2020 11th International Conference on Information and Knowledge Technology (IKT)
December 22-23, 2020; Shahid Beheshti University - Tehran, Iran

Challenges of Specification Mining-based Test Oracle for Cyber-Physical Systems

Maryam Raiyat Aliabadi
Computer Science and Engineering
Shahid Beheshti University, C.G.
Tehran,Iran
m_raiyataliabadi@sbu.ac.ir

Hassan Haghighi
Computer Science and Engineering
Shahid Beheshti University, C.G.
Tehran,Iran
H_haghighi@sbu.ac.ir

Mojtaba Vahidi Asl
Computer Science and Engineering
Shahid Beheshti University, C.G.
Tehran,Iran
mo_vahidi@sbu.ac.ir

Ramak Ghavamizadeh Meybodi
Computer Science and Engineering
Shahid Beheshti University, C.G.
Tehran,Iran
r-ghavami@sbu.ac.ir

2020 11th International Conference on Information and Knowledge Technology (IKT) | 978-1-6654-0441-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/IKT51791.2020.9345615

***Abstract*—Test oracle problem is considered as a major challenge in software testing. Specification mining techniques are shown to be effective to tackle test oracle problem in software systems. In addition, modern systems such as Cyber-Physical Systems (CPSes) have special constraints that should be satisfied when deriving the test oracles for these systems. However, comparing different specification mining techniques for CPS applications is challenging, because no common ground to assess the effectiveness of such techniques has been established yet. In this survey, our contribution is two folded: First, we analyze the CPS constraints from the test oracle point of view, and present a framework of requirements representing six essential criteria for evaluating to which extent specification miners satisfy CPS constraints. Secondly, we review the literature for the specification mining techniques, and use our framework to compare the effectiveness of various static and dynamic analysis-based specification mining techniques, and to discuss their respective advantages and disadvantages.**

***Index Terms*—Cyber-Physical Systems, Test Oracle, Specification mining, Safety-critical, Internet of Things.**

I. Introduction

Software testing involves examining the behaviour of a program in order to detect potential abnormalities. A test oracle is defined as a procedure that distinguishes between the proper and incorrect behavior of a program under test given a particular input [1]. Test oracle problem has been left as a key remaining bottleneck in software testing process as it needs intelligence to deploy the software specifications to conclude what is the expected result or whether the current behavior is correct. Here, there are two major challenges: First, the time to market constraint leaves many software systems without specifications. Secondly, designating a domain expert (human oracle) to infer the expected result given different test inputs is quite costly and error-prone. To address these challenges, specification-mining techniques came into play to automatically derive the test oracles. A derived test oracle distinguishes a system's correct from incorrect behavior based on information derived from various artifacts (e.g., source code, system executions).

With software evolution and the advent of new software systems such as *Cyber-Physical Systems (CPSes)*, the test oracle problem becomes even more challenging. On one hand, the rapid growth of Internet of Things has led to deployment of CPSes without adequate detection solutions against software bugs. On the other hand, these systems carry out critical tasks, in which every undetected bug has potential for catastrophic consequences. Moreover, as we will discuss later in this paper, CPSes have special constraints and characteristics that make them differentiated from the other software systems from the test oracle perspective. We therefore require a concerted effort to find methods to address the test oracle problem in CPSes, and to integrate automated or partially automated test oracle solutions into CPS testing procedure. This paper seeks to help address this challenge.

To the best of our knowledge, we are the first that provide a comprehensive review and analysis of the existing literature of the specification-mining techniques for the test oracle problem, and formulate the requirements that should be tailored to specification miners to satisfy the special constraints of CPS systems from the oracle perspective.

II. Cyber-physical systems

CPSes have been investigated as a key area of research since they are the core of internet of things. Recently, they are widely deployed in many critical infrastructures such as smart medical devices, surgical robots, smart grid, smart cars, and Unmanned Aerial Vehicles (UAVs). CPSs integrate computing and communication with control and monitoring of entities in the physical world. A CPS consists of a cyber unit (i.e., control program), and a physical unit connected by a communication channel. It also includes a control loop which involves interactions between the cyber and physical domains. As the interaction between the physical and the cyber domain increases, the physical unit becomes more susceptible to the software bugs that might exist in the cyber unit. Therefore, the

978-1-6654-0441-9/20/$31.00 ©2020 IEEE

TABLE I
FRAMEWORK OF REQUIREMENTS FOR A CPS' TEST ORACLE.

CPS Constraint	Test Oracle Requirement
C1. Continuous operations	R1. Dynamic test oracle
C2. Real time constraints	R2. Time-driven model for oracle
C3. Resource constraints	R3. Scalable oracle with reasonable resource overheads
C4. Safety-critical constraints	R4. Oracle soundness
C5. Large scale deployment	R5. Oracle completeness
C6.Unclear expected result	R6. White test oracle inferring behavioral pattern
C7. Early fault detection essence	R6. White test oracle inferring behavioral pattern

safety of the overall system strongly depends on the reliability of the CPS' control program.

A. CPS Challenges

In the following, we discuss the key constraints of a CPS that test oracles must satisfy to be applicable to the CPS' control program. Accordingly, we extracted a framework of requirements for the CPS test oracles, and presented in Table I. Our framework includes six essential criteria to evaluate the effectiveness of the specification mining-based test oracles for the CPS systems.

C1. Continuous operation CPSes are decision making agents that need to make decision in real-time and to address the continuous operation capabilities. In these systems, real-time availability is a necessity, which leads to a stricter operational environment [2]. Therefor, as the continuity of operation is of the utmost significance for CPSes, it is essential to generate dynamic test oracles for these systems. By *dynamic test oracle*, we mean those oracles that dynamically monitor the system for providing early response against faults or failures.

C2. Real-time constraints CPSes typically interact with their environment in a real-time fashion. From a test oracle point of view, taking real-time requirements into account is vital for two reasons: First, in a real-time system, the operational correctness depends on both logical correctness, and correct timing behavior [2]. This implies that the logical assertions are not enough for verifying the correctness of these systems, and thus incorporating time (as a semantic property) in their system model is essential to enable the test oracle to monitor the incorrect operations. Secondly, the performance overhead imposed by test oracle has to be small enough to satisfy the CPS real-time constraints.

C3. Resource constraints As a single thing in the Internet of Things, a CPS performs a single task on a single platform with limited CPU, memory and computational power. It implies that these systems need a lightweight test oracle that satisfies their resource constraints. For instance, an important component of a test oracle is the model that represents the correct behavior of the CPS. This model may occupy a large space in memory, which makes the test oracle inapplicable due to memory overheads.

C4. Safety-critical constraints As many CPSes have the potential for very high consequences of failure, they are considered safety-critical. For example, any undetected fault/failure in a pacemaker may be fatal for the pacemaker-implanted patient. Therefore, it is critical for CPSes to deploy *sound test oracles*, in which all faulty program states are rejected by the oracle, so there are no missed faults/failures (no false negatives) [3].

C5. Large-scale deployment CPSes are often deployed on a large scale (e.g., smart meters), in which false positives may aggregate within the whole grid and consume substantial resources. Moreover, these systems are deployed in mission-critical applications where shutting them down on a false alarm is not acceptable. Thus, they need *complete test oracles*. Completeness in the context of test oracle means all proper program states are accepted by the test oracle, and no false alarms are raised [3].

C6. Unclear expected results Unlike traditional software systems, in which there is certain expected results given the particular inputs, in most CPSes, there is no clear expected result given sensing input data, i.e., there is no direct relationship between input and output values as the control commands (outputs) might dynamically change based on external attributes. For instance, in smart grid, the output of one smart meter in conjunction with the other meters' outputs are used to modify the circuit breakers responsible for connecting/disconnecting transmission lines and hence, change the topology of transmission networks for load-balancing purposes. As a result, in these systems, the black-box approaches may not be suitable option for generating CPS test oracle; instead, CPSes need *white test oracles* that perform more in-depth analysis on system behavior using white box testing approaches to differentiate between the correct and incorrect states of program.

C7. Early fault detection Non-propagated faults are believed to be difficult to detect. These faults may corrupt the program internal states, but are masked or failed to propagate to the outputs to be observed. However, a small change in the execution environment may allow the corrupted state to propagate, causing unexpected system failure [4]. Therefore, test oracles should have the ability for early detection of software faults before they are propagated as error or failures. As discussed earlier, black box specification mining methods only deal with input and output of the system, thus they are not able to detect non-propagated faults. The white test oracles, however, are more likely to address this constraint as they observe internal states behavior to discover the relational patterns among different states of the program for fault detection.

III. Specification mining-based test oracles

Specification mining techniques have been shown to be effective to automatically derive test oracles for software systems [5], [2]. These techniques first build a behavioral model for system by defining a set of rules known as *invariants*. Then the test oracle enforces the inferred model over test procedure to define whether a given pattern of activity is correct or faulty. An invariant, or interchangeably a mined specification, is a logical condition that holds true at a particular set of program points. Invariants can be either discovered by *static analysis* or *dynamic analysis* of the program.

In this paper, we first survey the literature for the existing techniques in each category (as illustrated in Fiqure 1), then we discuss why they may/may not be sufficient for deriving a test oracle for a CPS's control program.

IV. Static analysis-based techniques

Static program analysis automatically compute information about the behavior of a program based on the source code without executing it. These techniques are inherently conservative with low false positives, as they only generate invariants that are rigorously provable. Among static analysis-based specification mining techniques, Finite state machines, Symbolic executions and Abstract interpretation account the most highlighted ones.

Finite State Machine (FSM): FSM is a widely used technique for specifying the behavior of software system using static program analysis.Wagner and Dean also extracts automaton model from source code for their FSM-based intrusion detection system [6]. They proposed an approach to build a model of the software system based on the system call-graph. They show that the call-graph leads to a high false negative rate, and they improve their approach by introducing non-deterministic pushdown automaton (NDPDA), which builds an extensive model of the software system based on the system calls. NDPA results in an improved accuracy. However, it is slow because of a high memory overhead. Giffen et. al. [7] proposed the Dyck model based on static analysis of the software and its system calls. While this improves the accuracy, it results in increasing the size of the model noticeably.

Abstract Interpretation: is a static program analysis-based technique that compute an approximation of a program correct behavior, that can be fed to model checkers for program verification purposes.In some cases, the abstract model is just a finite automaton, whose transitions are triggered by certain constructions in the source code [8]; this is a course-grained approach that allows checking at the source code's high-level properties.However, it leads to many false alarms. Predicate abstraction is currently the predominant abstraction technique in software model checking [9].The main challenge in predicate abstraction is identifying predicates, as predicates determine the accuracy of the abstraction. Clarke et.al. [10] proposed Counterexample-guided abstraction refinement(CEGAR) to improve the accuracy of abstraction. In CEGAR, if model checking of the abstraction results in a counterexample, which does not exist in the original program, the abstract counterexample is used to identify new predicates, which leads to obtain a more accurate abstraction.

Symbolic Execution: was primarily developed for software testing, but it has subsequently used for bug finding, program verification and specification mining. For example, [11] introduced a symbolic algorithm, based on binary decision diagrams (BDDs), that exploits the existing regularity in the representation of possible combinations of system components to find meaningful specifications. This specification are shown to be useful for discovering unknown bugs in large, real-world systems. Jaffar et. al. [12] presented extended symbolic execution with interpolation to address unbounded loops in the context of program verification. It proposed an algorithm that discovers invariant interpolants proving the unreachability of error nodes. Symbolic execution is an effective technique in detecting syntactic and semantic errors. However, the exponential number of symbolic paths is the main challenge of this technique.

V. Dynamic analysis-based Techniques

Dynamic analysis-based techniques exploit various system execution traces to derive test oracles. They log the key points of the program to peek into the actual program behavior at run-time, and infer a set of likely invariants. These techniques often take the benefits of **data mining**, or **machine learning** techniques, or combination of both.

A. Data Mining

In recent years, there has been a significant amount of work on using data mining to model the behavior of software systems for program understanding, formal verification, debugging, bug/intrusion detection and test oracles [13], [14], [15]. Generally, in these techniques, the program execution traces is first transformed into itemsets [13], graphs [15], or other forms. In the next step, data mining algorithms are applied on the transformed forms to discover common patterns (e.g., frequent itemsets or sub-graphs) and infer programming rules. The final step is to detect violations against the inferred rules. These techniques can be categorized into two groups including *Positive Association Rules(PAR)* and *Negative Association Rules (NAR)* mining techniques. PAR-based mining techniques extract the positive association rules in the form of $A \Rightarrow B$, meaning that, when an event A appears, the event B should also appear. According to Fiqure 1, PAR-based mining techniques can be categorized into three classes, based on the type and scope of the invariants that they generate: (a) data invariant detection, (b) *temporal specifications mining*, and (c) *multi-dimensional invariant inference* techniques. In the following, we bring more details about each class.

Data Invariant Inference: represent the relational constraints on data values. For instance, Ernst et. al. [16] proposed Daikon, the first dynamic analysis-based technique to automatically derive data invariants. Daikon involves program variables from dynamic traces, and learns (likely) invariants about data value relations. [17] proposed DIDUCE, a dynamic

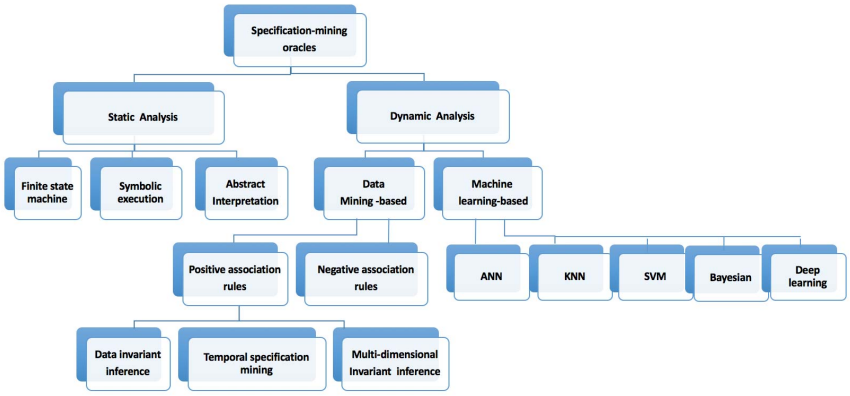

Fig. 1. Specification mining test oracles.

invariant detection technique that combines invariant detection and checking in a single tool for fault diagnosis purposes. The advantage of this technique is that it not only able to dynamically monitors and checks the software, but also scales dynamic invariant detection to large programs. Csallner et.al. [18] combine the advantages of dynamic invariant inference using Daikon and static analysis using symbolic execution in a tool called DySy , that results in inferring more accurate invariant set than that of Daikon.

Temporal specification mining: account a big class of dynamic analysis-based techniques. These techniques derive a set of rules that represent either a) events relationships, b) data and event relationships, or c) time dependencies of events in a program, which are mostly represented in the form of *temporal invariants*, *finite state automata* or *UML sequence chart*. For example, Perracotta [15] captures the sequence of events within a program execution paths by inferring finite state machines by tracking dynamic traces. It mine traces for two-event rules of the form $G(a \Rightarrow XF(b))$, in which G, X, and F are LTL operators. GK-tail algorithm generates integration models that capture the relationship between data and events. It merges temporal specifications and data invariants into Extended Finite State Machine models [19]. Lo et. al. [20] also used data mining methods to extract important scenario-based specifications in the form of live sequence charts (LSC), which leads to more sound and complete specifications. Ohmann et. al. proposed Perfume, that is a specification mining tool designed for modelling system properties based on resource (time and storage) consumption [21].

Multi-dimensional invariant inference: R.Aliabadi et.al. [2] proposed a dynamic invariant detection technique that mines invariants along the three dimensions of data, event, and time, and generate a multi-dimensional model of system, that can be used for finding both software bugs and security attacks. Their ARTINALI tool incorporates time as a first-class notion in the mined invariants, in addition to the traditional data and event invariants, inferred by previous categories, and discovers the interplay among three dimensions. This is important for CPSes as most of them have predictable timing behaviors to a first order of approximation, and hence leveraging this predictability leads to higher accuracy (i.e., lower false-positives and negatives). Using two classes of CPSes as case studies, ARTINALI indicated over 98% true positives on the faults injected into the software. However, FP rates in ARTINALI is still high for both studied CPS platforms.

Negative association rules (NAR): While invariant detection techniques and temporal specification miners rely on positive association rules in the form of $A \Rightarrow B$, there are a couple of work in the literature that focused on inferring the negative association rules in the form of $A \Rightarrow \neg B$.The main problem of this approach is the search space explosion, that leads to a very large number of uninteresting rules. [22] proposed multiple types of negative association rules of forms $A \Rightarrow \neg B$, $\neg A \Rightarrow B$ and $\neg A \Rightarrow \neg B$. Bian et.al. [23] introduced NAR-Miner, a tool that automatically extract negative association programming rules from large-scale systems, and detect their violations to find bugs. NAR-Miner explores various semantic relationships, e.g., data dependence, which leads to introduce more meaningful rules.

B. Machine Learning

During recent years, machine learning has got a lot of attention to resolve the test oracle problem. The accuracy measure of the ML-based oracle is the percentage of inputs that are correctly classified by the model. It is shown that ML-based techniques such as Artificial Neural Networks, Support Vector Machines, K-Nearest Neighbour and Bayesian Networks show high accuracy, and hence can be used as test oracle for software testing. In the following, we provide more details about ML-based oracles.

Artificial Neural Networks(ANN): is one of the most widely applied pattern recognition algorithm in machine learning that have been recently used in oracle testing. Vanmali et. al. [24] proposed an automated ANN-based oracle to simulate the software behavior using the previous version of

the software, and applied this model for regression testing. Aggarwal et. al. [25] applied a trained ANN to test the triangle classification problem. Shahmiri et. al. [26] applied ANN-based Oracles to test decision-making structures and verify complex logical modules. While all of the above ANN-based approaches use a single ANN to generate expected results, a multi-networks ANN based oracle was introduced by [5], in which each ANN is responsible to produce one of the expected outputs. This capability enables multi-network ANN-based oracles to identify faults more accurately; particularly; in complex software systems requiring large scale scenarios. The drawbacks, however is that training a multi-networks oracle impose considerably more time and space overheads than single network one.

K-Nearest Neighborhood (KNN): has been mostly used in the context of Ensemble of Classifiers (EoC) to build a test oracle since EoC has been shown to be useful for improving classification accuracy. If one classifier from an EoC properly classifies a given entry, then EoC is said to be able to classify this entry in oracle; therefore, the more diverse the EoC, the better the oracle. Ko et. al. [27] proposed the KNN-based Oracles family of techniques with different characteristics. Vriesmann et.al. [28] proposed a KNN-based dynamic ensemble selection (DES) method, in which overall local accuracy (OLA) and local class accuracy (LCA) are combined into a two step selection method. The OLA and LCA are calculated on the neighborhood of the test entry in a validation set to filter out the classifiers chosen by the KNN-based oracles.

While the above KNN-based techniques use only one criterion about the behavior of a base classifier to estimate its level of competence, [29] presents a KNN-based dynamic ensemble selection framework using five distinct sets of meta-features, each one corresponding to a different criterion to measure the level of competence of a classifier for the classification of entries.

Support Vector Machine (SVM): has been extensively used to solve many classification problems generally, and to tackle oracle problem particularly [30], [31]. Elish et. al. [30] proposed an SVM-based approach to predict bugs in defect-prone software modules and compared its prediction accuracy against eight statistical and machine learning models in the context of four NASA datasets. Xing et. al.A SVM-based modeling approach for software reliability prediction is proposed in [32]. This work is focused on optimizing an upper bound of the generalization error that eventually results in better generalization performance. Wang et. al. [33] combined genetic programming (GP) and SVM to build the test oracle, in which GP discovers and encodes the test cases into bit patterns, and SVM classifier is trained by bit patterns and is used to verify correct behavior. The advantage of this technique is that few test cases are required to construct a test oracle. This results in a light-weigh oracle with high prediction accuracy.

Bayesian Network (BN): There is also a literature on learning probabilistic specifications using Bayesian networks from programs. This category considers a specification as a probabilistic rather than a boolean model. For example, Cuifang Zheng et. al. [34] built a Bayesian Network, computing the probability of defects using inference of causes and effects. Their technique was effective and somewhat accurate in detecting defects. [35] presented a Bayesian framework that can learn probabilistic specifications from large, heterogeneous code corpora and then use these specifications to find likely API usage errors in Android programs.

VI. Discussion

Table II compares the described specification mining techniques based on six important criteria (framework of requirements), that we extracted in Section II for the CPS' oracle. In this following, we briefly describe the capabilities and limitations of static and dynamic mining techniques respectively. We use our criteria to evaluate each category, and discuss whether (or not) they are adequate to be deployed for the CPS test testing.

A. Static mining capabilities and limitations

Static specification mining techniques generate a model that includes all *potential* behaviors of system. The main advantage of static mining is low false positives that is essentially required in CPS systems according to R5. On other hand, static mining techniques analyze the program without executing the program, and hence in general, they can not provide adequate information about the real-time behavior of CPS system in its operational environment, which in turn, leads to high false negatives. Moreover, as these techniques need to access the source code, they can not statically verify many interesting properties of a program because of unavailable code (e.g., calls to compiled external libraries). Furthermore, these techniques incur high overheads, often exceeding the resource constraints of the CPS. This is specially the case for FSM-based approaches that generate a big model for complex CPS systems. This drawback leads to the lack of scalability of their model which affects their usefulness. Moreover, for the lack of observeability and controlability in CPSes, it is very difficult to find the relationship between input data and expected result using static mining. For instance, we are not able to calculate symbolic values for outputs based on symbolic input data using symbolic execution. In addition, the lack of scalability is a major issue in symbolic execution, which leads to unacceptable time overhead and inability to provide a dynamic oracle in CPS systems. Furthermore, abstract models are limited to a subset of program, and compute an approximation of a program correct behavior, and hence they suffer from high false negatives. These drawbacks make the static analysis-based techniques alone insufficient for building oracles for CPSes with respect to requirements R1,R2,R3 and R4 as indicated in Table II.

B. Dynamic mining capabilities and limitations

Dynamic specification mining techniques follow the assumption that common behavior is correct behavior, and hence

TABLE II
COMPARING THE SPECIFICATION MINING TECHNIQUES FROM CPS ORACLE REQUIREMENTS PERSPECTIVE.

	R1. Dynamic oracle	R2. Time-driven model	R3. Overheads	R4. soundness	R5. Completeness	R6. White oracle
Finite State Machine	✗	✗	✗	✗		
Abstract Interpretation	✗	✗	✗	✗		
Symbolic Execution	✗	✗	✗	✗		
Artificial Neural Network	✗	✗	✗			✗
K Nearest Neighbour	✗	✗	✗			✗
Support Vector Machine		✗				✗
Bayesian Network		✗				✗
Data invariant inference		✗	✗	✗	✗	
Temporal specification miners	✗	✗	✗	✗	✗	
Negative Association Rules	✗	✗	✗	✗	✗	
Multi-dimensional invariant inference					✗	

their mined specifications reflect the *common* behavior rather than *potential* behavior like what is identified in static mining.

The main advantage of these techniques is that they provide information about the real-time behavior of the program, and hence they are *potentially* able to provide a sound and time-driven model for system, which satisfy requirements R2 and R4 in Table II. However, these techniques have also drawbacks as follows:

Data mining-based techniques: mine execution traces of the system for deriving invariants with different type and scope. For instance, data invariant detection techniques produce a model for data relation constraints without taking into account the events or timing of the software system, which are necessary in order to build a rich model of software. Temporal specification mining techniques derive a set of rules that represent either events relationships (class a), data and event relationships (class b), or time dependencies of events (class c) in a program. Class a of temporal specification miners only captures the sequence of events within a program execution paths independent of data or timing information of system. Class b generates integration models that capture the relationship between data and events. E.g., they classify data invariants that arise among method calls. This class generates more complete set of invariants than class a. However, it lacks the timing information of program. Class c is designed for modelling system properties based on resource (time and storage) consumption. It generates an integration model of event relations and their time constraints. Although this class considers time as a part of model, it does not consider the relationship between data and time. As a result, none of temporal specification miners provide a rich model of system including all necessary dimensions (data, event and time) together, and hence their model are neither sound nor complete. The same limitations are relevant to the model inferred by NAR-based techniques, that only capture the infrequent event relationships in the program. These drawbacks make the above data mining techniques challenging to deploy for CPSes that are used in safety-critical (R4) infrastructures and/or on a large scale (R5). Moreover, they mostly generate a huge number of invariants, which leads to not only consuming a lot of memory, but also increasing the time overhead of checking invariants against run-time behavior, both of which are not acceptable for CPSes regarding their real-time and resource constraints (R2 and R3). Multi-dimensional invariant inference (ARTINALI) technique outperforms the other data mining-based techniques from oracle requirements perspectives as shown in Table II. ARTINALI is the only tool in the literature that generates a time-driven model of system, which is one of the critical requirements of CPS oracle (R2). It also suggests high detection accuracy with reasonable time and space overheads. However, it still suffers from false positives, and hence needs improvement. To reduce the false positives, one can deploy multiple variants of the code and switch to a different variant when a bug or an attack is detected. If the invariant is not violated in the second version, it may be a false positive. The second solution is to conservatively remove invariants that exhibit high false positives, but this may also increase the false negatives. The third option is to combine this technique with a powerful static analysis-based technique to iteratively filter the unsound and incomplete invariants.

1) ML-based techniques:: provide more accurate oracle model than the majority of data mining-based techniques. For example, multi-networks ANN indicated a competitive detection accuracy with multi-dimensional invariant inference technique (over 98%). However, the common paradigm among ML-based approaches is that they use black-box approach without considering the internal relations of program states, since only inputs and outputs of the system are presented to the algorithm. That means, these techniques are not able to satisfy CPS white oracle requirement (R6), that is necessary for the sake of controlability and early fault detection in CPSes. In addition, many of these techniques incur a high run-time computational cost. For example, multi-networks ANN needs n separate ANN implementations for n outputs. Or, KNN-based EoC deploys a huge number of classifiers to build an accurate test oracle. Moreover, their huge overheads not only leads to lack of scalability, but also is an obstacle for having dynamic oracle (R1). In addition, no ML-based oracle considers real-time constraints in the model. These limitations make ML-based techniques challenging to be deployed as CPS oracle with respect to R1, R2, R3 and R6.

VII. SUMMARY

This paper presented a comprehensive survey of specification mining oracles in software testing, covering different static analysis-based and dynamic analysis-based oracles. The paper also presented an analysis of CPS system requirements from the oracle perspective and a framework to assess the effectiveness of specification mining oracles for CPS testing. Dynamic and static specification mining supplement each other and each has its own capabilities and limitations that we documented in this survey. Our study indicated that there is no specification mining technique in the literature, satisfying all CPS requirements; particularly *sound* and complete oracle requirements. As a future work, we aim to propose a hybrid technique to build a highly accurate specification-based oracle with low overhead for CPS systems.

REFERENCES

[1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE transactions on software engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[2] M. R. Aliabadi, A. A. Kamath, J. Gascon-Samson, and K. Pattabiraman, "Artinali: dynamic invariant detection for cyber-physical system security," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 349–361.

[3] G. Jahangirova, D. Clark, M. Harman, and P. Tonella, "Test oracle assessment and improvement," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 247–258.

[4] G. Jahangirova, "Oracle problem in software testing," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2017, pp. 444–447.

[5] S. R. Shahamiri, W. M. Wan-Kadir, S. Ibrahim, and S. Z. M. Hashim, "Artificial neural networks as multi-networks automated test oracle," *Automated Software Engineering*, vol. 19, no. 3, pp. 303–334, 2012.

[6] D. Wagner and R. Dean, "Intrusion detection via static analysis," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 156–168.

[7] J. T. Giffin, S. Jha, and B. P. Miller, "Efficient context-sensitive intrusion detection." in *NDSS*, 2004.

[8] D. Engler, B. Chelf, A. Chou, and S. Hallem, "Checking system rules using system-specific, programmer-written compiler extensions," in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association, 2000, p. 1.

[9] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "Slam and static driver verifier: Technology transfer of formal methods inside microsoft," in *International Conference on Integrated Formal Methods*. Springer, 2004, pp. 1–20.

[10] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*. Springer, 2000, pp. 154–169.

[11] M. Gabel and Z. Su, "Symbolic mining of temporal specifications," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 51–60.

[12] J. Jaffar, J. A. Navas, and A. E. Santosa, "Unbounded symbolic execution for program verification," in *International Conference on Runtime Verification*. Springer, 2011, pp. 396–411.

[13] C. Lemieux, D. Park, and I. Beschastnikh, "General ltl specification mining (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 81–92.

[14] J. Abrahamson, I. Beschastnikh, Y. Brun, and M. D. Ernst, "Shedding light on distributed system executions," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 598–599.

[15] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal api rules from imperfect traces," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 282–291.

[16] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 35–45, 2007.

[17] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 2002, pp. 291–301.

[18] C. Csallner, N. Tillmann, and Y. Smaragdakis, "Dysy: Dynamic symbolic execution for invariant inference," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 281–290.

[19] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 501–510.

[20] D. Lo and S. Maoz, "Specification mining of symbolic scenario-based models," in *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2008, pp. 29–35.

[21] T. Ohmann, K. Thai, I. Beschastnikh, and Y. Brun, "Mining precise performance-aware behavioral models from existing instrumentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 484–487.

[22] X. Wu, C. Zhang, and S. Zhang, "Efficient mining of both positive and negative association rules," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 3, pp. 381–405, 2004.

[23] P. Bian, B. Liang, W. Shi, J. Huang, and Y. Cai, "Nar-miner: discovering negative association rules from code for bug detection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 411–422.

[24] M. Vanmali, M. Last, and A. Kandel, "Using a neural network in the software testing process," *International Journal of Intelligent Systems*, vol. 17, no. 1, pp. 45–62, 2002.

[25] K. Aggarwal, Y. Singh, A. Kaur, and O. Sangwan, "A neural net based approach to test oracle," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 3, pp. 1–6, 2004.

[26] S. R. Shahamiri, W. M. N. W. Kadir, and S. bin Ibrahim, "An automated oracle approach to test decision-making structures," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 5. IEEE, 2010, pp. 30–34.

[27] A. H. Ko, R. Sabourin, and A. S. Britto Jr, "From dynamic classifier selection to dynamic ensemble selection," *Pattern Recognition*, vol. 41, no. 5, pp. 1718–1731, 2008.

[28] L. M. Vriesmann, A. S. Britto, L. S. Oliveira, A. L. Koerich, and R. Sabourin, "Combining overall and local class accuracies in an oracle-based method for dynamic ensemble selection," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–7.

[29] R. M. Cruz, R. Sabourin, G. D. Cavalcanti, and T. I. Ren, "Meta-des: a dynamic ensemble selection framework using meta-learning," *Pattern recognition*, vol. 48, no. 5, pp. 1925–1935, 2015.

[30] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[31] F. Xing, P. Guo, and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," in *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*. IEEE, 2005, pp. 10–pp.

[32] L. Tian and A. Noore, "Dynamic software reliability prediction: an approach based on support vector machines," *International Journal of Reliability, Quality and Safety Engineering*, vol. 12, no. 04, pp. 309–321, 2005.

[33] F. Wang, J.-H. Wu, C.-H. Huang, and K.-H. Chang, "Evolving a test oracle in black-box testing," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2011, pp. 310–325.

[34] C. Zheng, J. Wu, F. Peng, and Z. Wu, "Software life cycle-based defects prediction and diagnosis technique research," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, vol. 8. IEEE, 2010, pp. V8–192.

[35] V. Murali, S. Chaudhuri, and C. Jermaine, "Bayesian specification learning for finding api usage errors," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 151–162.