IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 12, 2020, accepted May 30, 2020, date of publication June 8, 2020, date of current version June 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3000907

Survey on Learning-Based Formal Methods: Taxonomy, Applications and Possible Future Directions

FUJUN WANG[1], ZINING CAO[1,2,3,4], LIXING TAN[1], AND HUI ZONG[1]

[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2]Key Laboratory of Safety-Critical Software, Ministry of Industry and Information Technology, Nanjing 210023, China
[3]MIIT Key Laboratory of Pattern Analysis and Machine Intelligence, Nanjing University of Aeronautics and Astronautics, Nanjing 210023, China
[4]Science and Technology on Electro-optic Control Laboratory, Luoyang 471023, China

Corresponding author: Fujun Wang (wangfj@nuaa.edu.cn)

This work was supported in part by the National Key Fundamental Research and Development Plans (973 Plans) under Grant 2014CB744903, in part by the Aviation Science Foundation of China under Grant 20150652008 and Grant 20185152035, in part by the Fundamental Research Funds for the Central Universities under Grant NZ2013306, in part by the National Natural Science Foundation of China under Grant 61303022 and Grant 61572253, and in part by Foundation under Grant 61400020404.

ABSTRACT Formal methods play an important role in testing and verifying software quality, especially in modern society with rapid technological updates. Learning-based techniques have been extensively applied to learn (a model or model-free) for formal verification and to learn system specifications, and resulted in numerous contributions. Due to the fact that adequate system models are often difficult to design manually and manual definition of specifications for such software systems gets infeasible, which motivate new research directions in learning models and/or specifications from observed system behaviors automatically. This paper mainly concentrates on learning-based techniques in formal methods area. An up-to-date overview of the current state-of-the-art in learning-based formal methods is provided in the paper. This paper is not a comprehensive survey of learning-based techniques in formal methods area, but rather as a survey of the taxonomy, applications and possible future directions in learning-based formal methods.

INDEX TERMS Formal methods, formal specification, formal verification, learning model, learning specification.

I. INTRODUCTION

Software reliability mainly depends on two aspects: one is the method and process of software development, the other is the test and verification of software products. In many engineering practices, the design and development of software products still lack solid scientific basis and mature methodology. It is thus not surprising that seeking for new techniques to ensure the quality of the software production process is still a nontrivial task. Formal methods are new disciplines, which attempt to accompany the development with techniques and tools for finding and pointing out potential problems. Formal methods are a collection of notations and techniques for describing and analyzing systems [1]. These methods are formal in the sense that they are based on some mathematical theories, such as logic, automata, or graph theory. They are aimed at enhancing the quality of systems.

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

Formal methods include three main parts: system modeling, formal specification, and formal verification (FV).

A. PROBLEM STATEMENT

Traditionally, models used in model-checking are manually constructed, it is time-consuming and error-prone, especially for systems lacking updated and detailed documentations, such as legacy software, 3rd party components, and black-box systems. These difficulties are generally considered as a hindrance for adopting otherwise powerful model checking [2], [3] techniques, and have led to the emergence of a new research direction by using learning techniques in formal methods to learn a model from system observations. Similarly, in practice, often no formal specifications are available, or become outdated as the system evolves over time, which make it difficult to understand and analyze the behavior of the system. To address this issue, we can also use learning techniques to learn the specifications that the system meets from system observations.

B. MOTIVATION

Although there have been a lot of learning-based papers that are closely related to formal methods published, one can only find few comprehensive review papers that include both learning models and learning formal specifications. At the same time, there are various algorithms for learning models and learning specifications, but no one has classified and summarized the existing learning algorithms and analyzed them. The motivation of this paper is to give a survey on both learning (a model or model-free) for FV and learning formal specifications, meanwhile present taxonomy, applications, and future directions in learning-based formal methods.

C. RELATED WORK

Some early overview papers in this area were written by Leucker [4], Steffen *et al.* [5], Howar and Steffen [6], Bennaceur and Meinke [7]. In [4], some algorithms for learning automata and their recent applications to verification problems were surveyed, the author only concerned with learning an automata model or learning the language. In [5], the authors investigated current state of active automata learning research and applications of active automata learning in practice, and predicted four major topics to be addressed in the then near future: efficiency, expressivity of models, bridging the semantic gap between formal languages and the analyzed components, and solutions to the inherent problem of incompleteness of active learning in black-box scenarios. In [6], the authors reviewed the progresses that have been made over the past five years, assessed the status of active automata learning techniques with respect to applications in the field of software engineering, and presented an updated agenda for future research. In [7], an introductory survey of machine learning (ML) applications in software engineering was presented, and a classification in terms of the models they produce and the learning methods they use were given. Also, the open challenges for reaching the full potential of ML for software engineering and how ML can benefit from software engineering methods were discussed. Furthermore, two latest but unpublished survey papers appeared in this area in [8] and [9]. In [8], how ML helps FV in its classical approaches: static analysis, model-checking, theorem-proving, and Sat solving were explored. It is an attempt to provide a comprehensive survey of the various ways that ML contribute to enhance FV tools' efficiency. In [9], the authors reviewed the learning techniques, algorithms, and tools which form the basis of model learning, and provided the comparison summaries that highlight the merits and shortcomings of them. Also, the successful applications of model learning techniques in multidisciplinary fields making it promising for testing and verification of realistic systems were surveyed. But, the issues of learning formal specifications were not included in [8] and [9]. In particular, after these two survey papers, some new research results in learning specifications came out. To our knowledge, two recent closely related works on specifications are provided in [10], [11]. In [10], the authors summarized the state-of-the-art techniques for qualitative and quantitative monitoring of Cyber-Physical Systems (CPS) behaviors. However, that paper mainly concentrated on monitoring of CPS, few involved in learning. In [11], only a tutorial description on learning Signal Temporal Logic (STL) requirements from data was given. Table 1 makes a comparative study of several summary articles mentioned in this article.

TABLE 1. Comparison of related work.

Reference	Learning model	Learning specification	Application	Taxonomy
[4]	√	×	√	×
[5]	√	×	√	×
[6]	√	×	√	×
[7]	√	×	√	√
[8]	×	×	√	×
[9]	√	×	√	×
[10]	×	√	√	×
[11]	×	√	√	×

D. APPROACH

The material studied in this paper was obtained mainly from three aspects: Google scholar, dblp website (https://dblp.uni-trier.de/) and papers that have cited [12] and [13]. We queried Google scholar and dblp website using the following search terms: ''model learning formal methods'', ''model learning model checking'', ''learning specification formal methods'', ''specification mining formal methods'', ''specification mining formal verification'', ''specification inference formal methods'', ''requirements mining formal methods'', and ''learning property formal methods''. By sorting out the literature and deleting the irrelevant literature, we got the results that include journal papers, conference papers, academic dissertations, reports, and unpublished preprint papers. First, we made a comparative study of the existing literature reviews, and then classified the learning model and learning specification algorithms, respectively. Finally, these two types of algorithms were compared and studied.

E. OUR CONTRIBUTIONS

This paper focuses on learning algorithms in formal methods and presents an up-to-date overview of the current state of the art in learning (a model or model-free) for FV and learning formal specifications. Various learning algorithms were classified according to the learning style and purpose in formal methods. This paper presents the survey mainly from three perspectives: taxonomy of learning algorithms, applications of learning in formal methods, and possible future research directions. The objective of this paper is two-fold. The first is to provide a generic literature overview of the learning algorithms that have been proposed to learn formal specifications and to learn for FV. This is the first survey of taxonomy both

on learning formal specifications and on learning for verification. The second is to identify possible research directions in the area of formal methods by using ML techniques. Note that, due to the large number of ML techniques, it is not feasible to include all of them used in formal methods area in this paper. This paper only concentrates on learning formal specifications and learning for FV, and does not discuss other techniques used in formal methods. Therefore, this paper is not a comprehensive survey of the learning-based techniques in formal methods area, other learning techniques in the field of formal methods are not discussed, such as model checking based on Gaussian process [14], but rather as a survey of the taxonomy, applications and future directions in learning-based formal methods.

F. ORGANIZATION OF THE PAPER

The rest of this paper is organized as follows. Section II presents the preliminaries of some representative algorithms that were used in learning models and specifications. Taxonomy of learning algorithms in learning specifications and learning for verification are presented in section III. Section IV provides applications based on learning. It is followed by possible future research directions in section V. Finally, we conclude our presentation in Section VI.

II. PRELIMINARIES

In this section, we briefly introduce some model definitions, temporal logic, and seminal algorithms that are used in this paper.

A. MODEL

As too many models are involved in this paper, we cannot give all the definitions. Some of the basic models, like transition system (TS) [15], deterministic finite automaton (DFA) [16], non-deterministic finite automaton (NFA)[16], petri net (PN) [17], discrete-time Markov chain (DTMC) [18], Markov decision process (MDP) [18] will not be introduced here due to space limitations, we refer readers to related references for details. Next, we will give some definitions of the model that readers may not be familiar with them.

Definition 1: (Deterministic Timed Automaton, DTA [19]): A DTA is a tuple $< S, s_0, F, \Sigma, T, \Delta, X >$, where

- S is a finite set of states, $s_0 \in S$ is the initial state, and $F \subseteq S$ is a set of final states.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S \times \Delta \times X$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s_0, δ, R), where s, $s_0 \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, $\delta \in \Delta$ is the timing constraint, and $R \subseteq X$ is the set of clock resets.
- $\Delta \subseteq \{\delta = [t_1, t_2] : t_1, t_2 \in \mathcal{N}\}$ is a finite set of transition timing constraints. Constraints $\delta \in \Delta$ model the time spent in a state before the transition takes place.
- X is a finite set of clocks that record the continuous time evolution. The valuation of the clock $x \in X$ is defined by $v_t(x) : X \to \mathcal{N}$.

The Probabilistic DTA is a One-Clock Timed Automaton with probabilistic information. It allows only one clock, which is reset at each transition firing. It is defined as follows:

Definition 2: (Probabilistic Deterministic Timed Automaton, PDTA [20]): A PDTA is a 6-tuple $< S, s_0, F, P, \Sigma, T >$, where

- S is a finite set of states, where each state s includes a probability of leaving the automaton. If a state is not a final state, the probability $p = 0$, otherwise the probability is from the interval (0, 1].
- $s_0 \in S$ is the initial state.
- $F \subseteq S$ is a set of final states/ accepting states.
- Σ is a finite set of symbols.
- T is a set of transitions. A transition is represented with (s, a, δ, s_0), where δ is the clock constraint. The automaton changes from state s_i to state s_j triggered by a symbol $a \in \Sigma$ if the current clock value satisfies δ. After executing a transition, the clock c is set to 0, so that the clock starts counting time from executing this transition.
- P is a set of probability matrices, $p_{i,j}(a)$ gives the probability of taking the transition from state s_i to state s_j triggered by $a \in \Sigma$. With the probability $p(i)$ that a string ends at the state s_i, it holds that:

$$p(i) + \sum_{\forall s_i, s_j \in S, \forall a \in \Sigma} p_{i,j}(a) = 1 \tag{1}$$

As for real-world hybrid production systems, we can use Stochastic Deterministic Hybrid Automaton to represent it. The formal definition is as follows:

Definition 3: (Stochastic Deterministic Hybrid Automaton, SDHA [19]): A SDHA is a tuple $< S, s_0, \Sigma, T, \Delta, P, X, \Phi >$, where

- S is a finite set of states and $s_0 \in S$ is the initial state.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S \times \Delta$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s_0, δ), where s, $s_0 \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, and $\delta \in \Delta$ is the timing constraint.
- $\Delta \subseteq \{\delta = [t_1, t_2] : t_1, t_2 \in \mathcal{N}\}$ is a finite set of transition timing constraints. Constraints $\delta \in \Delta$ model the time spent in a state before the transition takes place.
- P is a set of probability functions with the elements $p : S \times (\Sigma \cup \{\lambda\}) \times S \times \Delta \to \mathrm{Q} \cap [0, 1]$. P includes both transition probabilities and probabilities of a string ending in a state.
- X is a finite set of clocks that record the continuous time evolution. The valuation of the clock $x \in X$ is defined by $v_t(x) : X \to \mathcal{N}$.
- Φ is a finite set of functions with elements $\theta s : \mathcal{R}^n \to \mathcal{R}^m; \forall s \in S, n, m \in \mathcal{N}$. I.e. $y = \theta s(t, u)$ is the function computing the value changes of the output signals

$y \in Y$ within state s based on the time t and values of continuous input signals u.

Definition 4: (Event-Recording Automaton, ERA [21]): An ERA over Σ is a tuple $< L, L_0, L^f, E >$ consisting of

- a finite set L of locations,
- a set $L_0 \subseteq L$ of start locations,
- a set L^f of accepting locations,
- a finite set E of edges.

Each edge is a quadruple (l, l', a, g) with a source location $l \in L$, a target location $l' \in L$, an input symbol $a \in \Sigma$, and a clock guard $g \in G_\Sigma$. G_Σ is the set of clock guards.

Definition 5: (Hybrid Petri Net, HPN [17]): An HPN is a sex-tuple $< P, T, Pre, Post, m_0, h >$ such that:

- $P = \{P_1, P_2, \ldots, P_n\}$ is a finite, not empty, set of places;
- $T = \{T_1, T_2, \ldots, T_m\}$ is a finite, not empty, set of transitions;
- $P \cap T = \varnothing$, i.e., the sets P and T are disjointed;
- $h : P \cap T \rightarrow \{D, C\}$, called "hybrid function," indicates for every node whether it is a discrete node (sets P^D and T^D) or a continuous node (sets P^C and T^C);
- Pre: $P \times T \rightarrow R^+$ or $\mathcal{N}$, is the input incidence mapping;
- $Post$: $P \times T \rightarrow R^+$ or $\mathcal{N}$, is the output incidence mapping;
- $m_0 : P \rightarrow R^+$ or N is the initial marking.

B. FORMAL SPECIFICATIONS

Even though there are various formal specifications (e.g., Computation Tree Logic (CTL) [18], Linear Temporal Logic (LTL) [18], Probabilistic CTL (PCTL) [18], STL [22], [23], etc.), we only present the one that is mostly used in this paper, especially in learning specifications, namely STL.

STL is a temporal logic for specifying properties of dense-time real-valued signals, which was used to reason about the future evolution of a continuous time behavior. It has been applied to the analysis of hybrid dynamical systems from various application domains such as analog and mixed signal circuits, systems biology, or CPS.

The syntax and semantics of STL we refer to [22], [23], are given as follows:

1) THE SYNTAX OF STL IS GIVEN AS FOLLOWS

$$\varphi ::= \pi^\mu \left| \neg\pi^\mu \right| \varphi \wedge \psi | \varphi \vee \psi | \Box_{[a,b]}\psi | \varphi \, \mathcal{U}_{[a,b]}\psi$$

where, π^μ is an atomic predicate $\mathcal{R}^n \rightarrow$ Bool whose truth value is determined by the sign of a function $\mu : \mathcal{R}^n \rightarrow \mathcal{R}$ and ψ is an STL formula.

2) THE SEMANTICS OF STL

The validity of a formula φ with respect to the discrete-time signal $\mathbf{x}$ at time t, noted $(\mathbf{x}, t) \vDash \varphi$ is defined inductively as follows:

$$\begin{aligned}
&(\mathbf{x}, t) \vDash \pi^\mu && \text{iff } \mu(\mathbf{x}(t)) > 0\\
&(\mathbf{x}, t) \vDash \neg\pi^\mu && \text{iff } \neg\left((\mathbf{x}, t) \vDash \pi^\mu\right)\\
&(\mathbf{x}, t) \vDash \varphi \wedge \psi && \text{iff } (\mathbf{x}, t) \vDash \varphi \wedge (\mathbf{x}, t) \vDash \psi\\
&(\mathbf{x}, t) \vDash \varphi \vee \psi && \text{iff } (\mathbf{x}, t) \vDash \varphi \vee (\mathbf{x}, t) \vDash \psi\\
&(\mathbf{x}, t) \vDash \Box_{[a,b]}\psi && \text{iff } \forall t' \in [t+a, t+b], \left(\mathbf{x}, t'\right) \vDash \psi\\
&(\mathbf{x}, t) \vDash \varphi \mathcal{U}_{[a,b]}\psi && \text{iff } \exists\, t' \in [t+a, t+b] \text{ s.t. } \left(\mathbf{x}, t'\right) \vDash \psi \wedge\\
& && \quad \forall t'' \in \left[t, t'\right] \left(\mathbf{x}, t''\right) \vDash \varphi
\end{aligned}$$

A signal $\mathbf{x} = x_0 x_1 x_2 \ldots$ satisfies φ, denoted by $\mathbf{x} \vDash \varphi$, if $(\mathbf{x}, 0) \vDash \varphi$. Additionally, we define $\Diamond_{[a,b]}\varphi = \mathrm{T}U_{[a,b]}\varphi$, so that $\mathbf{x} \vDash \Diamond_{[a,b]}\varphi$ if φ holds at some time step between a and b

3) ROBUST SATISFACTION OF STL FORMULAS

Quantitative or robust semantics define a real-valued function ρ^φ of signal $\mathbf{x}$ and t such that $(\mathbf{x}, t) \vDash \varphi \equiv \rho^\varphi(\mathbf{x}, t) > 0$. This is computed recursively from the above semantics in a straightforward manner, by propagating the values of the functions associated with each operand using min and max operators corresponding to various STL operators. The robust semantics is defined as follows:

$$\rho^{\pi^\mu}(\mathrm{x}, t) = \mu(\mathrm{x}(t)) \tag{2}$$

$$\rho^{\neg\pi^\mu}(\mathrm{x}, t) = -\mu(\mathrm{x}(t)) \tag{3}$$

$$\rho^{\varphi\wedge\psi}(\mathrm{x}, t) = \min(\rho^\varphi(\mathrm{x}, t), \rho^\psi(\mathrm{x}, t)) \tag{4}$$

$$\rho^{\varphi\vee\psi}(\mathrm{x}, t) = \max(\rho^\varphi(\mathrm{x}, t), \rho^\psi(\mathrm{x}, t)) \tag{5}$$

$$\rho^{\Box_{[a,b]}\psi}(\mathrm{x}, t) = \min_{t'\in[t+a,t+b]}\rho^\psi(\mathrm{x}, t') \tag{6}$$

$$\begin{aligned}\rho^{\varphi U_{[a,b]}\psi}(\mathrm{x}, t) = &\max_{t'\in[t+a,t+b]}\\ &\left(\min(\rho^\psi(\mathrm{x}, t'), \min_{t''\in[t,t']}\rho^\varphi(\mathrm{x}, t''))\right)\end{aligned} \tag{7}$$

C. ALGORITHMS

In what follows, we present some algorithms in learning models (the first three algorithms) and learning specifications (the last two algorithms). The first two are seminal and widely used algorithms in learning a model, the third and fourth algorithms are the work done by our team last year. The last algorithm is relatively novel, as it applies reinforcement learning to synthesis an STL formula. Other algorithms listed in Table 5 and Table 6 will not be introduced in detail here due to space limitations, interested readers can refer to related references.

1) L* LEARNING ALGORITHM

L* algorithm [12] is a pioneering learning algorithm proposed by Angluin in 1987, which was originally used to learn a DFA. The motivation of the algorithm is to identify an unknown regular language set U from a set of members and non-members of a set. The unknown regular language set is given by a Minimally Adequate Teacher (MAT). The learning process can be regarded as a cooperative game between a learner and a teacher. Based on the assumption that the teacher can answer membership queries about the set and judge whether the language described by the candidate DFA is equivalent to the target language through equivalence

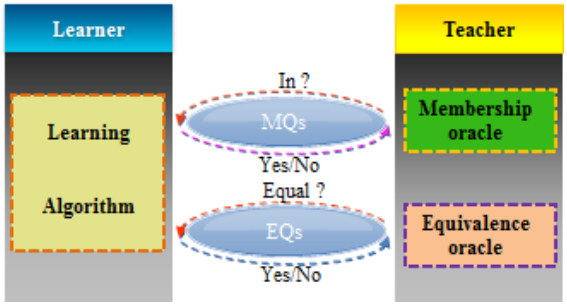

FIGURE 1. MAT model [9].

Algorithm 1 L* Learning Algorithm

Initialization: $S = E = \{\lambda\}$, x = True, y = False
Output: A candidate DFA M = $M(S, E, T)$
1: (membership query) ask for membership in each of the letters λ and Σ, construct an initial observation table (S, E, T)
2: **while** x
3:{
4: **while** x
5: {(consistency check) If the table (S, E, T) is inconsistent, then find s_1, s_2 in S, a in Σ,and e in E, so that row(s_1) = row(s_2) and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$, add $a{\cdot}e$ to E and extend T to $(S \cup S \cdot \Sigma) \cdot E$ through membership queries
6: (closure check) If the table (S, E, T) is not closed, then look for $s_1 \in S$, $a \in \Sigma$ so that row($s_1 \cdot a$) ≠ row(s) for all $s \in S$, add $s_1 \cdot$a to S, and extend T to $(S \cup S \cdot \Sigma) \cdot E$ through membership queries
7: **If** the table (S, E, T) is closed and consistent,
8: then $x = y$
9: **end if**
10: }
11: Let M = $M(S, E, T)$ be a candidate DFA,
If M is equivalent to the target regular language,
12: then $x = y$
13: **else** return counterexample t, add t and all its prefixes to S and extend T to $(S \cup S \cdot \Sigma) \cdot E$ through the membership query
14: **end if**
15:}
16:**return M**

queries, the learner can learn the target language set through two kinds of queries. The MAT model is shown in Fig.1.

Membership Queries: ask for a single word $w \in \Sigma^*$, the teacher answers the query with "yes" or "no" depends on whether the single word is in the unknown language L or not.

Equivalence Queries: ask whether a candidate language LH equals to L or not. In case a conjectured language LH does not equal to L, the teacher will provide a counterexample: a word from the symmetric difference of LH and L.

The algorithm is shown in algorithm 1.

Algorithm Description: The learner maintains an observation table (S, E, T), where S represents a nonempty prefix-closed state set, E represents a nonempty suffix-closed set and a finite function T mapping $((S \cup S \cdot \Sigma) \cdot E)$ to {0,1}. The observation table can be visualized as a two-dimensional array with rows labeled by elements of $(S \cup S \cdot \Sigma)$ and columns labeled by elements of E, with the entry for row s and column e equal to $T(s \cdot e)$. For $s \in (S \cup S \cdot \Sigma)$, $e \in E$, if $s \cdot e \in U$, then $T(s \cdot e) = 1$, else $T(s \cdot e) = 0$.

TABLE 2. Initial observation table, S = E = {λ}.

T_1	λ
λ	1
0	0
1	0

TABLE 3. Final observation table.

T_6	λ	0	1
λ	1	0	0
0	0	1	0
1	0	0	1
11	1	0	0
01	0	0	0
011	0	1	0
00	1	0	0
10	0	0	0
110	0	1	0
111	0	0	1
010	0	0	1
0110	1	0	0
0111	0	0	0

The final DFA is constructed according to the observation table when the algorithm stops. Elements on the table list S on behalf of the candidate states of the DFA, elements in E are used to distinguish between the states of the experiments, and elements in $S \cdot \Sigma$ are used to construct the transfer function.

Example (Taken From [12]): Suppose the unknown regular set U is the set of all strings over {0, 1} with an even number of 0's and an even number of 1's.

The initial observation table T_1 is shown in Table 2. This observation table is consistent, but not closed. After augmenting the observation table continuously by means of membership query and equivalence query until the algorithm stops, then the final deterministic finite automaton is constructed according to the observation table when the algorithm stops. The final observation table is shown in Table 3 and the final conjecture of L* is shown in Table 4.

2) ALERGIA ALGORITHM

The success of the L* algorithm is based on the assumption that there is a teacher who can answer membership queries

TABLE 4. The conjecture of L*.

δ	0	1
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

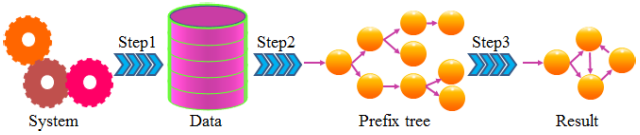

FIGURE 2. Basic ideas of ALERGIA algorithm [20].

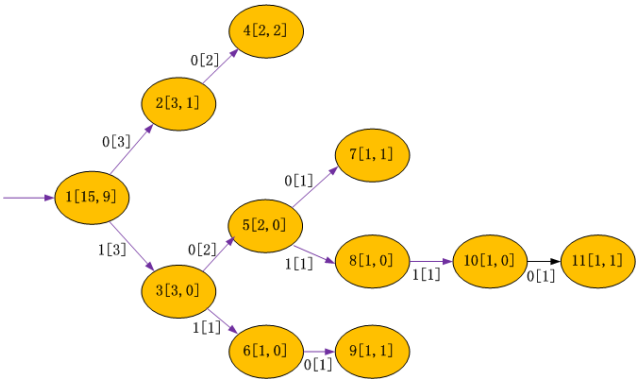

FIGURE 3. PTA for the sample S.

and equivalence queries. In real life, however, such a teacher may not exist, which hinders the wide application of L* algorithm. Several research works were proposed to solve this problem such as using neural network (NN), hidden Markov model (HMM) and other models to identify the regular language. Due to the shortcomings of these models such as long computation time and large sample set, they are not widely used. Until 1994, the emergence of ALERGIA algorithm [13] makes a significant breakthrough in solving such problems. The algorithm uses a state merging method to construct a prefix tree acceptor (PTA) based on the running samples of the system, and merges state nodes in PTA according to predefined well-orderings until stateless nodes in PTA can be merged. The basic idea of the algorithm is shown in Fig. 2.

The algorithm is shown in algorithm 2.

Example (Taken From [13]): Collect the set of sample S = {110, λ, λ, λ, 0, λ, 00, 00, λλ, λ, 10110, λ, λ, 100}.

The PTA is constructed according to the sample set S, depicted in Fig. 3. After merging, the final output is depicted in Fig. 4.

3) PAG-GA

At present, the solutions for Probabilistic Assume-Guarantee (PAG) problem, i.e., PAG-L* [24], [25] and PAG-SL*

Algorithm 2 ALERGIA Algorithm

Input: S ample set S run by the system
Output: finite automaton model T that can recognize random regular languages
1: construct PTA T according to the sample set S
2: calculate the relative frequency of output arc of each node according to the observed samples
3:**while** T is not traversed
4:{
5: **for** node = root, root+1, . . . , leaf
6: {combine states using Hoeffding bound compatibility standard according to the predefined well-ordering
7: **if** ((the relative frequency of reaching two nodes meet the Hoeffding bound) & (for any one of the alphabets, the input relative frequency of the two state nodes meet the Hoeffding bound) & (the successor nodes of the two nodes should also recursively meet the compatibility criteria))
8: **then** two state nodes combined
9: **end if**
10: eliminate uncertainty for merged nodes
11: }
12: }
13: **return** T

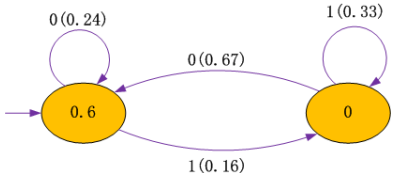

FIGURE 4. The final output.

[26], [27], can automatically generate sound and complete assumption, but the learning process must store the whole computation history in order to make continuous queries. Our team proposed a fully-automated genetic algorithm-based PAG (PAG-GA) framework for probabilistic safe property over MDP in [28]. By using GA, no intermediate results need to be recorded, so it can largely reduce the space complexity w.r.t assumption generation.

In this work, our team combined PAG reasoning framework with interface alphabet refinement to construct a small subset of the interface alphabet, then added actions to the necessary alphabet until the required property is shown to be hold or violated. Additionally, we employed the diagnostic sub-model as a counterexample for guiding the training set. Then, the assumption was generated by using GA from MDP interface alphabet, and model checking techniques were used to check the assume-guarantee tuple, and existing methods were used to construct the counterexample. In the end, the PAG problem was solved.

As the whole algorithm pseudocodes are presented in [28], we will not describe it here, interested readers can refer to [28] for detail.

4) LRx

As the complexity of the system increases, especially safety-critical systems, like CPS, it becomes hard to understand and analyze the system behavior. Researchers collect system behaviors, from which they try to analyze the properties satisfied by the system. The shortcoming of the existing algorithms is that they can only learn from positive samples of the system. In [29], our team proposed a novel learning approach, called LRx, which can deal with not only positive and negative examples, but also positive examples. This approach requires two inputs: traces of the system and a Parametric STL (PSTL) formula. The main steps of synthesizing PSTL formula parameters can be divided into four steps. First, classification accuracy given in Eq. (8) was used to define the objective function. Second, the Metric Discriminant Model given in Eq. (11) was defined and used to classify the traces. Third, we used the likelihood function given in Eq. (12) to replace the classification accuracy, which can use gradient-based optimization methods. Finally, a novel algorithm was given for specification mining. Interested readers can refer to [29] for specific algorithms and examples.

$$\gamma = \frac{|\{y_i \in C_0 \wedge (x_i, t) \models \phi^*\}| + |\{y_i \in C_1 \wedge (x_i, t) \models \phi^*\}|}{|C_0| + |C_1|} \tag{8}$$

where, x_i is a trace, $y_i \in \{0, 1\}$ is the label, ϕ^* is an STL formula, α is an unknown parameter, and C_0 (C_1) is a set that is unsatisfied (satisfied) with ϕ^* in the pre-label.

The probability of $(x, t) \models \phi^*$ is

$$P(y{=}1|x) = \frac{e^{f(x(t))-\alpha^*}}{1 + e^{f(x(t))-\alpha^*}} \tag{9}$$

The probability of $(x, t) \not\models \phi^*$ is

$$P(y = 0|x) = \frac{1}{1 + e^{f(x(t))-\alpha^*}} \tag{10}$$

$$MDM(x) = \begin{cases} C_0, & \text{if } P(y{=}0|x) \geq P(y{=}1|x) \\ C_1, & \text{if } P(y{=}0|x) < P(y{=}1|x) \end{cases} \tag{11}$$

$$L(\alpha) = \prod_{i=1}^{n} P(y_i = 1|x_i)^{y_i} P(y_i = 0|x_i)^{(1-y_i)} \tag{12}$$

5) Q-LEARNING

Q-learning is a model-free reinforcement learning method, which can be used to find the optimal policy for a finite MDP. In [30] and [31], Austin Jones *et al.* used an MDP to represent an unknown and stochastic dynamic system, in which the transition probabilities are unknown. They solved the following two problems: (Problem 1) maximize the probability of satisfying ϕ and (Problem 2) maximize the average robustness degree of satisfying ϕ, where ϕ is a given STL formula fragment. These two problems are to find policies μ^*_{mp} and μ^*_{mr} given in Eqs. (13) and (14). As problem 1 is not in the standard form to apply Q -learning algorithm, they proposed an approximation of the function such that STL synthesis problem can be solved via Q -learning. For problem 2, they proposed an alternate Q -learning formulation. called *batch Q-learning*, to solve the problem. Interested readers can refer to [30] and [31] for the algorithm pseudocodes.

$$\begin{aligned} \mu^*_{mp} = & \underset{\mu \in \mathcal{F}(\ell \times N, Act)}{\arg\max} \Pr_{s^{0:T}}[s^{0:T} \models \phi] \\ & \text{s.t. } s^{t+1} = f(s^t, \mu(s^{t-\tau+1:t}, T - t), \omega^t) \end{aligned} \tag{13}$$

$$\begin{aligned} \mu^*_{mr} = & \underset{\mu \in \mathcal{F}(\ell \times N, Act)}{\arg\max} E_{s^{0:T}}[r(s^{0:T}, \phi)] \\ & \text{s.t. } s^{t+1} = f(s^t, \mu(s^{t-\tau+1:t}, T - t), \omega^t) \end{aligned} \tag{14}$$

where, $s^{t+1} = f\left(s^t, \mu\left(s^{t-\tau+1:t}, T - t\right), \omega^t\right)$ is an unknown dynamic system, ω^t is a random process, ϕ is an STL formula, $s^{t-\tau+1:t}$ is the trajectory of the system, and μ is a policy.

III. TAXONOMY OF LEARNING ALGORITHMS IN FORMAL METHODS

In formal methods literature, the model learning problems were studied under different names, like system identification [32], grammatical inference [33], regular inference [34], regular extrapolation [35], model learning [36], or active automata learning [37], etc. We do not distinguish these names, sometimes we use them interchangeably, and here we just use the term model learning or learning model instead. Similarly, the learning specification problems also have different names under research, typically are specification mining [38]–[43], specification inference [44], requirements mining [45], mining properties [46], learning logic formulae [47], learning specifications [48], [49], learning properties [50], [51], etc. Here we use the term learning specification (or specification learning) in analogy with learning model (or model learning).

In this section, we investigate the taxonomy of learning algorithms in formal methods, which is an active field of research. On the whole, in formal methods area, there are two main research lines of learning, i.e., learning system's formal specifications and learning (a model or model-free) for verification. Although many learning-based articles have been published that are closely related to formal methods, one can only find few comprehensive review papers of learning formal specifications. For completeness, this paper will give the taxonomy of learning in formal methods, both in learning formal specifications and learning for verification. Fig. 5 shows the considered taxonomy of learning-based formal methods.

A. LEARNING SPECIFICATIONS

As system requirements are not always fully known in advance and indeed cannot be completely manually specified. In order to better understand the system, it is necessary to know the system's specification. For this reason, learning STL requirements from observed traces is an emerging field of research. Here we present the general taxonomy of specification learning algorithms based on their various characteristics.

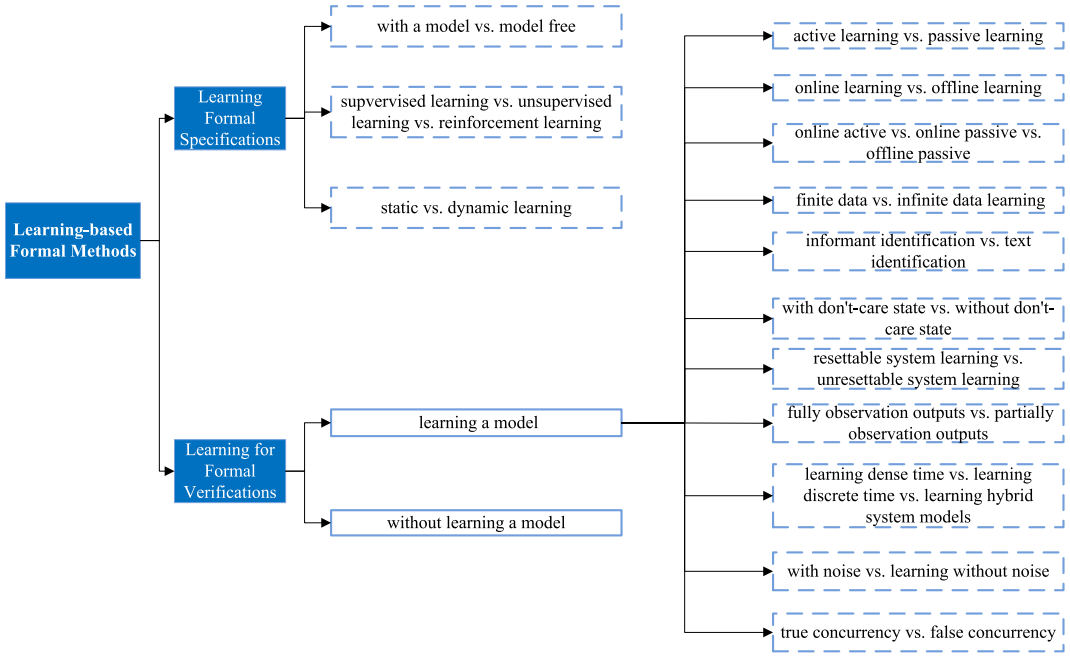

FIGURE 5. Taxonomy.

1) WITH A MODEL VS. MODEL FREE

a: WITHOUT a MODEL (OR MODEL-FREE)

i) GIVEN a PARAMETRIC SPECIFICATION, FIND THE RANGE OR VALUE OF PARAMETERS

It is an active research field on temporal logic inference that focused on the estimation of parameters associated with a given temporal logic structure [52]-[55]. That is, given a parametric specification, then we would like to infer the ranges of parameters for which the property holds/does not hold on to the given sample traces. In [52], Asarin *et al*. solved the problem of finding the range of parameters that render the formula satisfied by a given set of traces under a given PSTL formula, which extended the work of Fages and Rizk [56] that identified parameter ranges for numerical predicates on top of the discrete-time temporal logic LTL [57]. Recently, our team published an article based on logistic regression to learn a CPS specification template [29]. Not only can we learn from the positive examples, but also from the positive and negative examples, which make up for the lack of existing methods that can only learn from positive examples. Table 5 lists some work on the estimation of parameters associated with a given temporal logic structure.

TABLE 5. Some work on parametric estimation.

Logic	Samples	Reference
PSTL	positive	[52]-[54]
PMTL	positive	[55]
QFCTL	positive	[56]
TL	positive	[57]
PSTL	positive and negative	Our work[29]

ii) LEARNING FORMULA STRUCTURE AND ITS PARAMETERS

The structure of the formula reflects the domain knowledge of the designer as well as the properties of interest of a given system or given sample traces. But, if neither domain knowledge is available nor the user is not familiar with the system properties that are to be inferred, a step further is to infer the formula structure in addition to its parameters from data. In [48], [58], [59] the first algorithm was proposed to learn both the formula structure and its parameters from data, this approach was called *temporal logic inference* (TLI). Compared with the parameter estimation problem under a given structure circumstance, structural inference problem is generally hard and even ill posed. In [59], Kong, Jones and Ayala reduced the difficulty of structural learning by imposing a partial order on the set of reactive parametric STL formulae. The defined partial order allows them to search for a formula template in an efficient and orderly fashion, while the robustness degree allows them to formulate the inference problem as a well-defined optimization problem. In [60], the authors proposed a decision tree-based approach for learning the STL formula, while the optimality of the parameters was evaluated by using heuristic impurity measures.

iii) SELECT MODEL + LEARN FORMULA STRUCTURE + LEARN PARAMETERS

The previous two cases are model-free learning circumstances, as they do not use a given model in advance in the learning process. Bartocci *et al*. [51] proposed a novel approach to learn logical formulae characterizing the emergent behavior of a dynamical system from system

observations. Their approach can be divided into three steps. First, select a model (HMM, Continuous-Time Markov Chains, (Stochastic) Differential Equations or Hybrid Systems) based on Akaike Information Criterion (AIC) score. Second, select the good candidate formula structure based on the log odds ratio. Last, tune the parameters of formulae so as to maximize the satisfaction probability. In [61], the authors first inferred a generative statistical model of the observed data, then learnt the structure of TL classifiers based on an Evolutionary Algorithm, and last learnt the best formula parameters based on Bayesian optimization.

b: LEARNING WITH a MODEL

i) COMPUTE THE VALUES OF THE PARAMETERS UNDER a GIVEN MODEL AND FORMULA

The standard model checking paradigm of LTL, can only give a ''Yes/No'' answer, this answer information is limited. A ''model measuring'' paradigm where one can obtain more quantitative information was extended from the standard model checking paradigm in [62]. Given a formula of PLTL and a system model K, they showed one can not only determine whether there exists a valuation under which the system K satisfies the property, but also find valuations that satisfy various optimality criteria. Wang [63] solved the model-checking problem for parameterized branching-time logic. The author extended a TCTL model-checking problem to a parametric timing analysis problem with unknown timing parameters and provided an elementary complexity algorithm for the general solution condition of the problem.

ii) INFER PROPERTY UNDER a GIVEN MODEL AND FORMULA

Model checking was proposed as a verification (or falsification) technique [64], [65], it is valuable for better understanding model's behavior: the user hypothesizes a behavior of the system, which is expressed as a temporal logic formula, and attempts to use the model checker to validate the hypothesis. If the user wants to gain more knowledge about the system, the process will be iterated more times. To further help the user understand system's behaviors, infer temporal properties will be a good choice, which was called Temporal Logic Queries [66], [67]. In [66], a model of the system and a temporal logic formula φ were given, a sub-formula in φ was replaced with a special symbol ?. Then, the problem is to determine a set of Boolean formulas such that if these formulas were placed into the placeholder ?, then φ holds on the model. It can help the designer achieve a better understanding and explore the properties of a model of the system.

2) SUPERVISED LEARNING VS. UNSUPERVISED LEARNING VS. REINFORCEMENT LEARNING

a: SUPERVISED LEARNING

In case of given system outputs labeled according to whether a system behaves normally or not, the task is to learn a formula that can separate the two behaviors. In [59], Kong, Jones and Belta inferred a temporal logic formula that can be used to distinguish between normal system behaviors and anomalous (or undesired) behaviors via supervised learning. In [30], an on-line supervised learning algorithm was proposed, which is more appropriate for systems in which no or very little historical data was available.

b: UNSUPERVISED LEARNING

In another case, if the system outputs are not labeled, i.e., there is no expert-in-the-loop that determines whether a given trace represents a normal or attacked operation. Jones *et al*. [58] inferred a formula to detect out-of-the-ordinary (anomalous) outputs via unsupervised learning. In [68], an unsupervised anomaly detection algorithm was developed by using the Temporal-logic learning-based Anomaly Detection (TempAD) algorithm for detecting aircraft anomalies in the terminal airspace operations.

c: REINFORCEMENT LEARNING

In [31], [69], the authors used a reinforcement learning strategy called Q-learning [70], to provide provably convergent algorithms to maximize the probability of satisfaction or maximize the expected robustness with respect to a given STL formula.

3) STATIC VS. DYNAMIC LEARNING

''Specification mining'' algorithms can also be categorized as static [42], [44], [71] or dynamic [72], [73]. Static learning means to infer specifications from program code, while dynamic learning means to extract specifications from simulation or execution traces [53]. Specifications of program behavior play a central role in many software engineering technologies. Much research has addressed the challenge of learning specifications directly from code [42], so as to solve the problem that software lacking formal specifications. In [42], Weimer and Necula proposed a novel technique for temporal specification mining that uses information about program error handling. As dynamic approaches enjoy the significant virtue that they learn from behavior that definitively occurs in a run. Most such research deals with dynamic analysis and infers specifications from the observed behavior of runs. Based on the observation that common behavior is often correct behavior to refine the specifications mining problem into a problem of probabilistic learning from execution traces, in [72] specifications were extracted by learning probabilistic finite automata that represent temporal, as well as data dependencies from traces of correct system behavior. This is extended in [73] by prior cleansing and clustering of traces. As dynamic approaches can only learn from available representative runs, it has its limitation, i.e., incomplete coverage still remains a fundamental limitation.

B. LEARNING FOR VERIFICATION

FV can be very helpful in proving the correctness of systems, such as digital circuits, communication protocols, etc. There

are mainly two lines for verification by using learning techniques, one is learning a model and then performing verification, another is learning for verification directly without constructing explicit models.

1) LEARNING FOR VERIFICATION BY LEARNING A MODEL

In real life, some systems like black-box or third party systems, the user can't get access to these systems. If the user wants to get more information about the system or want to verify properties that the system satisfies, learning techniques can be used to learn a model that is behavior equivalent to the system. L* [12], ALERGIA [13], and their variants [74]–[80] are the mainly used algorithms for learning a model, especially for learning an automata model. Other learning algorithms, like learning PNs [81], [82] are not the focus of this paper.

Here, we will present the general classification of automata learning algorithms that used for learning models for verification purpose based on their various characteristics. In [20], Maier presented a rough classification, (i.e., the classification between a and f), and we expanded the classification from g to k.

a: ACTIVE LEARNING VS. PASSIVE LEARNING

Active automata learning is concerned with the problem of inferring an automaton model for an unknown formal language L over some alphabet Σ [6]. More specifically, the goal is to infer an unknown target DFA A over a given alphabet Σ. Identification from given finite data [83], identification in the limit [84], and Probably Approximately Correct learning (PAC-learning) [85] also belong to the type of *passive learning*, we will not list them alone in the classification.

Active learning is often formulated as a cooperative game between a learner and a teacher (or an oracle, which is capable of answering these queries appropriately and correctly according to the MAT model). The task of the learner is to learn a model of some unknown formal language L. The teacher can assist the learner by answering membership queries and equivalence queries.

Passive learning is a form of supervised learning, which uses provided sampled data (i.e., given learning examples or logs from some data source) and outputs the exact target automaton or its approximation. The active learning algorithm can influence or choose the data it receives, while the passive learning algorithm does not have any influence on the way the data were sampled.

The difference between passive and active learning is illustrated in Fig. 6 (due to Tong [86]).

FIGURE 6. Difference between active and passive learning.

b: ONLINE LEARNING VS. OFFLINE LEARNING

The main difference between online learning and offline learning is that online learning can request additional data or information for learning during their runtime apart from the given data, while offline learning can only use the given information for learning. This classification corresponds to the classification of learning automata identification frameworks on passive and active learning [19]. An example of the online learning algorithm is the famous Angluin's L* [12] algorithm, while Gold's algorithm [83] works in an offline manner, they both learn DFAs.

c: ONLINE ACTIVE VS. ONLINE PASSIVE VS. OFFLINE PASSIVE

The first two categories can be combined and then reclassified. In the literature [20], the terms *active* and *online* are often mixed up. The reason is that, usually, active learning algorithms work in an online manner and vice versa. To the best of our knowledge, there exists no offline active learning algorithm till now.

Offline passive learning algorithms have to deal with a given set of observations, which comes from a database. Most offline learning algorithms use the state merging approach to identify the structure of the automaton. Several existing algorithms learn an automaton in an offline passive manner. They all proceed through three steps, that is step (1) data extraction, step (2) create a prefix tree, step (3) check for compatibility and state merge, but they have different compatibility checks for state merging and different merging strategies. The best known offline passive learning algorithms are ALERGIA [13], MDI [87] and RTI+ [88].

Online active learning is also often referred to as query learning. As in this paper, the terms active and online are often mixed up, we can view online active learning as active learning or online learning. These two were introduced before, we will not introduce them again. Angluin's L* algorithm [12] is one of the first and most famous active learning algorithms that identify a DFA. In [89], Grinchtein, Jonsson and Leucker introduced an online active learning algorithm for identifying event-recording automata (ERA).

Online passive learning can be used to learn a model when the observations cannot be stored and the number of needed learning samples, as well as the convergence of the learning process, is not known beforehand. In [90], Maier introduced OTALA learning algorithm, which is the first online passive learning algorithm for Timed Automata using only positive learning examples.

d: GIVEN FINITE DATA VS. GIVEN INFINITE DATA LEARNING

As we all know, the more learning samples are available, the more accurate the learned model will be. Input data that used for learning can consist of either a given finite or an infinite number of learning examples. In [83], Gold

showed that the problem of learning the simplest DFA with k states from given finite data are NP-complete. However, when infinite data is present, a number of learning algorithms can converge to a target automaton, which is called learning in the limit [84]. These algorithms are ALERGIA [13] and MDI [87] that learn stochastic DFA, and RTI+ [88] and BUTLA [91] that learn stochastic deterministic timed automata (SDTAs) with one clock for tracking the time evolution.

e: INFORMANT IDENTIFICATION VS. TEXT IDENTIFICATION

Informant identification is a type of supervised learning, where the learning examples consist of both positive examples (i.e., examples that are generated by the target automaton) and negative examples (i.e., examples that are not generated by the target automaton). The learned automata should consistent with the learning examples, namely, it needs to accept the positive examples and reject the negative examples. Learning from informant is probably the best-studied topic of grammatical inference, which resulted in a number of theoretical and empirical results over years.

On the contrary, **text identification** allows learning from only positive examples. It is considered to be one of the purest and most basic problems of grammatical inference, from which many other problems are derived [92]. As faulty events rarely happen, or it is hard to collect sufficient amount of negative learning examples in the industrial context, this is the motivation for learning models from text. The aforementioned algorithms, namely ALERGIA, MDI, RTI+, and BUTLA algorithms learn from text, while RPNI (for learning DFAs) [93] and ID_1DTA (for learning 1-DTAs) [94] algorithms learn from informant.

f: WITH DO NOT-CARE STATE VS. WITHOUT DO NOT-CARE STATE

Algorithms that use incoming and outgoing events and event sequences for the state equivalence check usually have do not-care states, since it may become possible to reach a state using different paths [20]. To avoid do not-care states, the algorithm should check the state information itself (e.g., the signal vector in cyber-physical production systems). In [95] and [96], algorithms used to identify finite automata without do not-care states were proposed. Usually, automata without do not-care states have more states than that with do not-care states. Fig. 7 [20] shows an example to illustrate the difference between automata with and without do not-care states.

g: RESETTABLE SYSTEM LEARNING VS. NON-RESETTABLE SYSTEM LEARNING

Resettable system learning means the system under learning can be reset many times so as to produce as many output samples as possible. The more samples used for learning, the more accurate the result will be. But in real life, there are cases where a black box cannot be reset, or it is uncertain that the system can be reset to the same initial state. Besides, there

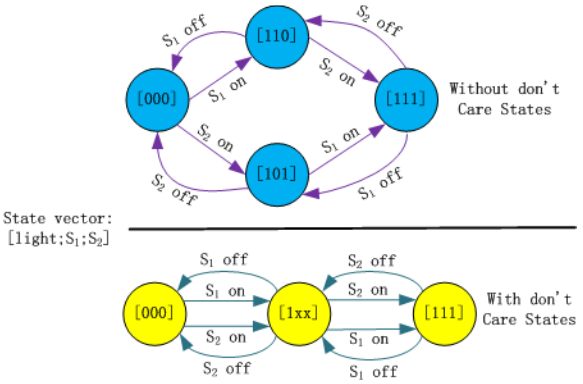

FIGURE 7. An example of automata with and without do not-care states.

are also cases that restart the system completely might be very costly and require a lot of time to reset the whole configuration (e.g., rebooting a machine, with possibly many software components to configure and reinitialize) [97]. Resettable system leaning algorithms can be seen in [98]–[100].

Non-resettable system learning means to learn a black box system without resetting it. The absence of reset was addressed in [101], [102]. Rivest and Schapire [102] pioneered the inference of automata without a reset, but their method was based on Angluin's L* algorithm, which assumes that an oracle can answer equivalence queries. In [97], the first model inference method that does not require resetting the system and does not require an external oracle to decide on equivalence was introduced. Peled *et al.* [103] and Groce*et al.* [104] highlighted the fact that MAT framework can be utilized to infer models of hardware and software components considering them as black boxes as the ''learner'' can reset the ''teacher'' at any point.

h: FULLY OBSERVATION OUTPUTS VS. PARTIALLY OBSERVATION OUTPUTS

Fully observation outputs mean the outputs (e.g., traces) of a system are fully observable. Many algorithms use these outputs for learning a model that is bisimilar to the system in a state-merging manner. State-merging style learning algorithms (e.g., ALERGIA, AALERGIA, IOALERGIA) [13], [76], [77], [99], [100] are fully observations learning algorithms, as no fragment is missing in every observation sequence.

Partially observation outputs mean the outputs of a system are not fully observable due to some reasons, like partial outputs were erosion, and the observations can be viewed as incomplete data. Such problems motivate the research on automated model identification, and various methods have been used to address this problem [105], [106]. In [105], an evolutionary algorithm for tackling the identification problem of Cellular Automata in the context of partial observations problem was presented. In [107], Babaee, *et al.* used a HMM to extend the partially observable paths of the system, and then used reachability analysis to construct a lookup table

TABLE 6. Comparison of commonly used learning algorithms.

Algorithms	Learned original model	Active/Passive	Online/Offline	Informant/Text identification	Positive/Negative samples	Merging strategy	Reference
L^*	DFA	active	online	-	-	-	[12]
ALERGIA	PDFA	passive	offline	text	positive	top-down	[13]
AALERGIA	LMC	passive	offline	text	positive	top-down	[77]
IOALERGIA	MDP	passive	offline	text	positive	top-down	[76],[112]
MDI	PDFA	passive	offline	text	positive	top-down	[87]
RTI+	1-PDTA	passive	offline	text	positive	top-down	[88]
RPNI	DFA	passive	offline	informant	positive & negative	top-down	[93]
ID_1-DTA	1-DTA	passive	offline	informant	positive & negative	top-down	[94]
BUTLA	1-PDTA	passive	offline	text	positive	buttom-up	[91]
HyBUTLA	1-SDHA	passive	offline	text	positive	buttom-up	[114]-[116]
LSGDERA	ERA	active	online	-	-	-	[89]
LDERA	ERA	active	online	-	-	-	[89]

which provides the probability that the extended path satisfies or violates the specification from the current state.

i: LEARNING DENSE TIME VS. LEARNING DISCRETE TIME VS. LEARNING HYBRID SYSTEM MODELS

Generally speaking, we can divide today's technical systems into three types. These are Discrete Event System (DES), continuous system, and hybrid system.

Discrete event systems are those systems that show a state-based behavior, i.e., they comprise a set of discrete states that represent their modes of operation and a set of events that trigger changes between those states [108]. These systems can have only a finite number of changes in a finite time.

Continuous systems are those systems whose belonging variables are value-continuous over time [109]. Real-world examples of such variables are temperature, electricity, voltage, etc. Changes in the continuous system are typically smooth and can occur at any moment in time. These systems can have an infinite number of changes in a finite time.

Systems that exhibit characteristics of both discrete event and continuous systems are called **hybrid systems** [110], [111].

Different learning algorithms were used to learn these three types of system models. Algorithms like IOALERGIA (for learning MDP) [76], [112], LSGDERA and LDERA (for learning ERA) [89] were used to learn discrete event systems, algorithm in [113] was used to learn continuous system, and HyBUTLA (for learning 1-SDHA) [114]–[116] was used to learn hybrid system.

j: LEARNING WITH NOISE VS. LEARNING WITHOUT NOISE

All technical data such as sensor values are subject to noise. So the accuracy of the learned model depends on the sample set whether it is noise-free or not. Learning with noise means partial or all of the learning samples are influenced by noise.

Angluin-style learning algorithms [12], [78]–[80] are noise-free learning algorithms, as these algorithms assume that each finite input sequence is uniquely labeled as either accepted or rejected by the system. In [117], Sloan contrasted the effects of four different types of noise on PAC-learning. As filtering is a procedure for decreasing the measurement noise (called denoising), in [19], advanced filtering and other signal processing technologies were applied before measurements are logged in a database, this process is called data preprocessing.

k: TRUE CONCURRENCY VS. FALSE CONCURRENCY

The term ''true concurrency'' arises in the theoretical study of concurrent and parallel computation [118]. It is in contrast to interleaving concurrency. True concurrency is concurrency that cannot be reduced to interleaving. False concurrency is also known as interleaving concurrency. Concurrency is interleaved if at each step in the computation, only one atomic computing action (e.g. an exchange of messages between sender and receiver) can take place. Concurrency is true if more than one such atomic action takes place in a step. PN and event calculus systems are true concurrency systems, labeled transition systems are false concurrency. In [119], a new learning algorithm that introduces the network learning method into PN update was proposed, and was used to model the web service discovery. In [120], a method based on random hill climbing that automatically builds PN models of non-linear (or multi-factorial) disease-causing gene–gene interactions was also described.

Based on the above commonly used learning algorithms, Table 6 and Table 7 present two comparisons of these algorithms.

TABLE 7. Comparison of the algorithms in usefulness, limitation and applicability aspects.

Algorithms	Model	Usefulness	Limitations	Applicability
L^*	DFA	the learned DFA is easy to analysis	need MAT framework	learn unknown regular set
ALERGIA	PDFA	stochastic behavior is considered, free from the restriction to the learning of acyclic automata	need system running samples, state merging operation is local	identify any stochastic deterministic regular language and also the probability
AALERGIA	LMC	can perform PLTL model-checking on the learned model to infer properties of the system	need the behaviors of the system, data must be noise-free	learn accurate high-level model
IOALERGIA	MDP	it can include both probabilistic and nondeterministic transitions	the system behavior must be in the form of alternating sequences of inputs and outputs	learn reactive systems
MDI	PDFA	it can reduce the complexity compared with ALERGIA, and it can also trade off minimal divergence from the training sample and minimal size	only need positive samples	learn probabilistic automata
RTI+	1-PDTA	it is compatible with the positive and negative data	it can only identify positive data	learn probabilistic timed automata
RPNI	DFA	it can do some sort of generalization in all circumstances compared	based on the assumption that function CHOOSE is a deterministic function	training data include both positive and negative examples
ID_1-DTA	1-DTA	it can induce the specification of a TA from observations	it cannot be identified efficiently from finite data	learn real-time systems
BUTLA	1-PDTA	it can be used for testing and diagnosis of automation systems	based on the assumption of a global time base and a deterministic, discrete system	learn timed automata
HYBUTLA	1-SDHA	it can be used for autonomous jumps detection	based on the assumption that the system's structure is known	learn hybrid automata
LSGDERA	ERA	for analysis of timed reactive systems	based on the assumption that the system can be described by a class of deterministic ERA	learn timed systems
LDERA	ERA	it can learn a (usually) smaller representation of the underlying system than LSGDERA algorithm	based on the assumption that the system can be described by a class of deterministic ERA	learn timed systems

2) LEARNING FOR VERIFICATION WITHOUT LEARNING A MODEL

As some formal specifications can be verified using ML or optimization algorithms, there is no need to learn the model for verification. In [121], reinforcement learning algorithms were adopted to verify MDP, without constructing explicit models.

IV. APPLICATIONS BASED ON LEARNING

Currently, learning-based methods have been applied successfully in formal methods area, especially in learning formal specification and in learning for FV. The applications of model learning were presented in [9], we will not discuss applications in this research direction. In the following subsections, we shall briefly discuss the applications of learning formal specifications in different domains.

A. ANOMALY DETECTION

As the complexity of real life systems (e.g., CPS, air traffic management systems) increases, this leads to systems more vulnerable to attacks, so automated anomaly detection methods that used to detect possible attacks are necessary. Anomaly detection is a new reach direction that has gained more intention in different research areas. Anomaly detection refers to the problem of finding patterns modes in data that do not conform to expected behavior [122]. In different application domains, these unexpected behaviors have different names, like anomalies, outliers, surprises, peculiarities, or exceptions. Among these, researchers often use anomalies and outliers, even interchangeably.

One solution to handle this problem is to infer a specification which can differentiate between expected and unexpected behavior through the observations of the system in question so that trajectories that do not satisfy the inferred

specification are classified as anomalous. A supervised learning algorithm was proposed by Kong *et al*. [59], which was used to infer formulae to distinguish between expected and unexpected behavior. In [58], a data classifier can be inferred from unlabeled data, which is an implementation of formal methods in anomaly detection and other tasks. The approach is able to distinguish the normal and attacked outputs with high discrimination rates. Furthermore, an unsupervised anomaly detection algorithm was proposed in [68] for detecting aircraft anomalies, by using TempAD algorithm.

B. CLASSIFICATION

Methods of ML that used for classification are too specific. In other words, the classification problem is the only goal that we need to solve, without offering other useful information to the user. In contrast, temporal logic formulae [47] not only have precise meaning but also allow for a rich specification of a system behavior, which is interpretable by human experts. Classification problems can be viewed as learning temporal logic formulae that differentiate objects. In order to distinguish between expected and unexpected behaviors, the method of inferring a temporal logic formula from observations was proposed in [59]. Similarly, a method for learning signal classifiers from data was presented in [52], by means of incrementally learning an STL formulae. Other algorithms were introduced to infer GTL formulas from data in [123], which were used for classification and identification.

C. DEBUGGING OR ANOMALY ANALYSIS

Engineers try to understand the behavior of software in all stages of the software development process. For example, from the start to the end of the development, during the requirement discussing, during the implementation planning, during the verification stage, and during the debugging process. Consider developing large, high-level software models, detecting abnormal behavior during the developing process have many benefits, such as reducing debugging time and saving costs. In [124], existing specification mining techniques were applied to test finite state machine in real-world industrial settings and have helped developers in the debugging process. In [125], a diagnosis framework was proposed to automatically localize an error, where specifications were mined from traces. Similarly, another mining approach for detecting and diagnosing software defects was presented in [126], with the FLTL properties mined from the negative traces.

D. MINING REQUIREMENTS

In the literature, system requirements (a.k.a. system specifications) are mathematical expressions expressed in suitable logic formulae, which are the design goals or properties of the system. Unfortunately, formalized system requirements are normally unavailable. For example, consider the case of systems built on legacy models. Moreover, to understand the system model for any engineers except its developers is not an easy task. Therefore, if the system requirements can be derived from observing the system operation results, it can help engineers understand the system and even maintain the system in the future. Based on this idea, a scalable technique, which is used to mine requirements from observations of the system behavior, was proposed in [45].

V. POSSIBLE FUTURE RESEARCH DIRECTIONS

In the previous section, we provided an overview of the important applications of learning specification (or specification mining) presented thus far. Different from others in [127], [128], applications in this paper are based on learning. Namely, we can use the learned model or specification from the unknown system (like block-box) or traces of the system running, and then go one step further to do model checking, testing, and verification, etc.

A. LEARNING FROM COALGEBRA PERSPECTIVE

The first possible research line is to develop more efficient algorithms from coalgebra point of view. Coalgebra [129], [130] studies ''generated behavior'' that can be observed when interacting with a system. Existing work using the observations to learn a model which is (approximate) behavior equivalent to the system in question. However, little work has been down to use the observed behavior to learn how the system works. In [131], Jacobs and Silva revisited Angluin's original algorithm from a categorical perspective. They presented the first step towards a categorical understanding and generalization of Angluin's learning algorithm, originally defined for DFA. In [132], Barlocco and Kupke provided a fresh take on Dana Angluin's algorithm for learning using ideas from coalgebraic modal logic, and proposed the ''L^{co} algorithm'', which is a generalization of Angluin's original algorithm from DFAs to coalgebras. It allows the learning of regular coalgebras for an arbitrary finitary set functor. The connections between coalgebra and learning are still far from being completely understood. Research in this line is few, it has the potential to do further research, like develop more efficient algorithms from coalgebra perspective.

B. LEARNING HYBRID SYSTEM

The second research direction is to learn n-SDHA that clock variables can change uniformly or differently, and also to learn other types of HA. As the world we are living consists of various facilities that can change continuously and discretely, we all called this type of system as hybrid system. Hybrid systems are almost everywhere, they occur frequently in automotive industries, process control, mobile robotics, medical devices, and mixed analog-digital chip design area. In order to analyze hybrid systems, hybrid automata and HPN are the two mostly used models to model them. But manual modeling of a complex real-world hybrid system is a very hard task, it is both time consuming and error-prone. An alternative is the automatic model identification that relies on the theory of learning. Namely, learning the model automatically based on the observation of the system behavior. To the best of our

knowledge, HyBUTLA [19], [114] is the first and only hybrid automata learning algorithm that can deal with all hybrid system characteristics. This algorithm was used to learn 1-SDHA model. A further research direction is to learn n-SDHA that clock variables can change uniformly or differently, and also to learn other types of HA.

C. RESEARCH ON THE RELATIONSHIP BETWEEN THE LEARNED MODEL AND THE UNKNOWN SYSTEM

The third possible work is related to the research on the relationship between the learned model and the system under observation. There are many behavior equivalence relations between two systems, like simulation, strong bisimulation, weak bisimulation, branching bisimulation, trace equivalence, etc. HyBUTLA is the first hybrid automata learning algorithm that can deal with all hybrid system characteristics, but the relationship between the learned HA and the unknown hybrid system has not been discussed. So, this may be a possible research direction.

VI. CONCLUSIONS

Incorporation of ML techniques in formal methods has been an important topic of formal methods research, because in real-world, system models and/or formalized system specifications are normally unavailable due to various reasons. So there exist two research lines of learning, i.e., learning system's formal specifications and learning for verification. This paper provides an up-to-date overview of the current state-of-the-art in learning-based formal methods. This paper only concentrates on learning formal specification and learning for FV, and does not discuss other techniques used in formal methods. Therefore, this paper is not a comprehensive survey of the learning-based techniques in formal methods area, but rather as a survey of the taxonomy, applications, and possible future directions in learning-based formal methods.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their comments that helped to substantially improve the quality of the article.

REFERENCES

[1] D. A. Peled, *Software Reliability Methods*. New York, NY, USA: Springer, 2001.
[2] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Cham, Switzerland: Springer, 2018.
[3] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[4] M. Leucker, "Learning meets verification," in *Proc. FMCO*, Amsterdam, The Netherlands, 2007, pp. 127–151.
[5] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *Proc. SFM*, Bertinoro, Italy, 2011, pp. 256–296.
[6] F. Howar and B. Steffen, "Active automata learning in practice an annotated bibliography of the years 2011 to 2016," in *Proc. Mach. Learn. Dyn. Softw. Anal. Potentials Limits*, 2018, pp. 123–148.
[7] A. Bennaceur and K. Meinke, *Machine Learning for Software Analysis: Models, Methods, and Applications* (Lecture Notes in Computer Science), vol. 11026. Wadern, Germany: Springer, 2018, pp. 3–49.
[8] M. Amrani, L. Lúcio, and A. Bibal, "ML + FV = ♡? A survey on the application of machine learning to formal verification," 2018, *arXiv:1806.03600*. [Online]. Available: http://arxiv.org/abs/1806.03600
[9] S. Ali, H. Sun, and Y. Zhao, "Model learning: A survey on foundation, tools and applications," 2018, *arXiv:1901.01910*. [Online]. Available: http://arxiv.org/abs/1901.01910
[10] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications," in *Lectures on Runtime Verification*. Cham, Switzerland: Springer, 2018, pp. 135–175.
[11] E. Bartocci, "Monitoring, learning and control of cyber-physical systems with STL (tutorial)," in *Proc. RV*, Limassol, Cyprus, 2018, pp. 35–42.
[12] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Nov. 1987.
[13] R. C. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," in *Proc. ICGI*, Alicante, Spain, 1994, pp. 139–152.
[14] L. Bortolussi, D. Milios, and G. Sanguinetti, "Smoothed model checking for uncertain continuous-time Markov chains," *Inf. Comput.*, vol. 247, pp. 235–253, Apr. 2016.
[15] R. M. Keller, "Formal verification of parallel programs," *Commun. ACM*, vol. 19, no. 7, pp. 371–384, Jul. 1976.
[16] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2001.
[17] R. David and H. Alla, "On hybrid Petri nets," *Discrete Event Dyn. Syst. Theory Appl.*, vol. 11, nos. 1–2, pp. 9–40, 2001.
[18] C. Baier and J. P. Katoen, *Principles of Model Checking*. London, U.K.: MIT Press, 2008.
[19] A. Vodenčarević, "Identifying behavior models for hybrid production systems," Ph.D. dissertation, Dept. Elect. Eng., Paderborn Univ., Paderborn, Germany, 2013.
[20] A. Maier, "Identification of timed behavior models for diagnosis in production systems," Ph.D. dissertation, Dept. Elect. Eng., Paderborn Univ., Paderborn, Germany, 2015.
[21] O. Grinchtein, B. Jonsson, and P. Pettersson, "Inference of event-recording automata using timed decision trees," in *Proc. CONCUR*, Bonn, Germany, 2006, pp. 435–449.
[22] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proc. HSCC*, Seattle, WA, USA, 2015, pp. 239–248.
[23] V. Raman, A. Donze, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Proc. CDC*, Los Angeles, CA, USA, Dec. 2014, pp. 81–87.
[24] L. Feng, M. Kwiatkowska, and D. Parker, "Compositional verification of probabilistic systems using learning," in *Proc. QEST*, Williamsburg, VA, USA, Sep. 2010, pp. 133–142.
[25] L. Feng, "On learning assumptions for compositional verification of probabilistic systems," Ph.D. dissertation, Dept. Trinity, Univ. Oxford, Oxford, U.K., Oct. 2013.
[26] F. He, X. Gao, M. Wang, B.-W. Wang, and L. Zhang, "Learning weighted assumptions for compositional verification of Markov decision processes," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–21, Aug. 2016.
[27] R. Bouchekir and M. C. Boukala, "Learning-based symbolic assume-guarantee reasoning for Markov decision process by using interval Markov process," *Innov. Syst. Softw. Eng.*, vol. 14, no. 3, pp. 229–244, Jun. 2018.
[28] Y. Ma, Z. Cao, and Y. Liu, "A probabilistic assume-guarantee reasoning framework based on genetic algorithm," *IEEE Access*, vol. 7, pp. 83839–83851, Jun. 2019.
[29] M. Hu and Z. Cao, "LRx: Specification mining based on logistic regression," in *Proc. ICMSS*, Wuhan, China, Jan. 2020, pp. 68–72.
[30] A. Jones, "Formal methods paradigms for estimation and machine learning in dynamical systems," Ph.D. dissertation, Dept. Eng., Boston Univ., Boston, MA, USA, 2015.
[31] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. CDC*, Las Vegas, NV, USA, Dec. 2016, pp. 6565–6570.
[32] S. Yoshimura, A. Matsubayashi, and M. Inoue, "System identification method inheriting steady-state characteristics of existing model," *Int. J. Control*, vol. 92, no. 11, pp. 2701–2711, Apr. 2018.

[33] H. Hungar, O. Niese, and B. Steffen, "Domain-specific optimization in automata learning," in *Proc. CAV*, Boulder, CO, USA, 2003, pp. 315–327.
[34] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen, "On the correspondence between conformance testing and regular inference," in *Proc. FASE*, Edinburgh, U.K., 2005, pp. 175–189.
[35] A. Hagerer, H. Hungar, O. Niese, and B. Steffen, "Model generation by moderated regular extrapolation," in *Proc. FASE*, Grenoble, France, 2002, pp. 80–95.
[36] F. W. Vaandrager, "Model learning," *Commun. ACM*, vol. 60, no. 2, pp. 86–95, 2017.
[37] M. Isberner, "Foundations of active automata learning: An algorithmic perspective," Ph.D. dissertation, Dept. Comput. Sci., Tech. Univ. Dortmund, Dortmund, Germany, 2015.
[38] M. Gabel and Z. Su, "Symbolic mining of temporal specifications," in *Proc. ICSE*, Leipzig, Germany, 2008, pp. 51–60.
[39] S. Shoham, E. Yahav, J. Stephen Fink, and M. Pistoia, "Static specification mining using automata-based abstractions," *IEEE Trans. Softw. Eng.*, vol. 34, no. 5, pp. 174–184, Sep. 2008.
[40] X. Jianyu and D. Jingguo, "Specifications mining based on adjusted automata learning algorithm," *Inf. Technol. J.*, vol. 11, no. 10, pp. 1391–1399, Oct. 2012.
[41] A. W. Biermann and J. A. Feldman, "On the synthesis of finite-state machines from samples of their behavior," *IEEE Trans. Comput.*, vol. C-21, no. 6, pp. 592–597, Jun. 1972.
[42] W. Weimer and G. Necula, "Mining temporal specifications for error detection," in *Proc. TACAS*, Edinburgh, U.K., 2005, pp. 461–476.
[43] P. R. Bokil, "Generating effective test suites for reactive systems using specification mining," M.S. thesis, Dept. Bus., Bond Univ., Gold Coast, QLD, Australia, 2014.
[44] M. K. Ramanathan, A. Grama, and S. Jagannathan, "Static specification inference using predicate mining," in *Proc. PLDI*, San Diego, CA, USA, 2007, pp. 123–134.
[45] X. Jin, A. Donze, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1704–1717, Nov. 2015.
[46] B. Hoxha, A. Dokhanchi, and G. E. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 20, no. 1, pp. 79–93, Feb. 2018.
[47] G. Bombara and C. Belta, "Online learning of temporal logic formulae for signal classification," in *Proc. ECC*, Limassol, Cyprus, Jun. 2018, pp. 2057–2062.
[48] L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi, "A robust genetic algorithm for learning temporal specifications from data," in *Proc. QEST*, Beijing, China, 2018, pp. 323–338.
[49] A. Mrowca, M. Nocker, S. Steinhorst, and S. Günnemann, "Learning temporal specifications from imperfect traces using Bayesian inference," in *Proc. DAC*, Las Vegas, NV, USA, Jun. 2019, pp. 96–101.
[50] D. Neider and I. Gavran, "Learning linear temporal properties," in *Proc. FMCAD*, Austin, TX, USA, Oct. 2018, pp. 1–10.
[51] E. Bartocci, L. Bortolussi, and S. Sanguinetti, "Data-driven statistical learning of temporal logic properties," in *Proc. FORMATS*, Florence, Italy, 2014, pp. 23–37.
[52] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Proc. RV*, San Francisco, CA, USA, 2012, pp. 147–160.
[53] S. Jha, A. Tiwari, A. S. Seshia, T. Sahai, and N. Shankar, "TeLEx: Passive STL learning using only positive examples," in *Proc. RV*, Seattle, WA, USA, 2017, pp. 208–224.
[54] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "TeLEx: Learning signal temporal logic from positive examples using tightness metric," *Formal Methods Syst. Des.*, vol. 54, no. 3, pp. 364–387, Jan. 2019.
[55] H. Yang, B. Hoxha, and G. Fainekos, "Querying parametric temporal logic properties on embedded systems," in *Proc. ICTSS*, Aalborg, Denmark, 2012, pp. 136–151.
[56] F. Fages and A. Rizk, "From model-checking to temporal logic constraint solving," in *Proc. CP*, Lisbon, Portugal, 2009, pp. 319–334.
[57] A. Pnueli, "The temporal semantics of concurrent programs," *Theor. Comput. Sci.*, vol. 13, no. 1, pp. 45–60, 1981.
[58] A. Jones, Z. Kong, and C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach," in *Proc. CDC*, Los Angeles, CA, USA, Dec. 2014, pp. 848–853.
[59] Z. Kong, A. Jones, A. M. Ayala, E. A. Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proc. HSCC*, Berlin, Germany, 2014, pp. 273–282.
[60] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," in *Proc. HSCC*, Vienna, Austria, 2016, pp. 1–10.
[61] S. Bufo, E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, and L. Bortolussi, "Temporal logic based monitoring of assisted ventilation in intensive care patients," in *Proc. ISoLA*, Corfu, Greece, 2014, pp. 391–403.
[62] R. Alur, K. Etessami, S. La Torre, and D. Peled, "Parametric temporal logic for 'model measuring,'" *ACM Trans. Comput. Log.*, vol. 2, no. 3, pp. 388–407, Jul. 2001.
[63] F. Wang, "Parametric timing analysis for real-time systems," *Inf. Comput.*, vol. 130, no. 2, pp. 131–150, Nov. 1996.
[64] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994.
[65] L. Huimin and Z. Wenhui, "Model checking: Theories, techniques and applications," *Acta Electronica Sinica*, vol. 30, no. 12A, pp. 1907–1912, Aug. 2002.
[66] W. Chan, "Temporal-logic queries," in *Proc. CAV*, Chicago, IL, USA, 2000, pp. 450–463.
[67] M. Chechik and A. Gurfinkel, "TLQSolver: A temporal logic query checker," in *Proc. CAV*, Boulder, Co, USA, 2003, pp. 210–214.
[68] R. Deshmukh and I. Hwang, "Anomaly detection using temporal logic based learning for terminal airspace operations," in *Proc. AIAA SciTech Forum*, San Diego, CA, USA, Jan. 2019, pp. 1–11.
[69] A. Jones, D. Aksaray, Z. Kong, M. Schwager, and C. Belta, "Robust satisfaction of temporal logic specifications via reinforcement learning," in *Proc. CDC*, Osaka, Japan, 2015, pp. 1–8.
[70] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Mach. Learn.*, vol. 16, no. 3, pp. 185–202, Sep. 1994.
[71] R. Alur, P. Černý, P. Madhusudan, and W. Nam, "Synthesis of interface specifications for Java classes," in *Proc. POPL*, Long Beach, CA, USA, 2005, pp. 98–109.
[72] G. Ammons, R. Bodík, and J. R. Larus, "Mining specifications," in *Proc. POPL*, Portland, OR, USA, 2002, pp. 4–16.
[73] D. Lo and S.-C. Khoo, "SMArTIC: Towards building an accurate, robust and scalable specification miner," in *Proc. SIGSOFT FSE*, Portland, OR, USA, 2006, pp. 265–275.
[74] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *Proc. RV*, Toronto, ON, Canada, 2014, pp. 307–322.
[75] M. Shahbaz and R. Groz, "Inferring mealy machines," in *Proc. FM*, Eindhoven, The Netherlands, 2009, pp. 207–222.
[76] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. and Nielsen, "Learning Markov decision processes for model checking," in *Proc. QFM*, Paris, France, 2012, pp. 49–63.
[77] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning probabilistic automata for model checking," in *Proc. QEST*, Aachen, Germany, Sep. 2011, pp. 111–120.
[78] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of NFA," in *Proc. IJCAI*, Pasadena, CA, USA, 2009, pp. 1004–1009.
[79] S. Chaki, E. M. Clarke, N. Sinha, and P. Thati, "Automated assume-guarantee reasoning for simulation conformance," in *Proc. CAV*, Edinburgh, Scotland, 2005, pp. 534–547.
[80] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang, "Learning minimal separating DFA's for compositional verification," in *Proc. TACAS*, York, U.K., 2009, pp. 31–45.
[81] L. Feng, M. Obayashi, T. Kuremoto, and K. Kobayashi, "Construction and application of learning Petri net," in *Petri Nets—Manufacturing and Computer Science*. Rijeka, Croatia: InTech, 2012, pp. 143–176.
[82] J. Esparza, M. Leucker, and M. Schlund, "Learning workflow Petri nets," *Fundamenta Informaticae*, vol. 113, nos. 3–4, pp. 205–228, 2011.
[83] E. M. Gold, "Complexity of automaton identification from given data," *Inf. Control*, vol. 37, no. 3, pp. 302–320, Jun. 1978.
[84] E. M. Gold, "Language identification in the limit," *Inf. Control*, vol. 10, no. 5, pp. 447–474, May 1967.
[85] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984.
[86] S. Tong, "Active learning: Theory and applications," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2001.

[87] F. Thollard, P. Dupont, and C. de la Higuera, "Probabilistic DFA inference using Kullback-Leibler divergence and minimality," in *Proc. ICML*, Stanford, CA, USA, 2000, pp. 975–982.
[88] S. Verwer, "Efficient identification of timed automata: Theory and practice," Ph.D. dissertation, Dept. Comput. Sci., Delft Univ. Technol., Delft, The Netherlands, 2010.
[89] O. Grinchtein, B. Jonsson, and M. Leucker, "Learning of event-recording automata," in *Proc. FORMATS/FTRTFT*, Grenoble, France, 2004, pp. 379–395.
[90] A. Maier, "Online passive learning of timed automata for cyber-physical production systems," in *Proc. INDIN*, Porto Alegre, Brazil, 2014, pp. 60–66.
[91] A. Maier, O. Niggemann, A. Vodencarevic, R. Just, and M. Jäger, "Anomaly detection in production plants using timed automata," in *Proc. ICINCO*, Noordwijkerhout, The Netherlands, Jul. 2011, pp. 363–369.
[92] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. New York, NY, USA: Cambridge Univ. Press, 2010.
[93] J. Oncina and P. Garcia, "Identifying regular languages in polynomial time," in *Advances in Structural and Syntactic Pattern Recognition* (Series in Machine Perception and Artificial Intelligence), vol. 5. Singapore: World Scientific, 1992, pp. 99–108.
[94] S. Verwer, M. de Weerdt, and C. Witteveen, "One-clock deterministic timed automata are efficiently identifiable in the limit," in *Proc. LATA*, Tarragona, Spain, 2009, pp. 740–751.
[95] M. Roth, J.-J. Lesage, and L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems," in *Proc. ACC*, Orlando, FL, USA, Jun. 2010, pp. 2601–2606.
[96] M. Roth, S. Schneider, J.-J. Lesage, and L. Litz, "Fault detection and isolation in manufacturing systems with an identified discrete event model," *Int. J. Syst. Sci.*, vol. 43, no. 10, pp. 1826–1841, Oct. 2012.
[97] R. Groz, A. Simao, A. Petrenko, and C. Oriat, "Inferring finite state machines without reset using state identification sequences," in *Proc. ICTSS*, Sharjah, United Arab Emirates, 2015, pp. 161–177.
[98] R. Babaee, A. Gurfinkel, and S. Fischmeister, "Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning," in *Proc. RV*, Limassol, Cyprus, 2018, pp. 187–204.
[99] J. Wang, J. Sun, S. Qin, and C. Jegourel, "Automatically 'verifying' discrete-time complex systems through learning, abstraction and refinement," *IEEE Trans. Softw. Eng.*, early access, 2018, doi: 10.1109/TSE.2018.2886898.
[100] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning deterministic probabilistic automata from a model checking perspective," *Mach. Learn.*, vol. 105, no. 2, pp. 255–299, May 2016.
[101] F. C. Hennine, "Fault detecting experiments for sequential circuits," in *Proc. SWCT*, Princeton, NJ, USA, Nov. 1964, pp. 95–110.
[102] 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.
[103] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," *J. Automata, Lang. Combinat.*, vol. 7, no. 2, pp. 225–246, 2002.
[104] A. Groce, D. Peled, and M. Yannakakis, "Adaptive model checking," *Log. J. IGPL*, vol. 14, no. 5, pp. 729–744, Oct. 2006.
[105] W. Bolt, J. M. Baetens, and B. De Baets, "An evolutionary approach to the identification of cellular automata based on partial observations," in *Proc. EC*, Sendai, Japan, May 2015, pp. 2966–2972.
[106] W. Bołt, B. Wolnik, M. J. Baetens, and B. De Baets, "On the identification of α-asynchronous cellular automata in the case of partial observations with spatially separated gaps," in *Challenging Problems and Solutions in Intelligent Systems*. Cham, Switzerland: Springer, 2016, pp. 23–36.
[107] R. Babaee, A. Gurfinkel, and S. Fischmeister, "Prevent: A predictive run-time verification framework using statistical learning," in *Proc. SEFM*, Toulouse, France, 2018, pp. 205–220.
[108] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.
[109] K. M. Hangos and I. T. Cameron, *Process Modelling and Model Analysis*. San Diego, CA, USA: Academic, 2001.
[110] M. S. Branicky, "Introduction to hybrid systems," in *Handbook of Networked and Embedded Control Systems*. New York, NY, USA: Birkhäuser, 2005, pp. 91–116.
[111] J. Lygeros, "Lecture notes on hybrid systems," Autom. Control Lab., ETH Zürich, Zürich, Switzerland, Tech. Rep. AUT06-08, Dec. 2006.
[112] Y. Chen and T. D. Nielsen, "Active learning of Markov decision processes for system verification," in *Proc. ICMLA*, Boca Raton, FL, USA, Dec. 2012, pp. 289–294.
[113] K. Sen, M. Viswanathan, and G. Agha, "Learning continuous time Markov chains from sample executions," in *Proc. QEST*, Enschede, The Netherlands, 2004, pp. 146–155.
[114] A. Vodencarevic, H. Kleine Buning, O. Niggemann, and A. Maier, "Identifying behavior models for process plants," in *Proc. ETFA*, Toulouse, France, Sep. 2011, pp. 1–8.
[115] A. Vodencarevic, "Learning behavior models of hybrid systems using wavelets for autonomous jumps detection," in *Proc. INDIN*, Beijing, China, Jul. 2012, pp. 151–156.
[116] A. Vodencarevic, H. K. Buning, O. Niggemann, and A. Maier, "Using behavior models for anomaly detection in hybrid systems," in *Proc. ICAT*, Sarajevo, Bosnia Herzegovina, Oct. 2011, pp. 1–8.
[117] R. H. Sloan, "Four types of noise in data for PAC learning," *Inf. Process. Lett.*, vol. 54, no. 3, pp. 157–162, May 1995.
[118] H. Bowman and R. Gomez, *Concurrency Theory*. Berlin, Germany: Springer, 2006, pp. 1–139.
[119] L. Feng, M. Obayashi, T. Kuremoto, and K. Kobayashi, "A learning fuzzy Petri net model," *IEEJ Trans. Electr. Electron. Eng.*, vol. 7, no. 3, pp. 274–282, May 2012.
[120] M. Mayo, "Learning Petri net models of non-linear gene interactions," *Biosystems*, vol. 82, no. 1, pp. 74–82, Oct. 2005.
[121] T. Brazdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Kwiatkowska, D. Parker, and M. Ujma, "Verification of Markov decision processes using learning algorithms," in *Proc. ATVA*, Sydney, NSW, Australia, 2014, pp. 98–114.
[122] M. Zandrahimi, H. R. Zarandi, and M. H. Mottaghi, "Two effective methods to detect anomalies in embedded systems," *Microelectron. J.*, vol. 43, no. 1, pp. 77–87, Jan. 2012.
[123] Z. Xu, A. J. Nettekoven, A. A. Julius, and U. Topcu, "Graph temporal logic inference for classification and identification," 2019, *arXiv:1903.09714*. [Online]. Available: http://arxiv.org/abs/1903.09714
[124] M. Jafar Mashhadi and H. Hemmati, "An empirical study on practicality of specification mining algorithms on a real-world application," 2019, *arXiv:1903.11242*. [Online]. Available: http://arxiv.org/abs/1903.11242
[125] W. Li, A. Forin, and S. A. Seshia, "Scalable specification mining for verification and diagnosis," in *Proc. DAC*, Anaheim, CA, USA, 2010, pp. 755–760.
[126] F. Wang, J.-H. Wu, C.-H. Huang, C.-C. Chang, and C.-C. Li, "Temporal specification mining for anomaly analysis," in *Proc. APLAS*, Melbourne, VIC, Australia, 2013, pp. 273–289.
[127] S. P. Nanda and E. S. Grant, "A survey of formal specification application to safety critical systems," in *Proc. IEEE 2nd Int. Conf. Inf. Comput. Technol. (ICICT)*, Kahului, HI, USA, Mar. 2019, pp. 296–302.
[128] H. A. Gabbar, *Modern Formal Methods and Applications*. Dordrecht, The Netherlands: Springer, 2006.
[129] B. Jacobs, *Introduction to Coalgebra: Towards Mathematics of States and Observation*. New York, NY, USA: Cambridge Univ. Press, 2016.
[130] B. Jacobs and J. Rutten, "An introduction to (co)algebras and (co)induction," in *Advanced Topics in Bisimulation and Coinduction* (Cambridge Tracts in Theoretical Computer Science), vol. 5. 2011, pp. 38–99.
[131] B. Jacobs and A. Silva, "Automata learning: A categorical perspective," in *Horizons of the Mind. A Tribute to Prakash Panangaden*. New York, NY, USA: Springer, 2014, pp. 384–406.
[132] S. Barlocco and C. Kupke, "Angluin learning via logic," in *Proc. LFCS*, Deerfield Beach, FL, USA, 2018, pp. 72–90.

FUJUN WANG was born in Yangzhou, Jiangsu, China, in 1987. He received the M.S. degree from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2013. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing. His research interests include formal methods and logic in computer science.

ZINING CAO received the M.S. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1998, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2001. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. He has published more than 30 articles. His current research interests include formal methods and logic in computer science.

LIXING TAN received the M.S. degree from Hangzhou Dianzi University, Hangzhou, China, in 2011. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include artificial intelligence and logic in computer science.

HUI ZONG received the M.S. degree from the Nanjing University of Science and Technology, in 2011. She is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. She is currently a Senior Laboratory Engineer with the Huaiyin Institute of Technology, China. Her current research interests include data mining, parallel computing, and formal methods.