Automata Learning Enabling Model-Based Diagnosis

Edi Muškardin[1,2], **Ingo Pill**[1], **Martin Tappler**[2,1] and **Bernhard K. Aichernig**[2]
[1]Silicon Austria Labs, TU Graz - SAL DES Lab, Graz, Austria
{edi.muskardin,ingo.pill}@silicon-austria.com
[2]Graz University of Technology, Institute of Software Technology, Graz, Austria
{aichernig,martin.tappler}@ist.tugraz.at

Abstract

The lack of a diagnostic model often prohibits us from deploying diagnostic reasoning for reasoning about the root causes of encountered issues. For overcoming this obstacle, we discuss in this manuscript how to exploit active automata learning for learning deterministic and stochastic models from black-box reactive systems for diagnostic purposes. On one hand, we can learn models of faulty systems for being able to deploy model-based reasoning. Furthermore, we will also show how to exploit fault models in the learning process, such as to derive a behavioral model describing the entire corresponding diagnosis search space. In terms of applications, we will discuss several concrete diagnosis scenarios and how our models can be exploited accordingly, as well as report first experiments and corresponding results.

1 Introduction

When a system does not behave as expected, model-based diagnosis [1; 2; 3; 4; 5] has proven to be a well-structured and complete approach to search for explanations, i.e., diagnoses in the form of component sets that can explain the behavior—if we assume these components to be faulty. In the reasoning process, we take a specific system model into account and then reason towards the diagnoses by investigating the effects of the components' faults on the behavior. In particular, this means assessing the effects of assuming their malfunction, either with a weak fault model where we do not restrict the faulty behavior of a component at all, or with dedicated fault models like a stuck-at-1 for a logic gate (see also Sec. 2). The motivation is to derive hypotheses concerning diagnoses and assess whether fault assumptions specified in a diagnosis hypothesis make the observed behavior consistent with our expectations from a system. Algorithms like RC-Tree [4] allow us to conduct an efficient and complete search in the diagnosis space while possibly considering constraints, e.g., in terms of cardinality.

Often, we lack the required model though, so that concepts where we create knowledge bases concerning the effects of faults in terms of observable symptoms and then abductively reason from observed symptoms towards diagnoses [6] can be very valuable. There we do not consider consistency but look for sets of faults that entail the observed symptoms in their union. Creating and maintaining corresponding knowledge-bases manually is a complex procedure, but this task can also be tackled using automated approaches based on simulation and fault injection/mutation [7]. Spectrum-based fault localization [8; 9] where we only need execution data about which component was involved in which (failing or correct) behavior, i.e. a spectrum, is another (and light-weight) alternative. There we consider similarity coefficients [8] between the individual behaviors' failing/correctness and a component's involvement in the respective behavior in order to establish a ranking of the likelihoods of a system's components to be at fault—in contrast to computing diagnoses as explanations.

In our work, while we assume a black box scenario, we neither create an abductive knowledge base, nor focus on execution data in the form of spectra. Rather, we focus on actively learning a formal model from a system, with the only requirement being that during the learning process we need to be able to interact with the system or a simulation. That is, during learning we provide stimuli and need to be able to observe a system's reactions. In particular, we focus on learning formal models in a minimal teacher environment that allows us to deploy model-based diagnostic reasoning. Such an environment was initially proposed in [10] for regular language inference. Over the years, several extensions have been proposed. With AALPY [11], our automata learning library, we developed, designed, and improved algorithms to extend the learning to deterministic, non-deterministic and stochastic models. Learning such formal models then allows us to deploy formal reasoning in corresponding scenarios for black-box systems.

In this manuscript, we discuss how to principally exploit learned models, and the learning process itself in a diagnostic context. So, first we show how active automata learning can be used to (a) learn the language of faults/property violations, (b) learn the deterministic and stochastic fault models, and (c) learn the models suitable for diagnostic reasoning. Furthermore, we reason about the applicability of active automata learning in a diagnostic setting and provide several examples to demonstrate the presented concepts.

2 Preliminaries

When talking about model-based diagnosis, we refer to the consistency-oriented reasoning characterized in [3] or [1] for single scenarios (see [5] for diagnoses for multiple scenarios). In this reasoning, a system model *SD* describes a system's behavior in sentences of the form $h_i \rightarrow$ *nominal behavior of* c_i. In principle, we state that under the assumption that some component c_i is correct (encoded in a health state variable $h_i \in H$), we know how c_i behaves.

Providing such sentences for all components, and complementing them with connections and background knowledge, we can define a system model for diagnostic purposes. In this simple form, we make no assumptions about how the components behave in case of a fault and thus implement a *weak fault model*. Given some observations *OBS* about the system's actual behavior, we can reason now with *SD* whether *OBS* is consistent with the expected behavior (described in *SD*) under the assumption that all components work as expected (such that $SD \cup OBS \cup \{h_i | h_i \in H\}$ is consistent or satisfiable). If this is not the case, we can furthermore define and verify weak fault model hypotheses $\Delta \subseteq H$ concerning faulty components (a diagnosis hypothesis)—verifiable via checking $SD \cup OBS \cup \{h_i | h_i \in H \setminus \Delta\}$. This is exploited also in a structured exploration of the diagnosis search space like in RC-Tree[4].

Definition 1. *A diagnosis for a diagnosis problem* (SD, H, OBS) *is a subset-minimal set* $\Delta \subseteq H$ *such that* $SD \cup OBS \cup \{h_i | h_i \in H \setminus \Delta\}$ *is consistent (satisfiable).*

Alternatively to using a weak fault model, we can describe also faulty behavior (like a stuck-at-one fault for an AND-gate) so that h_i becomes in principle a selector variable for selecting between some well-defined behavioral variants. Such a strong fault model (SFM) approach as defined in [12] has the advantage that diagnoses become more specific since a diagnosis Δ defines a specific behavior also for each faulty component (and not only the correct ones). It also entails the issue of the diagnosis search space growing from $2^{|H|}$ to $O(m^{|H|})$ though (with m being the maximum number of possible modes for any $h_i \in H$). While we often assume faults to be persistent, there is also work on intermittent faults, e.g., [13].

Automata Learning. We apply Angluin's L^* algorithm and variations thereof [10] to mine finite-state models. L^* is an active automata learning algorithm in the minimally adequate teacher (MAT) framework. Such learning algorithms infer automata by interacting with the teacher through queries. For illustrating L^*-based learning, let us assume that we aim to learn a deterministic finite automaton (DFA) accepting an unknown regular language L over alphabet Σ. The learner starts by posing automatically constructed membership queries to the teacher. A membership query checks whether a word over Σ is in L. Once the learner has sufficient membership information and thus creates a hypothesis DFA H, it performs an equivalence query. Such a query checks whether H accepts exactly L. In case of a positive response from the MAT, learning can be stopped with H as result. Otherwise, the teacher provides a counterexample to equivalence, which is a word in the symmetric difference between L and the language accepted by H. In that case, the learner then integrates the counterexample into its knowledge and starts a new learning round.

L^* has been extended to various types of automata, generally with underlying regular languages. In this paper, we learn DFAs, Mealy machines [14], and Moore machines using L^*. The latter two produce input-output traces that form regular languages $L_{io} \subseteq (I \times O)^*$ for an input alphabet I and an output alphabet O. These machines are deterministic, i.e., for every input sequence $i \in I^*$ there is exactly one output sequence $o \in O^*$ of equal length s.t. their pair-wise combination is in L_{io}. This changes membership and equivalence queries: a membership query takes an input sequence i as input and returns the output sequence s that should be produced in response in accordance with L_{io}. A counterexample to equivalence is an input sequence such that the hypothesis disagrees with L_{io}.

Test-Based Automata Learning. In theory, a teacher needs perfect knowledge of L in order to answer equivalence queries. However, in a black-box approach this assumption does not hold. The absence of exact equivalence queries is commonly approached by simulating such queries by randomly sampling words and asking membership queries [10; 15]. We take a test-based view of black-box automata learning, where we implement membership queries and equivalence queries through testing of the system under learning (SUL), i.e., the system from which we want to learn a model. This is a common approach in automata-learning-based testing and verification [16]. Rather than sampling words completely randomly for equivalence queries, we use conformance-testing techniques to select words (a single word w refers basically to a test case). To perform a membership query for w, we provide w to the SUL, and an equivalence query is implemented by a series of membership queries.

Conformance testing usually takes a specification model of a software system and generates test cases from this model to reveal non-conformance between model and system [17]. Here, we take a hypothesis model H and generate test cases to reveal non-equivalence between H and the SUL. In other words, we want to find words where H disagrees with the SUL. Please note that we use equivalence as conformance relation. When implementing equivalence queries via testing, automata learning may not find the true, correct automaton underlying the SUL due to the inherent incompleteness of testing. Instead, learning will halt upon finding a hypothesis that is deemed to conform to the SUL. An important property of L^*-based learning is that the *learned hypothesis is the minimal automaton*, in terms of the number of states, that is consistent with the queried information [10]. This means that additional information (counterexamples) adds states. Hence, we can compare equivalence-query implementations based on the size of learned models. A larger model means that the corresponding equivalence-query implementation is better and found more counterexamples.

When learning Mealy or Moore machines, the basic approach remains the same. The test cases are input sequences and our goal is to find an input sequence, where the output sequence produced by hypothesis disagrees with the output sequence produced by the SUL.

3 Enabling Model-Based Reasoning With Automata Learning

In this section, we will show how automata learning can enable model-based reasoning trough model learning. Learned models are suitable for debugging, strategy synthesis, monitoring, and diagnostic reasoning. For the methods outlined in Sect. 3.1, we assume that the system under learning is intrinsically faulty. On the other hand, we outline in Sect. 3.2 how one can mine models to be used for diagnostic reasoning via automata learning and fault injection—following a similar motivation as behind approaches for automatically creating knowledge-bases for abductive diagnosis [7] as mentioned in the introduction.

3.1 Assessment and Recovery in Faulty Systems

In the remainder of this section, we consider a faulty system, i.e., a system with an inherent fault. Faults are cases of non-conformance between the system behavior and its specification, where such faults can manifest themselves deterministically or stochastically. In a deterministic setting, faults are found upon an execution of an input sequence that always reveals the violation of the specification. On the other hand, an input sequence may or may not lead to a fault in a stochastic setting. In particular, we have that inputs in a stochastic setting/automaton lead to a set of new states, where we reach those from the current state with specific probabilities.

For a deterministic setting, Kunze et al. [18] outlined how one can use automata learning to generate/learn failure models. Building on their high-level approach, Khoo et al. [19] showed how fault models of cyber-physical systems could be learned with automata learning. Finally, Chockler et al. [20] applied active automata learning to learn a language of software errors.

All listed approaches follow the same high-level approach that assumes the existence of an oracle. An oracle considers some property defined by the specification and returns a Boolean verdict indicating whether said property has been violated. With such knowledge, one can learn a DFA whose accepting states are the ones causing a property violation. This DFA can be seen as a language of faults and can be used to construct many new, previously unseen, test cases/scenarios that lead to the property violation. Note that [18] formalized learned automata as Mealy machines, but outputs were consistent with the Boolean verdict indicating whether the fault occurred (*ok, fail*), therefore making them structurally identical to DFAs. For completeness, we show how such models can be learned in Sect. 4.2.

We propose a generalization of this approach. Whereas previously mentioned approaches only learn a subset of the system's behavior (constrained to a property/fault), we surpass this limitation by learning the system's whole input-output behavior. Instead of using an oracle that provides an output whether a property violation has occurred, we consider the system's outputs. By doing so, our model is a (deterministic or stochastic) finite-state transducer instead of a DFA. To make learning feasible, we use a mapper [21] to form abstract equivalence classes over concrete inputs and outputs. Elements of the abstract equivalence class are characterized by the same future input-output behavior on the SUL. Depending on the system, an oracle might be used during learning as part of the mapper component [21] assigning fault labels to property violations. Corresponding automated oracles, e.g., for FLTL properties [22] sometimes also feature diagnostic support for isolating the parts of a property triggering the violation [23] which gives more insight into the failing behavior. In principle, we can also diagnose the resulting automaton similar to an approach used for diagnosing specifications, i.e., where the authors created a symbolic automaton from the specification for that purpose [24]. An open question would be though where to introduce the health state variables for achieving a balance between the size of the search space on one hand, and representative and effective feedback given by a diagnosis on the other hand. In particular, in the current setting we cannot draw on the focus of a specification's subformulae, but consider directly the automaton. Thus the placement of the health state variables (abnormal predicates) for an efficient and effective diagnosis process is an open question.

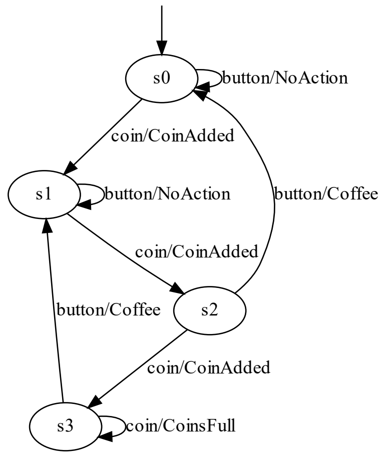

Figure 1: Fault-free behavior of the coffee machine.

To the best of our knowledge, related work only considered deterministic faults and system behavior. We postulate that this assumption can be a limiting factor in practice. Consequently, we extend the active learning of fault models to stochastic reactive systems [25]. More concretely, we learn the system's input-output behavior and represent it as Markov Decision Process (MDP) or as a Stochastic Mealy Machine (SMM). Learned stochastic fault models can be used for debugging, model checking, and strategy synthesis. For example, one can synthesize a strategy that completely avoids faults or minimizes the probability of a fault occurring. However, the faults might be unavoidable due to the stochastic nature of the system. In that scenario, knowledge about the consequences of the fault can be exploited to compute a fault mitigation/compensation strategy.

In Sect. 4.2, we show how one can learn the models of faulty reactive systems and reason about the practical application of these models.

3.2 Learning-Based Monitoring and Diagnosis of Reactive Systems

Section 3.1 outlined how we can use active automata learning to mine models of faulty reactive systems. Those models can be used for model-checking and strategy synthesis. They can reveal input sequences that lead to a fault, but we cannot use them to reason about the causes of said faults easily. To mitigate this problem, we show how active automata learning and fault injection [26] can be used to mine models suitable for model-based diagnosis. Availability of the simulated behavior of faults is supported by standards like ISO 26262-1[1], that suggest fault injection as a method for ensuring safety in the automotive domain.

In the previous section, we assumed that the SUL is inherently faulty and the input alphabet used for learning contains all possible (abstract) inputs to the system. In this section, however, we assume that the SUL behaves according to the specification. We can further explicitly trigger faulty behavior through fault-injection inputs. Once a fault is injected, a

[1]`https://www.iso.org/standard/68383.html`

system's future behavior may differ from the specification. We therefore extend the input alphabet with abstract inputs that correspond to fault injections.

This approach enables injecting different types of faults, such as intermittent faults. In our experiments, fault injection during learning considers the current state, whose history defines previously injected faults. Based on such history, fault injection might be ignored, for instance, to limit the number of simultaneously activated faults in a given component. Allowing at most one fault per component can be achieved, by ignoring faults if the corresponding component is already in a faulty state. We generally limit the maximum number of injected faults, since the size of learned models grows exponentially in the number of active faults.

Once a fault-injected model has been learned, we can use it for monitoring and diagnostic reasoning. As the system operates, we trace its behavior through the automaton. When we detect a discrepancy between the expected system behavior and the observed behavior for some trace t, we simulate t on the fault-injected model to find a diagnosis. A diagnosis in this setting is a sequence of pairs of inject faults and the position in t where the corresponding fault may have been injected.

In line with approaches to diagnosis of discrete-event systems [27; 28], we partition the set of possible events and use trace projections to discard events from traces. We concretely partition the input alphabet Σ for learning into Σ_n and Σ_f. The former contains all normal inputs, whereas the latter contains input symbols representing faults being injected. Suppose that Λ is the output alphabet of our system, we observe traces in $(\Sigma_n \times \Lambda)^*$. The traces of the fault-injected learned model M are $T(M) \subseteq (\Sigma \times \Lambda \cup \{\tau\})^*$, where τ is produced by fault-injection inputs in Σ_f.

For diagnosis, we associate each $t \in T(M)$ with a trace in Σ_n by projecting it onto $\sigma(t) \in (\Sigma_n \times \Lambda)^*$, i.e., removing input-output pairs in $\Sigma_f \times \{\tau\}$ from t. Let λ denote the empty trace, then projection is defined by:

$$\begin{aligned}\sigma(\lambda) &= \lambda\\ \sigma((i,o)\cdot t) &= t \text{ if } i \in \Sigma_f\\ \sigma((i,o)\cdot t) &= (i,o)\cdot\sigma(t) \text{ otherwise}\end{aligned}$$

To diagnose a faulty trace $t \in (\Sigma_n \times \Lambda)^*$, we first determine $T_d(t) = \{t_d \in T(M) \mid \sigma(t_d) = t\}$. Each $t_d \in T_d$ yields a different diagnosis that is a sequence of pairs $(\mathit{fi}, \mathit{pos})$. The pair element fi is a fault injection in Σ_f and pos is the index of the next normal input in t_d, where fault-injection inputs are not counted, thus a diagnosis may contain elements with the same pos.

We implement diagnosis by simulating t on the fault-injected learned machine M and the non-fault injected machine M_{corr}, while treating fault-injection inputs as ϵ transitions. By doing that, we compute all interleavings of normal inputs and fault-injection inputs that are consistent with t. To avoid diagnoses that are likely irrelevant, we may divide the computation into two parts: (1) the first part computes the longest prefix t' of t that is only consistent with non-faulty behavior – such a t' reaches exactly one state in M and is observable in M_{corr}. (2) Only from the state reached by t', we then compute the interleavings of fault injection and normal operation. The first part essentially avoids sub-diagnoses related to intermittent faults that produce behavior conforming to the specification.

3.3 Practical Considerations and Limitations

In this section, we reason about the efficiency of applying automata learning for diagnostic model mining. We isolated three challenges that could hinder the applicability of the proposed approaches. Those are (a) non-regular behavior of SUL, (b) high-dimensional input not suitable for abstraction, and (c) state-space explosion caused by fault injection.

In the context of this paper, automata learning is used to learn regular languages. If the system's behavior cannot be captured by a regular language (e.g. the system's behavior is context-free), we cannot learn the complete model of the system, since equivalence queries may find infinitely many counterexamples. This limitation can be mitigated by introducing a second stopping criterion in addition to positive results from equivalence queries. One approach could be limiting the scope of the equivalence oracle to specific parts of the input space. For example, we could explore the consequences of a fault up to the predefined depth. This is a proposed solution if the fault introduces non-regular behavior in the otherwise regular language describing the system's input-output behavior.

Abstraction over input and output alphabet was addressed in the Sect. 3.1. However, abstraction as proposed by Aarts et at. [21] is not suitable for high-dimensional inputs (i.e., if the inputs or outputs of the system are arrays of floating-point values). In such scenarios, large input alphabets make automata learning infeasible.

Construction of diagnostic models outlined in Sect. 3.2 suffers from state-space explosion. This can be mitigated by limiting the number of injected faults or by assuming persistent instead of intermittent faults. Furthermore, injecting faults only in some states or parts of the automaton could limit the increase in the learning complexity while providing models capturing the consequences of faults.

4 Examples

To evaluate our approach, we implemented several reactive systems and learned them with AALPY. We will demonstrate techniques outlined in this paper on several variants of a coffee machine[2]. Modeling more complex systems in modeling languages such as Modelica [29] are supported, but for simplicity, we focus on this simple reactive system that we developed in Python.

Figure 1 depicts the correct behavior of the coffee machine, where users can interact with the it by adding coins or by pushing a button. A coffee machine can store up to three coins, and all further coin insertions are ignored. Coffee costs two coins. If a user has provided two or three coins, he can push the button and get the desired coffee, consuming two coins.

4.1 Learning a Language of Errors

The specification of the coffee machine described in Sect. 4 and depicted as Mealy Machine in Fig. 1 shows that the cost of coffee should always be two coins. Therefore, a property *(num(coins)* $\geq$ *2* $\wedge$ *action == button)*$\rightarrow$ *output(coffee)* states that coffee should be obtained iff two or more coins were inserted and the button was pressed.

[2]All examples presented in the paper can be found at: `https://github.com/DES-Lab/Automata-Learning-Based-Diagnosis`

We now create a faulty coffee machine. If the number of coins in the coffee machine is three (max. number of coins) and a button is pressed, this coffee machine will return a coffee, but at the cost of one coin.

Figure 2 shows a learned language in the form of a DFA that violates said property. This language generates scenarios where a user first inserts three coins and pushes a button, reaching state $s4$. Upon pressing a *button* in state $s4$ coffee is obtained at the cost of one coin reaching state $s5$. Execution of the input sequence *(button, coin)* from the state $s4$ will result in infinitely many coffees at the cost of one coin.

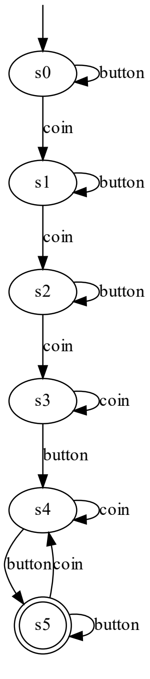

Figure 2: Learned language of a property violation.

4.2 Learning a Fault Model

The same faulty coffee machine can be modeled by learning its input/output behavior. Instead of focusing on one specific property, we learn the entire behavior of the SUL. This model can then later be manually analyzed or model checked. We observe that the complete failure model depicted in Fig. 3 is smaller than the language of property violation shown in Fig. 2. Note that this is not the case in general. The language a property violation will explore only parts of the automaton that are related to the property. The same can be achieved with input-output models by limiting the scope of exploration in the equivalence oracle.

So far, we have considered only deterministic faults. Suppose that the coffee machine has two stochastic faults. We want to learn a model that encodes the consequences of said faults and the probability of their occurrence. We can reason about the faulty stochastic behavior of the coffee machine with the help of the learned model shown in Fig. 4. Firstly, if there are fewer than two coins in the coffee machine, a user has a 2% chance of getting a coffee by pressing a button. Secondly, if the user inserts three coins and keeps inserting coins, there is a 20% chance that the coffee machine will return all coins the user has inserted instead of just one. Even on this trivial example we can see how a strategy can be synthesized. To minimize the cost of coffee, one only has to press the button repeatedly until a free coffee is obtained (with probability of 2%).

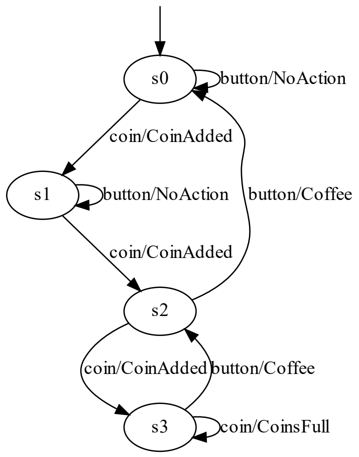

Figure 3: Learned deterministic fault model.

4.3 Learning a Diagnostic Model

Suppose we are given a coffee machine as our SUL. Its inputs are $\Sigma_n = \{$*coin, button*$\}$. In the fault-free behavior, the coffee machine will return *coffee* if more than two coins are inserted and *NoAction* otherwise. Let the possible faults be *button_no_effect* and *add_two_coins* in Σ_f. The first fault causes the button to have no effect, while the latter increments the internal counter as if two coins were inserted instead of one. For simplicity, both faults are permanent once triggered and only one fault can be triggered at any time.

In the fault-free behavior, the input sequence is = *(coin, coin, button)* results in the output sequence os = *(coinAdded, coinAdded, coffee)*. Suppose that during system monitoring, said input sequence produces the output sequence os' = *(coinAdded, coinAdded, NoAction)*. We observe that expected and actual observations differ and we are interested in the fault that caused this difference. We compute the diagnosis as outlined in Sect. 3.2 given the learned model shown in Fig. 5 (τ is denoted *True* in the figure). In order to diagnose the faulty trace $t = is/os'$, we compute $T_d(t)$. This set includes, for example, the traces $(\mathit{button\_no\_effect}, \tau) \cdot (\mathit{coin}, \mathit{coinAdded}) \cdot (\mathit{coin}, \mathit{coinAdded}) \cdot (\mathit{button}, \mathit{NoAction})$

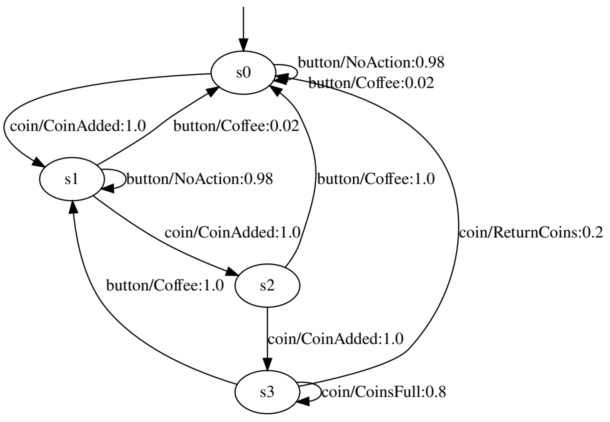

Figure 4: Learned stochastic fault model.

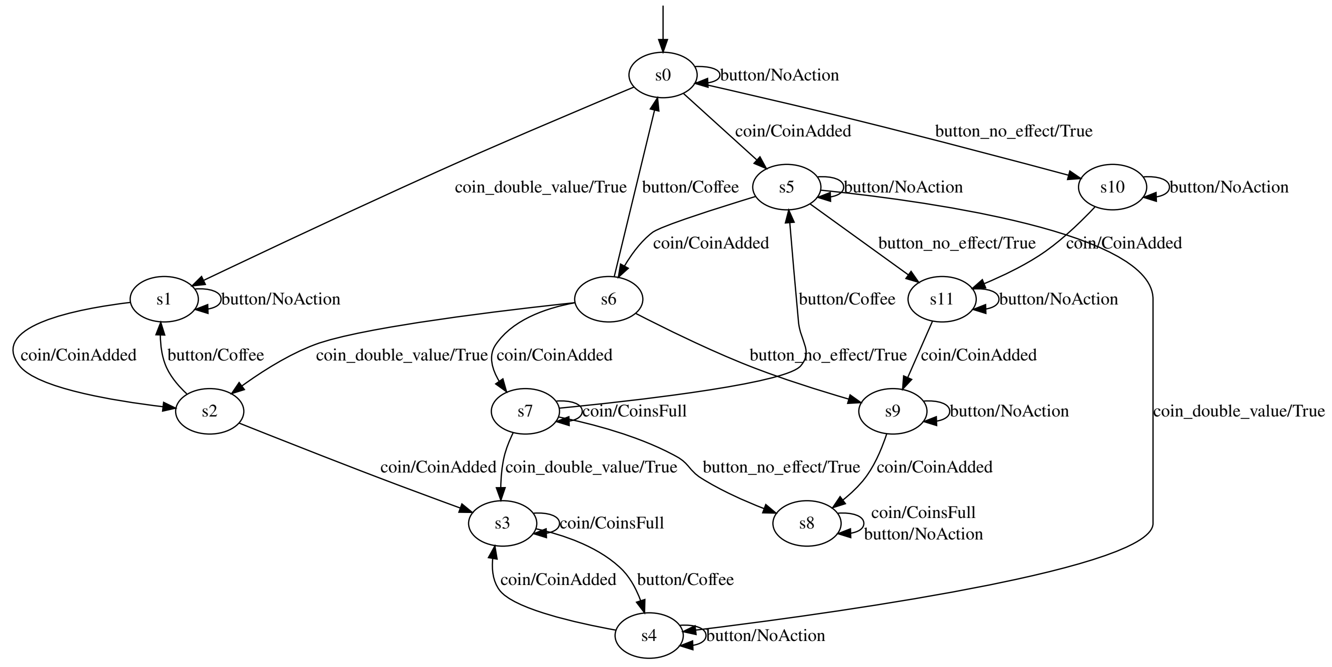

Figure 5: Learned diagnostic model.

and $(coin, coinAdded) \cdot (coin, coinAdded) \cdot (button\_no\_effect, \tau) \cdot (button, NoAction)$. Using one-based indices, this yields the diagnoses $(button\_no\_effect, 1)$ and $(button\_no\_effect, 3)$. Hence, the behavior may have been caused by a faulty button.

4.4 More Advanced Examples

The coffee machine example and its variations served as a proof of concept, and their small size made them suitable for manual analysis and visualization.

To further assess the feasibility and applicability of the proposed approaches, we implemented several simulated reactive systems. This section will examine how the proposed methods could be applied to those systems.

Gearbox and Clutch. A gearbox system where gears can be changed and the clutch can be pressed and released. If the gearbox is set to reverse twice from any gear (but the first) the gearbox breaks. A fault language (40 state DFA) and fault model (36 state Mealy Machine) of the gearbox systems were successfully learned. Both models can be used to construct previously unseen scenarios that lead to failure/property violation.

Vending Machine. A simple vending machine where the user can input three different coins. Once a sufficient amount of coins have been inserted, users can select one of the offered items from the vending machine. A specific combination of inputs invokes a fault in the vending-machine logic. By learning a fault model (90 state Mealy Machine), we successfully found the sequences that lead to the fault. During learning, the structure of the intermediate hypotheses was exploited when performing equivalence queries. A purely random equivalence oracle might not be able to learn a complete model.

Crossroad with Traffic Lights and Pedestrians. A crossroad where two traffic lights control the flow of traffic. The traffic control units balances the number of queued cars in each direction (north-south and east-west). Each direction also has a pedestrian button. The pedestrian button increases the priority of its respective lane. Each lane has a sensor that tells the control unit how many cars are queued. A fault can be injected into each sensor. If the sensor is faulty, it might lead to a traffic jam, as the control unit may not adequately react to newly arriving cars. Learning resulted in a 3807 state diagnostic model and took 150 seconds.

Differential Drive Robot. A simple robot with two wheels. The difference in speed of both wheels determines the direction of the robot. The maximum speed of each wheel is ten units. The speed is capped, because placing no limit on maximum speed results in a context-free language. Each wheel can speed up, slow down, or keep its current speed at each time step. What is more, each wheel can be in one of three fault modes: spin faster, slower, or stuck.

MQTT and TCP protocols. Deterministic models of MQTT and TCP protocols were successfully learned with active automata learning [30; 31]. Stochastic faults were injected in the deterministic models capturing the abstracted input-output behavior of both protocols. Protocol fault models of were successfully learned, resulting in 17-state (MQTT) and 12-state (TCP) stochastic Mealy machine.

5 Conclusion

In this paper, we presented an active automata learning approach at model-based diagnosis. We have shown how automata learning can be used to learn three types of models: the language of property violations as DFA, models capturing a faulty system's input-output behavior, and diagnostic models. Learned models are suitable for debugging, verification, test-case generation, and diagnostic reasoning.

While we showed principal viability via simple proof-of-concept examples we will apply the proposed methods to a real-world case study. If a live interaction with the system under learning is not possible, as in safety-critical systems that are already online, we might consider the automatic construction of diagnostic models via passive automata learning. Passive automata learning does not interact with the system under learning but constructs models consistent with previously observed system traces (logs). Furthermore, we will develop methods that support feasible automata learning in complex, dynamic cyber-physical systems. Another point of interest will be the development of automated input-output abstraction methods.

References

[1] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[2] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artif. Intelligence*, 41(1):79–88, 1989.

[3] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[4] I. Pill and T. Quaritsch. RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm. In *IEEE Int. Symp. on Software Reliability Engineering Workshops (ISSREW)*, pages 78–84, 2015.

[5] I. Pill and F. Wotawa. Computing multi-scenario diagnoses. In *31st International Workshop on Principles of Diagnosis*, 2020.

[6] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Hypothesis classification, abductive diagnosis and therapy. In *First International Workshop on Principles of Diagnosis*, Menlo Park, July 1990. Also appeared in Proceedings of the International Workshop on Expert Systems in Engineering, Lecture Notes in Artificial Intelligence, Vol. 462, Vienna, September 1990, Springer-Verlag.

[7] I. Pill and F. Wotawa. On using an I/O model for creating an abductive diagnosis model via combinatorial exploration, fault injection, and simulation. In *29th International Workshop on Principles of Diagnosis*, 2018. http://ceur-ws.org/Vol-2289/paper9.pdf.

[8] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. Spectrum-based multiple fault localization. In *ASE*, pages 88–99, 2009.

[9] Ingo Pill and Franz Wotawa. Spectrum-based fault localization for logic-based reasoning. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 192–199, 2018.

[10] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[11] Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. AALpy: An active automata learning library. In *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, Australia, October 18-22, 2021, Proceedings*, 2021.

[12] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 1324–1330. Morgan Kaufmann, 1989.

[13] Johan De Kleer. Diagnosing multiple persistent and intermittent faults. In *21st International Jont Conference on Artifical Intelligence*, page 733–738, 2009.

[14] Muzammil Shahbaz and Roland Groz. Inferring Mealy machines. In *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, pages 207–222, 2009.

[15] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[16] Bernhard K. Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. Model learning and model-based testing. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, volume 11026 of *Lecture Notes in Computer Science*, pages 74–100. Springer, 2018.

[17] Angelo Gargantini. Conformance testing. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *Lecture Notes in Computer Science*, pages 87–111. Springer, 2004.

[18] Sebastian Kunze, Wojciech Mostowski, Mohammad Reza Mousavi, and Mahsa Varshosaz. Generation of failure models through automata learning. In *2016 Workshop on Automotive Systems/Software Architectures (WASA)*, pages 22–25, 2016.

[19] Teck Ping Khoo, Jun Sun, and Sudipta Chattopadhyay. Learning fault models of cyber physical systems. In Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony, editors, *Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings*, volume 12531 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2020.

[20] Hana Chockler, Pascal Kesseli, Daniel Kroening, and Ofer Strichman. Learning the language of software errors. *J. Artif. Intell. Res.*, 67:881–903, 2020.

[21] Fides Aarts, Harco Kuppens, Jan Tretmans, Frits W. Vaandrager, and Sicco Verwer. Improving active Mealy machine learning for protocol conformance testing. *Mach. Learn.*, 96(1-2):189–224, 2014.

[22] Ingo Pill and Franz Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139:124–141, 2018.

[23] Ingo Pill and Franz Wotawa. Extending automated FLTL test oracles with diagnostic support. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 354–361, 2019.

[24] Ingo Pill and Thomas Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 1053–1059. AAAI Press, 2013.

[25] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-based learning of Markov decision processes (extended version). *Form. Asp. Comp.*, 2021.

[26] Stuart Reid. Software fault injection: Inoculating programs against errors, by jeffrey voas and gary mcgraw, wiley, 1998 (book review). *Softw. Test. Verification Reliab.*, 9(1):75–76, 1999.

[27] Zh. Feng and Al. Grastien. Model based diagnosis of timed automata with model checkers. In *31st International Workshop on Principles of Diagnosis (DX-20)*, 2020.

[28] Alban Grastien, Patrik Haslum, and Sylvie Thiébaux. Conflict-based diagnosis of discrete event systems: Theory and practice. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012.

[29] Peter Fritzson. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press, September 2011.

[30] Martin Tappler, Bernhard K. Aichernig, and Roderick Bloem. Model-based testing IoT communication via active automata learning. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 276–287. IEEE Computer Society, 2017.

[31] Paul Fiterau-Brostean, Ramon Janssen, and Frits W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2016.