then extending and filling S_8 accordingly. This process can be followed in Figure 10. The Table V_8 in Figure 10 is now complete (closed and consistent) for which $G_t = G(V_8)$ is constructed and shown in Figure 11 which is language-equivalent to G. While the Supplier may not have exact insight into the OEM's system, they would still be able to test proper integration methods based upon a language equivalent model that would function similarly to the true ECU-OEM integrated system.

V. CONCLUSION

In this paper, a novel method was developed to learn a partially-known DES system. The developed method first captures the known part of the system in a tabular form, and then utilizes a novel active-learning technique to discover the unknown part of the system and construct a language-equivalent model through (*membership* and *equivalence*) queries made to an oracle. The information is collected in a series of tables until the table is complete and the equivalence query is replied by *"Yes"*. It was proven that the algorithms will terminate after a finite number of iterations and eventually returns a correct automaton which is language-equivalent to the original system. It was also proven that the proposed algorithms are able to reduce the number of membership queries.

ACKNOWLEDGMENT

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, AFRL, OSD, DoED, or the U.S. Government.

REFERENCES

[1] B. Hrúz and M. Zhou, *Modeling and Control of Discrete-event Dynamic Systems: With Petri Nets and Other Tools*. London, U.K.: Springer, 2007.
[2] R. David and H. Alla, "Petri nets for modeling of dynamic systems: A survey," *Automatica*, vol. 30, no. 2, pp. 175–202, 1994.
[3] A. Giua and M. Silva, "Petri nets and automatic control: A historical perspective," *Annu. Rev. Control*, vol. 45, pp. 223–239, Jun. 2018.
[4] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 1, pp. 251–260, Jan. 2018.
[5] N. Ran, H. Su, and S. Wang, "An improved approach to test diagnosability of bounded Petri nets," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 2, pp. 297–303, Apr. 2017.
[6] S. Wang, D. You, and M. Zhou, "A necessary and sufficient condition for a resource subset to generate a strict minimal siphon in s 4PR," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 4173–4179, Aug. 2017.
[7] Y. Shi, C. Tian, Z. Duan, and M. Zhou, "Model checking Petri nets with MSVL," *Inf. Sci.*, vol. 363, pp. 274–291, Oct. 2016.
[8] M. Wang, G. Liu, P. Zhao, C. Yan, and C. Jiang, "Behavior consistency computation for workflow nets with unknown correspondence," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 1, pp. 281–291, Jan. 2018.
[9] J. Komenda, S. Lahaye, J.-L. Boimond, and T. van den Boom, "Max-plus algebra and discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 1784–1790, 2017.
[10] J. C. M. Baeten, "A brief history of process algebra," *Theor. Comput. Sci.*, vol. 335, nos. 2–3, pp. 131–146, May 2005.
[11] J. Sakarovitch, R. Thomas, and R. Thomas, *Elements Automata Theory*, vol. 6. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[12] R. Kumar, *Theory of Automata, Languages & Computation*. New York, NY, USA: McGraw-Hill, 2010.
[13] J. E. Hopcroft, *Introduction to Automata Theory, Languages, and Computation*. London, U.K.: Pearson, 2008.
[14] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. New York, NY, USA: Springer, 2009.
[15] K. C. Wong and W. M. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 14, no. 1, pp. 55–107, Jan. 2004.
[16] F. Cassez, "Dynamic observers for fault diagnosis of timed systems," in *Proc. 49th IEEE Conf. Decis. Control (CDC)*, Dec. 2010, pp. 4359–4364.
[17] J. Luo and M. Zhou, "Petri-net controller synthesis for partially controllable and observable discrete event systems," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1301–1313, Mar. 2017.
[18] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for discrete event systems," *Annu. Rev. Control*, vol. 37, no. 2, pp. 308–320, Dec. 2013.
[19] S. Lafortune, "Diagnosis of discrete event systems," *Encyclopedia Syst. Control*, pp. 268–275, Feb. 2014.
[20] A. White, A. Karimoddini, and R. Su, "Fault diagnosis of discrete event systems under unknown initial conditions," *IEEE Trans. Autom. Control*, vol. 64, no. 12, pp. 5246–5252, Dec. 2019.
[21] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*. Springer, 2015.
[22] W. M. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history," *Annu. Rev. Control*, vol. 45, pp. 250–256, 2018.
[23] Z. Jiang, Z. Li, N. Wu, and M. Zhou, "A Petri net approach to fault diagnosis and restoration for power transmission systems to avoid the output interruption of substations," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2566–2576, Sep. 2018.
[24] X. Guo, S. Wang, D. You, Z. Li, and X. Jiang, "A siphon-based deadlock prevention strategy for S3PR," *IEEE Access*, vol. 7, pp. 86863–86873, 2019.
[25] C. De la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge, U.K.: Cambridge Univ. Press, 2010.
[26] K. S. Narendra and M. A. Thathachar, *Learning Automata: An Introduction*. Chelmsford, MA, USA: Courier Corporation, 2012.
[27] O. Maler and I. E. Mens, "Learning regular languages over large alphabets," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer, 2014, pp. 485–499.
[28] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987.
[29] K. P. Murphy, "Passively learning finite automata," Santa Fe Inst., Santa Fe, NM, USA, Tech. Rep. 96-04-017, 1996.
[30] E. M. Gold, "Complexity of automaton identification from given data," *Inf. Control*, vol. 37, no. 3, pp. 302–320, Jun. 1978.
[31] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of NFA," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, 2009, pp. 1004–1009.

[32] R. L. Rivest and R. E. Schapire, "Inference of finite automata using homing sequences," *Inf. Comput.*, vol. 103, no. 2, pp. 299–347, Apr. 1993.
[33] X. Zhang, B. Wu, and H. Lin, "Supervisor synthesis of POMDP based on automata learning," 2017, *arXiv:1703.08262*. [Online]. Available: http://arxiv.org/abs/1703.08262
[34] *Automated Testing of Electronic Control Units (ECU)*. Accessed: Aug. 2018. [Online]. Available: https://www.imc-tm.com/fileadmin/Public/Downloads/Application_notes/App-Notes-ENG/APP_eng_Automated_Testing_of_ECUs.pdf
[35] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proc. Workshop Joining AcadeMiA Ind. Contrib. Test Autom. Model-Based Test. (JAMAICA)*, 2014, pp. 1–6.
[36] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *Proc. Future Softw. Eng. (FOSE)*, May 2007, pp. 55–71.
[37] K. Kim, K.-Y. Choi, and J.-W. Lee, "Fault localization method by partitioning memory using memory map and the stack for automotive ECU software testing," *Appl. Sci.*, vol. 6, no. 9, p. 266, 2016.

IRA WENDELL BATES, II, received the bachelor's degree in computer and electrical engineering and the M.Sc. degree in electrical engineering from Florida Agricultural and Mechanical University, in 2008 and 2011, respectively. He then joined industry and worked for BMW Manufacturing Co., as a Quality Engineer, from 2011 to 2016. He is currently pursuing the Ph.D. degree with North Carolina A&T State University. He is also a member of the Autonomous Cooperative Control of Emergent Systems of Systems (ACCESS) Laboratory and the Testing, Evaluation, and Control of Heterogeneous Large-Scale Systems of Autonomous Vehicles (TECHLAV) Center. His research interests include reliable control systems, hybrid control systems, discrete event systems, and robotics.

ALI KARIMODDINI (Senior Member, IEEE) received the bachelor's degree in electrical and electronics engineering from the Amirkabir University of Technology, in 2003, the M.Sc. degree in instrumentation and automation engineering from the Petroleum University of Technology, in 2007, and the Ph.D. degree from the National University of Singapore, in 2012. He then joined the University of Notre Dame to conduct his postdoctoral studies. He is currently an Associate Professor with the ECE Department, N. C. A&T State University. He is also the Director of the NC-CAV Center of Excellence on Advanced Transportation, the Director of the ACCESS Laboratory, and the Deputy Director of the TECHLAV DoD Center of Excellence. His research interests include cyber-physical systems, control and robotics, resilient control systems, flight control systems, multiagent systems, and human–machine interactions. His research has been supported by different federal funding agencies and industrial partners. He is a member of AIAA, ISA, and AHS.

MOHAMMAD KARIMADINI received the bachelor's degree in electrical and electronics engineering from the Azad University of Tehran, Iran, in 1996, the M.Sc. degree in control and automation from the UPM university of Malaysia, in 2006, and the Ph.D. degree in control engineering from the National University of Singapore (NUS), in 2012, where he followed by two Postdoctoral fellowships. He then joined industry and had worked as an Engineer and a Supervisor with the Instrumentations and Control Department, National Iranian Copper Industries Company (NICICO), from 1996 to 2004. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Arak University of Technology (ARAKUT), and the Vice-President of Technology and Innovation with Arak Science and Technology Park (ASTP). His research interests include supervisory control of discrete event systems, decentralized cooperative control of multiagent systems, industrial automation, collective intelligence, and management of technology.