Revisiting AI and Testing Methods to Infer FSM Models of Black-Box Systems

Roland Groz
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
Roland.Groz@univ-grenoble-alpes.fr

Adenilso Simao
Universidade de Sao Paulo, ICMC
Sao Carlos/Sao Paulo, Brasil
Adenilso@icmc.usp.br

Nicolas Bremond
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
Nicolas.Bremond2@univ-grenoble-alpes.fr

Catherine Oriat
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
Catherine.Oriat@univ-grenoble-alpes.fr

ABSTRACT

Machine learning in the form of inference of state machine models has gained popularity in model-based testing as a means of retrieving models from software systems. By combining an old idea from machine inference with methods from automata testing in a heuristic approach, we propose a new promising direction for inferring black box systems that cannot be reset. Preliminary experiments show that this heuristic approach scales up well and outperforms more systematic approaches.

KEYWORDS

Query learning, FSM testing, Software Engineering

ACM Reference Format:
Roland Groz, Adenilso Simao, Nicolas Bremond, and Catherine Oriat. 2018. Revisiting AI and Testing Methods to Infer FSM Models of Black-Box Systems. In *AST'18: AST'18:IEEE/ACM 13th International Workshop on Automation of Software Test , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194733.3194736

1 INTRODUCTION

Machine learning techniques, thanks to the predictive ability of the models they provide, have been used for various software engineering tasks. Among learning techniques, automata learning techniques (and grammatical inference) have been identified as a key enabler for model-driven engineering to compensate for the absence of models or the difficulty of writing or updating behavioural models of software systems.

These models can be used for various purposes: documentation, as in "specification mining" [1], verification with model checkers [11], test generation using model-based testing techniques [4] [10], security analysis [3] etc.

AST'18, May 28–29, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5743-2/18/05…$15.00
https://doi.org/10.1145/3194733.3194736

In this context, and in particular for testing, it makes sense to use active learning inference algorithms, for when testing the system, we can query it by sending inputs to observe its outputs. The corresponding automata models are Mealy machines, usually called Finite State Machines (FSMs).

Query learning has attracted interest in model-based software engineering with Angluin's L^* algorithm [2], which has been adapted to input/output models (e.g. in [7] [6] [14]).

In that context, typical queries are not membership queries (checking whether a given input sequence is accepted), but output queries: sending sequences of inputs and observing the corresponding sequence of outputs. Thus, the inference algorithm will exploit traces: sequences of input/output pairs. The other type of queries required by L^*, equivalence queries — whereby a so-called "oracle" will either confirm equivalence or provide a counterexample — have to be approximated in software testing contexts.

Most algorithms used so far have considered that the system that is inferred can be reliably reset, so that it is possible to root the observed traces to a fixed known initial state. However, in many black box contexts, typically when a system is queried over a network, the system under inference (SUI) cannot be reset. In other cases, although the system can be reset, each reset is very costly, especially with respect to time. As a typical example, interacting over a local network for querying a web system on a virtual machine takes around a millisecond for a single input/output observation, whereas resetting a virtual machine takes typically almost over a minute; thus, the reset typically costs much more than an I/O observation.

Rivest and Schapire [13] had addressed the problem of inferring without reset by using a variant of the L^* algorithm with the assumption that a homing sequence was given; a homing sequence is a fixed input sequence such that the output observed completely determines the state reached at the end of the sequence [see 9]. This approach was exemplified in a robotics framework with autonomous robots.

A different approach was proposed by Groz et al. [5]. Instead of relying on a homing sequence and the classical learning algorithm L^*, it uses two classical assumptions from FSM testing. First, it assumes a bound on the number of states of the SUI. Moreover, instead of a homing sequence, it starts from a given characterization set, a.k.a. W-set, in reference to Vasilievskii [15]. A W-set is a set of input sequences that, when applied from a given state, make it possible to know what is this state. The advantage as compared to Rivest and Schapire's approach is that this algorithm no longer requires an oracle. Implementing a precise oracle that can answer equivalence queries cannot be done for an unknown software system, so it is usually approximated. However there is a difficulty: since the W-set could contain several sequences, it implies that, for a system which cannot be reset, the algorithm must ensure returning to a state where a previous sequence was applied, from the same state. This is achieved by a recursive procedure called a localizer. The absence of an oracle is compensated by assumption that there is a known bound on the number of states.

Both approaches [5, 13] rely on an initial assumption that either a homing sequence or a characterization set is known for the SUI. Actually Rivest and Schapire [13] showed that it is possible to avoid such an assumption by going for a probabilistic algorithm (in that case, they use a bound on the number of states). Recently, another deterministic approach has been proposed that avoids any initial knowledge apart from a bound on the number of states, and it does not require any oracle [12]. However, it does not scale beyond a dozen states for random machines (and 7 states in the usual worst case of Moore locks).

In this paper, we propose a new approach that combines the learning methods from Rivest and Schapire [13] with the approach inspired by conformance testing [5]. It can infer non-resettable systems with no a priori knowledge on the system, and scales up to state machines that have thousands of states.

We call this approach the hW-inference method. It uses an approximate homing sequence h that is progressively learnt, along with a partial characterization set W that is similarly refined. It is a form of optimistic heuristic that infers from approximate h and W as if they were really homing and characterizing; if they are not, this implies that distinct states from the system can be confused and merged in the learnt model, leading to apparent non-determinism that will provide information to refine h or W. An oracle is still needed to confirm that a learnt model is adequate; however experiments show that hW-inference works well with approximated oracles.

Inference is closely linked to conformance testing, and can be seen as another way of checking the behavioural model implemented by a black-box system [12]. Therefore, the hW-inference method can be used for automating the test of a non-resettable black-box system. It produces a sequence of inputs that will exhaustively check the behaviour of the system. In the process, it will either raise internal software failures (e.g., exceptions), or provide a model that can be checked for required properties.

2 DEFINITIONS

In this section, we recall a few classical definitions for the type of automata we are considering here, namely Mealy machines, which we shall call indifferently Finite State Machines, FSM for short. A Finite State Machine is a complete deterministic Mealy machine. Formally, it is a 6-tuple $M = (Q, q_0, I, O, \delta, \lambda)$ where

- Q is a finite set of states with the initial state q_0,
- I is a finite set of inputs (the input alphabet), and O a finite set of outputs,
- $\delta : Q \times I \rightarrow Q$ is the transition mapping, and $\lambda : Q \times I \rightarrow O$ is the output mapping.

Notations δ and λ are lifted to sequences, including the empty sequence ϵ: $\delta(q,\epsilon) = q$, $\lambda(q,\epsilon) = \epsilon$ and for $q \in Q$, for $\alpha x \in I^*$, $\delta(q,\alpha x) = \delta(\delta(q,\alpha),x)$ and $\lambda(q,\alpha x) = \lambda(q,\alpha)\lambda(\delta(q,\alpha),x)$. We will call $\lambda(q,\alpha)$ the answer or response of the machine in state q to α. And $\delta(q,\alpha)$ will be the tail state of the sequence. We will use $\alpha/\beta \in (IO)^*$ to denote the interleaved sequence of inputs and corresponding outputs observed when α applied to the machine yields the response β. Such a sequence of input/output pairs is called a *trace*. Given a trace $\omega = \alpha/\beta$, $\omega \downarrow I = \alpha$ denotes its input projection, and $\omega \downarrow O = \beta$ its output projection.

For inference without reset, we will assume that the FSM to be inferred is *strongly connected*, i.e., for all pairs of states (q, q') there exists an input sequence $\alpha \in I^*$ such that $\delta(q,\alpha) = q'$.

A sequence of inputs $h \in I^*$ is *homing* iff $\forall q, q' \in Q, \lambda(q,h) = \lambda(q',h) \Rightarrow \delta(q,h) = \delta(q',h)$. In other words, the observed output sequence uniquely determines the state reached at the end of the sequence.

Two states $q, q' \in Q$ are *distinguishable* by $\gamma \in I^*$ if $\lambda(q,\gamma) \neq \lambda(q',\gamma)$. Two states are distinguishable by a set $Z \subset I^*$ if there exists $\gamma \in Z$ that distinguishes them. A set W of sequences of inputs (henceforth conventionally called a W-set, following Vasilievskii [15]) is a *characterization set* for an FSM M if each pair of states is distinguishable by W.

Given a characterization set W, rather than naming or numbering states, we may refer to a state by its *state characterization*. A state characterization ϕ is actually a mapping from W to O^*, such that $\phi(w) = tr(w) \downarrow O$, where O^* is the output sequence observed when applying w to M in state q. The set of mappings $\Phi_M = \{\phi_1, ..., \phi_m\}$ corresponds to the set of states Q of the machine. Namely, for $\phi \in \Phi_M$ and $q \in Q$, we write $\phi \leftrightarrow q$ if $\forall w \in W, \phi(w) = \lambda(q,w)$. Thus, while inferring an unknown FSM with characterization set W, we will consider the set of mappings Φ_M as its set of states.

3 *HW*-INFERENCE

We assume we are provided with a Black-Box FSM B, which is a deterministic, complete strongly-connected Mealy machine. We can test it by sending inputs and observing outputs. We cannot reset it. Our goal is to infer a model M equivalent to B except for the initial state.

3.1 Rationale

If we know a homing sequence $h \in I^*$ and a characterization set $W \subset I^*$ for B, then inferring it is easy: we repeatedly apply h to "reset" the machine to an identifiable state, and characterize it by applying sequences from W: each time we home into the same identifiable state, we can pick a sequence from W that has not yet been applied there, until we can characterize the state reached after this answer to h. Once we have characterized a state reached after applying h, say q, we can learn transitions from it: from q, we apply an input x, observe the output o so $\lambda(q,x) = o$, then an element $w \in W$. When we home into the same state again, we can reapply x and complete our knowledge of the tail state of the transition by applying another element $w' \in W$ until we can characterize $\delta(x,o) = q'$. Since B is strongly connected (and of course complete and deterministic), we can traverse all transitions of B by applying input sequences of the form $h\alpha xw$ where α is a transfer sequence to reach a state that is not directly reached at the end of a homing sequence. In that way, we are progressively building a *conjecture*, a partial machine that in the end will converge to an equivalent of B.

The key intuition of hW-inference is that we can pick some "tentative" homing sequence and characterization set. If they are not really homing or characterizing, states of the black-box machine will be confused, and we will start building a partial machine with merged ϕ mappings. This in turn will lead to non-determinism in the conjecture, and we show how this can be exploited to refine h and W, until they are precise enough to infer B.

3.2 hW-inference algorithm

Similar to the algorithm by Rivest and Schapire [13], we use the homing sequence h as an ersatz for a reset. However, instead of associating a copy of a table and algorithm with each possible answer, we simply try to associate a state with it, so in fact a mapping ϕ for W. The mapping associated with an output sequence r obtained as a response to the input sequence h will be denoted by $H(r)$.

In the following algorithm, each time an input is applied, we check for inconsistencies (see 3.3) of type 1 (Definition 3.3.1) and 2 (Definition 3.3.2) except at line 19 where we check only type 2.

When an inconsistency is found, h or W are refined (see 3.3). We abort current steps and restart from line 3.

Algorithm 1 Simplified hW-inference algorithm

```
1: procedure INFER
2:     initialize : $h \leftarrow \epsilon$; $W \leftarrow \{\epsilon\}$   ▷ (equivalent to $W \leftarrow \emptyset$ here)
3:     repeat
4:         $Q, \lambda, \delta \leftarrow \emptyset$
5:         repeat
6:             apply $h$ and observe $r \in O^*$
7:             if $H(r)$ is undefined then
8:                 $H(r) \leftarrow \emptyset$
9:             end if
10:            if $H(r)$ is undefined for some $w \in W$ then
11:                apply $w$, observe $y$, $H(r) \leftarrow H(r) \cup \{w \mapsto y\}$
12:            else
13:                let $q = H(r)$ be the state reached at end of $h$;
14:                find shortest input sequence $\alpha \in I^*$ leading from $q$ to a state $q' \in Q$ with incompletely known transition $(q', x)$;
15:                apply $\alpha.x.w$ observe $\beta.o.y$;
16:                $\lambda(q', x) = o$ and $\delta(q', x)(w) = y$
17:            end if
18:        until $M = (Q, I, O, \delta, \lambda)$ contains a strongly connected complete component
19:        ask for a counterexample.
20:        process counterexample as a Type-2 inconsistency
21:    until no counterexample can be found
22: end procedure
```

3.3 Refining h and W

3.3.1 *Type 1 inconsistency.*

The first type of inconsistency is when we have evidence that h is incorrect. It happens when we have observed a sequence $h/r.u/v$ and a sequence $h/r.u/v'$ in the global trace (from the start of the inference). If $v \neq v'$ we have an inconsistency and h can be extended to hu.

Indeed, if h was homing, after observing h/r, we should be in a unique state and because the black box is deterministic, we should always have the same answer to u after this state. Observing two distinct answers to u means that the starting states are different and thus, h is not homing. As already observed by Rivest and Schapire [13] hu will distinguish two tail states that were previously considered equal, and h can be extended at most $n-1$ times until it is really homing, where n is the number of states of B.

3.3.2 *Type 2 inconsistency.*

The second kind of inconsistency consists of a difference between the outputs provided by the driver and the outputs expected from the conjecture.

We had observed $h/r.t/s.u/v$, and we are observing $h/r'.t'/s'.u/v'$ such that $\delta(H(r), t) = \delta(H(r'), t')$ and $v \neq v'$. This implies that $\delta(H(r), t)$ and $\delta(H(r'), t')$ can be distinguished by u. Actually, all states traversed while applying u can be distinguished by some suffix of u. For technical reasons that we do not detail here, we apply the *Suffix1by1* approach presented in [8]. Namely, we add the shortest suffix of u that is not yet in W and at the same time we clean W of all its prefixes: if a sequence w' extends another sequence $w \in W$, then it is unnecessary to keep w since w' will distinguish at least the same states.

3.4 Getting counterexamples

In the machine learning context where query learning was expressed initially, a teacher (called oracle, although with a slightly different meaning from software testing) could be asked to check whether the conjecture was correct. This teacher provides a counterexample when the conjecture is nonequivalent to B. In black-box testing, no teacher is available. So when a conjecture is reached, some way of finding counterexamples must be found.

The easiest other way to approximate an oracle is to use a random walk on the graph of the conjecture, until we find a discrepancy between the conjecture and the black-box. This has to be bounded: at some point, if no discrepancy is observed, we will consider that the conjecture is equivalent to B.

Currently, we have experimented with a (bounded) random walk. The advantage of using the *Suffix1by1* counterexample processing method is that it is well adapted to long counterexamples as provided by random walks.

4 PRELIMINARY RESULTS

The hW-inference method was implemented in the *SIMPA* learning framework, and compared to the Rivest and Schapire [13] and to the ICTSS 2015 [5] algorithms. As usual with inference algorithms, we checked its efficiency and scalability by testing it on large sets of randomly generated machines. We generated 4628 strongly connected random finite state machines of increasing sizes (from 5 to 3000 states), with two input symbols and two output symbols (more inputs and outputs would provide better distinguishability, so easier inference: we experimented with a tough setting). For the hW-inference and Rivest and Schapire's inferences, random walks were used to find counterexamples, and the *Suffix1by1* [14] method to treat them.

At the end of the process, we can, in fact, check whether the model is equivalent to the black-box, since we generated this black-box and only pretend that we do not know it when learning it with the algorithms. Actually, in our experiments, we first check for existence of a counterexample before performing a random walk. In all cases, the hW-inference algorithm converged on the correct model, and it did so faster than previous algorithms.

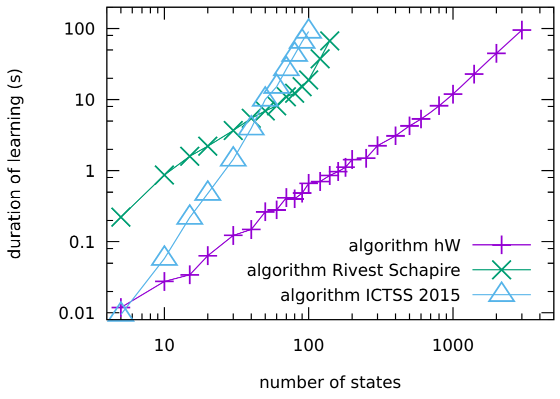

Figure 1: Duration of learning

Figure 1 shows the average time (in seconds) taken on a simple laptop (Intel® Core™ i7-4600U processor, 2.10 GHz, 16 Go RAM) by all three algorithms according to the number of states of the random FSM. We can see that we were not able to infer automata of more than 100 states with the ICTSS 2015 method, and 200 states with the Rivest and Schapire's method. However, automata up to 3000 states have been learnt with the hW-method, in less than two minutes on average.

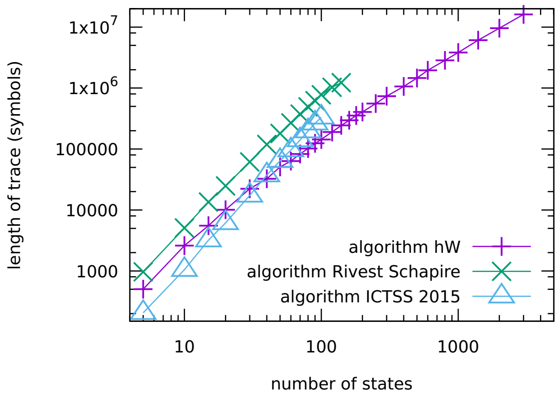

Figure 2: Inference trace length

Figure 2 shows the average length of the total input/output trace of the inference as a function of the number of states of the automata. The hW-algorithm clearly outperforms the other two algorithms as soon as the number of states is above 100. Linear regression on the data, taking into account the logarithmic scales, shows an experimental complexity of around $O(n^{1.99})$ for the Rivest and Schapire's algorithm, and around $O(n^{1.34})$ for the hW-method.

Our procedure for finding counterexamples, namely random walks, is naive. However, preliminary analyses show that the number of calls to this procedure remains low: for random machines, there are on average 5 calls to the oracle, regardless of the state number; the maximum number of calls to the oracle on our 4268 machines was 12. Consequently, the length of all counterexamples is also low compared to the length of the whole trace. This suggests that there would not be much to be gained with a smarter procedure for hW-inference .

5 CONCLUSION

In this position paper, we have tried to make the most of AI-inspired and machine learning methods to get behavioural models of black-box software systems without resetting them. Whereas traditional conformance testing, or even learning-based testing have tended to concentrate on sound and, if possible, complete methods, relying on proven and guaranteed W-sets or similar bases (as was still the case in Groz et al. [5]) hW-inference uses a heuristic approach that may seem unsound at the beginning, but that is fast and converges to the correct result, thanks to refinement of h and W. More importantly, it scales up to large automata. Therefore, it has the potential to address automated testing of real-size software components or systems without resorting to aggressive abstraction. Preliminary results are quite promising, and our next step will be to consolidate the method and apply it to various software systems.

REFERENCES

[1] G. Ammons, Rastislav Bodík, and J. R. Larus. 2002. Mining specifications. In *POPL 2002*. 4–16. https://doi.org/10.1145/503272.503275
[2] D. Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 2 (1987), 87–106.
[3] M. Büchler, K. Hossen, P. F. Mihancea, M. Minea, R. Groz, and C. Oriat. 2014. Model inference and security testing in the SPaCIoS project. In *CSMR-WCRE*.
[4] R. Groz, K. Li, A. Petrenko, and M. Shahbaz. 2008. Modular System Verification by Inference, Testing and Reachability Analysis. In *TestCom/FATES*. 216–233.
[5] R. Groz, A. Simao, A. Petrenko, and C. Oriat. 2015. Inferring finite state machines without reset using state identification sequences. In *ICTSS 2015*. Dubai.
[6] Y. Hsu, G. Shu, and D. Lee. 2008. A Model-based Approach to Security Flaw Detection of Network Protocol Implementations. In *International Conference on Network Protocols*. 114–123. https://doi.org/10.1109/ICNP.2008.4697030
[7] H. Hungar, T. Margaria, and B. Steffen. 2003. Test-Based Model Generation For Legacy Systems. In *ITC*. 971–980.
[8] M. N. Irfan, C. Oriat, and R. Groz. 2010. Angluin style finite state machine inference with non-optimal counterexamples. In *MIIT*. ACM, New York, NY, USA, 11–19.
[9] D. Lee and M. M Yannakakis. 1996. Principles and methods of testing finite state machines – a survey. *Proc. IEEE* 84, 8 (1996), 1090–1123.
[10] K. Meinke. 2010. CGE: A Sequential Learning Algorithm for Mealy Automata. In *ICGI*.
[11] D. Peled, M. Y. Vardi, and M. Yannakakis. 1999. Black Box Checking. In *Proceedings of FORTE'99*. Beijing, China.
[12] A. Petrenko, F. Avellaneda, R. Groz, and C. Oriat. 2017. From Passive to Active FSM Inference via Checking Sequence Construction. In *ICTSS 2017 (LNCS 10533)*. 126–141.
[13] R. L. Rivest and R. E. Schapire. 1993. Inference of Finite Automata Using Homing Sequences.. In *Machine Learning: From Theory to Applications*. 51–73.
[14] M. Shahbaz and R. Groz. 2009. Inferring Mealy Machines. In *FM (LNCS 5850)*. Eindhoven, The Netherlands, 207–222.
[15] M. P. Vasilievskii. 1973. Failure Diagnosis of Automata. *Cybernetics* 9 (1973), 653–665.