From Passive to Active FSM Inference via Checking Sequence Construction

Alexandre Petrenko[1], Florent Avellaneda[1], Roland Groz[2], and Catherine Oriat[2]

[1] CRIM, Canada
{Alexandre.Petrenko, Florent.Avellaneda}@crim.ca
[2] Univ. Grenoble Alpes, LIG, France
{Roland.Groz, Catherine.Oriat}@imag.fr

Abstract. The paper focuses on the problems of passive and active FSM inference as well as checking sequence generation. We consider the setting where an FSM cannot be reset so that its inference is constrained to a single trace either given a priori in passive inference scenario or to be constructed in active inference scenario or aiming at obtaining checking sequence for a given FSM. In each of the last two cases, the expected result is a trace representing a checking sequence for an inferred machine, if it was not given. We demonstrate that this can be achieved by a repetitive use of a procedure that infers an FSM from a given trace (identifying a minimal machine consistent with a trace) avoiding equivalent conjectures. We thus show that FSM inference and checking sequence construction can be seen as two sides of the same coin. Following an existing approach of constructing conjectures by SAT solving, we elaborate first such a procedure and then based on it the methods for obtaining checking sequence for a given FSM and inferring a machine from a black box. The novelty of our approach is that it does not use any state identification facilities. We only assume that we know initially the input set and a bound on the number of states of the machine. Experiments with a prototype implementation of the developed approach using as a backend an existing SAT solver indicate that it scales for FSMs with up to a dozen of states and requires relatively short sequences to identify the machine.

Keywords: FSM testing, machine inference, machine identification, active learning, checking experiments, checking sequences.

1 Introduction

Model-based testing from finite state models of systems, when it is only possible to interact with the system through its input/output interfaces, relies on traversing transitions of the model and being able to check that states reached after transitions in the system are consistent with those expected from the model. At the end of the test, the goal is to be able to guarantee that the system under test behaves as expected in the model. So the test must be built as a checking sequence of inputs that can uniquely identify (up to equivalence) a given model machine.

Computing a checking sequence from a finite state model dates back to the very early history of automata in computer science, starting with the work of Moore [15] and many approaches have been proposed to generate checking sequences for various types of models under various assumptions for the machine w.r.t determinism, completeness, and the existence of specific sequences for a machine such as distinguishing sequences [10], signatures [21], state identifiers [17] etc.

More recently, at the turn of the century, model-based approaches have led to an interest in inference techniques. Instead of checking whether a system behaves as specified by a model, it works the other way round: we try to build a model, called a conjecture that will predict as accurately as possible the behavior of a system. This can be based on a corpus of given observed behaviors of the system (passive inference), or on the ability to submit test sequences (active inference). One key driver for such approaches is that experience in industrial context have shown that building and maintaining accurate and up-to-date models was complicated, and needed specific expertise. Being able to derive models automatically relieves the burden of creating and maintaining them.

Building a checking sequence can be seen as a top-down approach (from model to implementation) and inference as bottom-up approach (from implementation to conjectured model). The two are in fact closely linked: in active inference, if a sequence is built that uniquely identifies a machine, then this sequence is a checking sequence for this machine. The main difference is in the starting point: for checking sequence generation, we assume we know the (specification) machine to be identified. For inference, the machine is unknown.

In this paper, we propose an iterative approach that alternates passive inference with construction of checking experiments. Initially, an input sequence will be too short to uniquely identify a machine. But one can exhibit one of many possible conjectures that would match the observed input/output sequence (the running trace). So the idea is to build a checking experiment that will distinguish among conjectures, and which is appended to the current trace. Following this idea, the set of potential conjectures is reduced, and the process is iterated until we get to a point where the set is reduced to a singleton, at which point the input projection of the observed trace is a checking sequence.

Interestingly, this theoretical framework had already been envisioned by J. Kella, in one of the early papers on passive inference [14]. Let us quote the end of his introduction (our comments in brackets): “When the machine has no distinguishing sequences the reducing technique can help in minimizing the length of the checking experiment by iterative construction of the experiment. An initial sequence is applied to the machine and the resulting input-output sequence is reduced [by state merging]; the result will indicate a family of machines responding in the same way. An additional sequence which eliminates nondesired machines is then applied and another reduction is performed; by repeated application of the basic iteration the sequence will reduce uniquely to the checked machine [up to the initial state]. This method of checking experiment construction was tried for some examples but there is no proof yet to whether it is more efficient than other methods [10] and whether it will converge in all cases.”

Our approach shows that is indeed possible to uniquely identify a non-resettable deterministic complete machine, while building a checking sequence for it, with no a priori knowledge apart from a bound on the number of states, and the input set of the machine. Contrary to previous work [9], it does not require a characterization set or another assumption on sequences to distinguish states in the machine.

Section 2 will provide precise definitions for our formal framework, while Section 3 will define the inference problems and checking sequence generation in our context, i.e. from a single trace for a non-resettable machine, in relation with the state of the art. Section 4 shows how passive inference, i.e. the computation of a conjecture from a single trace can be encoded into a Boolean formula, so that a SAT solver can be used to efficiently get a conjecture. Sections 5 and 6 present our iterative approaches, showing the two sides of the coin: checking sequence generation and active inference. Section 7 presents experiments that show that the algorithms can work with middle-sized automata. Section 8 concludes.

2 Definitions

A *Finite State Machine* (FSM) M is a 5-tuple (S, s_0, I, O, T), where S is a finite set of states with an initial state s_0; I and O are finite non-empty disjoint sets of inputs and outputs, respectively; T is a transition relation $T \subseteq S \times I \times O \times S$, $(s, a, o, s') \in T$ is a transition. When we need to refer to the machine M in a state $s \in S$, we write M/s.

M is *completely specified* (complete) if for each tuple $(s, a) \in S \times I$ there exists transition $(s, x, o, s') \in T$. It is *deterministic* if for each $(s, a) \in S \times I$ there exists at most one transition $(s, a, o, s') \in T$, otherwise it is *nondeterministic*. We consider in this paper only deterministic FSMs.

An *execution* of M/s is a sequence of transitions forming a path from s in the state transition diagram of M. The machine M is *initially* connected, if for any state $s \in S$ there exists an execution from s_0 to s. M is *strongly* connected, if the state transition diagram of M is a strongly connected graph.

A *trace* of M/s is a string of input-output pairs which label an execution from s. Let $Tr(s)$ denote the set of all traces of M/s and Tr_M denote the set of traces of M/s_0. For trace $\omega \in Tr(s)$, we use s-after-ω to denote the state M reached after the execution of ω, for an empty trace ε, s-after-$\varepsilon = s$. When s is the initial state then we write M-after-ω instead of M/s_0-after-ω.

Let also $out(s, \alpha)$ be an output sequence produced by the input sequence $\alpha \in I^*$ in M/s. For input sequence α applied in state s, we let $tr_s(\alpha)$ denote the trace with the input projection α; we will omit the subscript when no confusion occurs.

Given an input sequence α, states $s, s' \in S$ are *equivalent w.r.t.* α, if $out(s, \alpha) = out(s', \alpha)$, denoted $s \cong_\alpha s'$, they are *distinguishable* by α, if $out(s, \alpha) \neq out(s', \alpha)$, denoted $s \ncong_\alpha s'$ or simply $s \ncong s'$. A *distinguishing* sequence of M is an input sequence α for which the output sequence produced by M in response to α identifies the state of M: for all $s, s' \in S$, $out(s, \alpha) \neq out(s', \alpha)$. A *characterization* set of M is a set input sequences such that every $s, s' \in S$, there exists a sequence α in the set such that $out(s, \alpha) \neq out(s',$

α). States s and s' are equivalent if they are equivalent w.r.t. all input sequences, thus $Tr(s) = Tr(s')$, denoted $s \cong s'$. The equivalence and distinguishability relations between FSMs is similarly defined. Two FSMs are equivalent if their initial states are equivalent. A complete FSM is *minimal*, if it has no equivalent states.

Given two FSMs $M = (S, s_0, I, O, T)$ and $M' = (S', s'_0, I, O, T')$, their *product* $M \times M'$ is an FSM (P, p_0, I, O, H), where $p_0 = (s_0, s'_0)$ is such that P and H are the smallest sets satisfying the following rule: If $(s, s') \in P$, $(s, x, o, t) \in T$, $(s', x, o', t') \in T'$, and $o = o'$, then $(t, t') \in P$ and $((s, s'), x, o, (t, t') \in H$.

Lemma. *If M and M' are complete machines then they are equivalent iff the product $M \times M'$ is complete in the input set I.*

Two complete FSMs $M = (S, s_0, I, O, T)$ and $M' = (S', s'_0, I, O, T')$ are called *isomorphic* if there exists a bijection f: $S \rightarrow S'$ such that $f(s_0) = s'_0$ and for all $a \in I$, $o \in O$, and $s \in S$, $f(s\text{-after-}ao) = f(s)\text{-after-}ao$. Isomorphic FSMs are equivalent, but the converse does not necessarily hold. Note that we do not require equivalent machines to be minimal.

Given a trace $\omega \in (IO)^*$ of length $|\omega|$, let $Pref(\omega)$ be the set of all prefixes of ω. We define a linear FSM $W = (X, x_0, I, D_\omega)$, such that $|X| = |\omega| + 1$, and there exists a bijection f: $X \rightarrow Pref(\omega)$, such that $f(x_0) = \varepsilon$, $(x_i, ao, x_{i+1}) \in D_\omega$ iff $f(x_i)ao = f(x_{i+1})$ for all $i = 0, \ldots, |\omega| - 1$, in other words, $Tr_W = Pref(\omega)$. We call it an ω-*machine* W.

While the set of traces of the ω-machine is $Pref(\omega)$, there are many FSMs which have the trace ω among other traces. We restrict our attention to the class of FSMs with at most n states and alphabets I and O, denoted $\Im(n, I, O)$. An FSM $C = (S, s_0, I, O, T)$, $C \in \Im(n, I, O)$ is called an ω-*conjecture*, if $\omega \in Tr_C$. Let $\Im_\omega(n, I, O)$ be the set of all ω-conjectures in the set $\Im(n, I, O)$. Clearly, the ω-machine is also an ω-conjecture, if $|\omega| < n$.

The states of the ω-machine $W = (X, x_0, I, D_\omega)$ and an ω-conjecture $C = (S, s_0, I, O, T)$, $C \in \Im_\omega(n, I, O)$ are closely related to each other. A state of the ω-machine reached after any prefix of the trace ω corresponds to a unique state of the ω-conjecture that is reached after that prefix. Formally, there exists a mapping μ: $X \rightarrow S$, such that $\mu(x) = s_0\text{-after-}f(x)$, the state reached by C when the trace $f(x) \in Pref(\omega)$ is executed. The mapping μ induces a partition π_C on the set X such that x and x' belong to the same block of the partition π_C, denoted $x =_{\pi_C} x'$, iff $\mu(x) = \mu(x')$.

Given an ω-conjecture C with the partition π_C, let D be an ω'-conjecture with the partition π_D, such that $\omega' \in Pref(\omega)$, we say that the partition π_C is an *expansion* of the partition π_D, if its projection to ω' coincides with the partition π_D; viz $\pi_D = \{P \cap X' \mid P \in \pi_C\}$ where $X' = \{x_i \in X \mid i \leq |\omega'|\}$.

An input sequence $\alpha \in I^*$ is a *checking sequence* for a complete minimal FSM M with n states M if for each FSM $N \in \Im(n, I, O)$, there is a state s in M such that $N \cong_\alpha M/s$, it holds that $N \cong M/s$. The trace $tr_s(\alpha)$, where α is a checking sequence, is called a *checking trace* of M. In this definition, we allow uncertainty in the initial state of M since it may have other states which converge with the initial state on a checking trace (an example of such an FSM can be found in Section 5).

Checking sequence is a special type of checking experiments for FSM, it is usually considered for FSM based testing when a reset operation in FSM implementations is unavailable or formidably costly to execute.

3 Problem Statement and Related Work

We consider the following closely related problems, passive and active FSM inference as well as checking sequence construction. Significantly, we restrict our setting to the case where a FSM may not be reset, so that the definitions we give here refer to a single trace. Actually, if a FSM can be reliably reset, the reset sequences can be included in the trace, so the definitions below can cover the general case. We state the problems using the definitions given above.

Passive inference is a classical problem whereby given a trace ω we need to build an ω-conjecture with a minimal number of states [14][6][2].

Active inference, aka active automata learning, is another problem addressed in the literature [5]. Restated in our FSM context, given a black box, which behaves as an unknown minimal complete strongly connected FSM with the input alphabet *I* and the number of states equal to *n*, infer the FSM, i.e. build an ω-conjecture equivalent to the FSM and its checking trace ω.

The checking sequence problem differs from active inference in assuming that the expected behavior of a black box with at most *n* states submitted for testing is given as a strongly connected FSM *M* called a specification machine and we need to determine its checking trace ω. The relation to passive inference is direct, once ω is constructed, any ω-conjecture must be equivalent to *M*.

In this section, we briefly discuss the existing approaches addressing these problems which do not rely on the existence of a reset operation.

3.1 Passive Inference from a Single Trace

Passive FSM inference problem is stated by Kella in 1971 [14] as sequence machine identification and later as system/automaton identification problem by Gold [6]. The problem has been studied ever since. The problem is known to be computationally very hard, nevertheless, numerous proposals have been made, mainly on developing state merging techniques to transform an ω-machine into an ω-conjecture as small as possible, see, e.g., [14] [27] etc. The most recent approaches are based on satisfiability (SAT) solvers [1][11].

In Section 4, we propose an approach to build an ω-conjecture within a bound on the number of states using a SAT solver that avoids obtaining conjectures which were already considered.

3.2 Checking Sequence Problem

The problem of checking sequence generation from an FSM has a long history starting from work of Moore [15] and Hennie [10]. Almost all existing methods require a machine be complete and minimal. Moreover, the vast majority of the proposed methods only apply to FSMs which have distinguishing sequences or distinguishing sets, see, e.g. [4][7][12][22][23][26]. Not all FSMs possess these sequences and their construction is a non-trivial problem. Only few methods can generate checking sequence from complete and minimal FSM which has just characterization set and no other distinguishing sets, see [10][19][18]. Moreover, they cannot be called efficient, since the size of a checking sequence generated by using characterization set grows exponentially with the length and number of sequences in characterization set [25].

The problem of checking sequence generation without even determining distinguishing sequences or finding an "optimal" or any other characterization set remains open, to the best our knowledge. In Section 5 we propose an approach that does not assume any distinguishing or characterization set computation.

3.3 Active Inference without Reset

Active inference has most often been addressed in the context of learning from samples and queries [5][8], so that the problem of dealing with a single trace has not received a lot of attention. An early attempt was proposed by [20], as an adaptation to Angluin's L* algorithm. It assumes that an external oracle can be queried to provide a counterexample (hence an input sequence to distinguish the black box and the conjecture), and starts with the knowledge of a homing sequence. More recently an approach was proposed that does not require an external oracle, but still assumes knowledge of a characterization set [9].

However, the assumptions about the existence of an external oracle, knowledge of homing or state characterizing sequences, such as distinguishing sequences and characterization sets, are not easy to justify in practice, therefore the problem of active inference of FSMs with neither reset operation nor strong assumptions about a given back box remains open. In Section 6 we propose an approach that does not require such assumptions.

4 Passive Inference with SAT solving

Since an ω-machine is itself an ω-conjecture, the minimization problem boils down to merging states of the ω-machine without introducing traces that would contradict the trace ω. Therefore by encoding a trace into a Boolean formula, and expressing state merging possibilities in that formula, we may use a SAT solver to determine acceptable mergers.

4.1 Problem encoding

Here we present a procedure for encoding a trace into a Boolean formula, and at the same time express a constraint on the number of states.

Let $W = (X, x_0, I, D_\omega)$ be an ω-machine. To find an ω-conjecture with at most n states amounts to determine a partition π on the set of states X such that the number of blocks does not exceed n. This problem can be cast as a constraint satisfaction problem (CSP) [3]. Let X be $\{x_0, \dots, x_{|\omega|}\}$, so each integer variable represents a state of the ω-machine. Since the ω-machine is deterministic, the state variables satisfy the following constraint:

$$\forall x_i, x_j \in X:$$

$$\text{if } x_i \not\cong x_j \text{ then } x_i \neq x_j \text{ and}$$

$$\text{if } \exists a \in I \text{ s.t. } out(x_i, a) = out(x_j, a) = o \text{ then } x_i = x_j \Rightarrow x_i\text{-after-}ao = x_j\text{-after-}ao \quad (1)$$

If the number of states in an ω-conjecture to be constructed should be at most n then each state variable $x_i \in \{0, \dots, n - 1\}$. Then an assignment of values to variables in $\{x_0, \dots, x_{|\omega|}\}$ such that the formula (1) is satisfied defines a mapping $\mu: X \rightarrow S$, where S is the set of states of an ω-conjecture, i.e., the mapping μ defines a partition of X into n blocks.

These formulas can be translated to SAT using unary coding for each integer variable $x \in X$, such that x is represented by n Boolean variables $v_{x,0}, \dots, v_{x,n\text{-}1}$. For each $x \in X$, we have the clause:

$$v_{x,0} \vee \dots \vee v_{x,n\text{-}1} \qquad (2)$$

These clauses mean that each state of the ω-machine W should be in at least one block.

For each state $x \in X$ and all $i, j \in \{0, \dots, n - 1\}$ such that $i \neq j$, we have the clauses:

$$\neg\, v_{x,i} \vee \neg\, v_{x,j} \quad (3)$$

The clauses mean that each state of the ω-machine W should be in at most one block.

Since a sought-after ω-conjecture must be deterministic, the formula (1) is encoded into the following clauses. First, distinguishable states of W should be in different blocks, so for every $x, y \in X$ such that $x \not\cong y$ and all $i \in \{0, \dots, n - 1\}$

$$\neg v_{x,i} \vee \neg v_{y,i} \quad (4)$$

Second, states of W equivalent w.r.t. to some input if placed in the same block must have their successors also in one block. Hence for all $x_i, x_j \in X$ such that $out(x_i, a) = out(x_j, a) = o$ and all $i, j \in \{0, \dots, n - 1\}$ we have a formula which can directly be translated into clauses

$$(v_{x,i} \wedge v_{x',i}) \Rightarrow (v_{(x\text{-after-}ao),i} \Rightarrow v_{(x'\text{-after-}ao),i}) \ (5)$$

To simplify learning that $x = y$ for some $x, y \in X$ we further rewrite the clauses (4) and (5) using auxiliary variables $e_{x,y}$ modeling the fact that $x = y$. For every $x, y \in X$ such that $x \not\cong y$ we have

$$\neg e_{x,y} \quad (6)$$

For all $x, y \in X$ such that $out(x, a) = out(x, a) = o$, we have

$$e_{x,y} \Rightarrow e_{x\text{-after-}ao,y\text{-after-}ao} \qquad (7)$$

The relation between auxiliary state variables is expressed in the following clauses. For every $x, y \in X$ and all $i \in \{0, \dots, n - 1\}$

$$e_{x,y} \wedge v_{x,i} \Rightarrow v_{y,i} \qquad (8)$$

$$\neg e_{x,y} \wedge v_{x,i} \Rightarrow \neg v_{y,i} \qquad (9)$$

The resulting Boolean formula is the conjunction of clauses (2), (3), (6), (7), (8) and (9). To check its satisfiability one can use any of the existing solvers. If a solution exists then we have an ω-conjecture with *n* or fewer states. In the context of passive inference, we are usually interested in finding an ω-conjecture as small as possible. This requires several trials with varying values of *n*.

4.2 Passive inference of different (new) conjectures

In the context of active inference as well as checking sequence construction we aim at obtaining a single ω-conjecture while avoiding constructing isomorphic conjectures. A key building block will be provided by the following procedure to infer a conjecture that differs from already considered conjectures. We identify isomorphic conjectures by their common partition, hence we add as a constraint that we look for an ω-conjecture that does not expand a set of "forbidden" partitions. If such ω-conjectures are found they will be used in Sections 5 and 6 to augment the trace ω by adding suffixes that eliminate distinguishable conjectures until only one remains.

Algorithm 1. *Infer_conjecture*(ω, *n*, Π)
Input: A trace ω, an integer *n*, and a set of partitions Π
Output: An ω-conjecture with at most *n* states such that its partition does not expand any partition in Π, or False.

```
formula = conjunction of the clauses (2), (3), (6), (7), (8) and (9)
for all $\pi \in \Pi$ do
        clause = False
        for all $x$, $y$ such that $x =_\pi y$ do
        clause = clause $\vee \neg e_{x,y}$
        end for
        formula = formula $\wedge$ clause
end for
return call-solver(formula)
```

5 Checking Sequence Construction

The idea of the proposed method for checking sequence (trace) generation is to find an FSM that reacts as the given specification FSM to a current input sequence using passive inference and eliminate it by extending the sequence with a suffix distinguishing the two machines or forbidding the passive inference from further regeneration if they cannot be distinguished any further. This process iterates until no more conjectures distinguishable from the given FSM can be found. The procedure is implemented in Algorithm 2.

Algorithm 2. Generating checking trace
Input: A complete strongly-connected FSM *M* with *n* states
Output: A checking trace ω

```
$\omega := \varepsilon$
```

```
$\Pi := \varnothing$
while an $\omega$-conjecture $C$ is returned by Infer_conjecture($\omega$, $n$, $\Pi$) do
    if $C$-after-$\omega \times M$-after-$\omega$ is complete then
        $\Pi := \Pi \cup \{\pi_C\}$
    else
        Determine an input sequence $\beta a$ such that $\beta$ is a shortest transfer sequence from the state $C$-after-$\omega$ to a state with the undefined input $a$ in $C$-after-$\omega \times M$-after-$\omega$
        $\omega := \omega tr(\beta a)$, where $tr(\beta a)$ is the trace of $M$-after-$\omega$
    end if
end while
return $\omega$
```

Algorithm 2 calls *Infer_conjecture*(ω, *n*, Π), which in turn calls a SAT solver constraining it to avoid solutions of already considered conjectures.

Note that the Boolean formula used by the SAT solver can be built incrementally by saving a current formula and adding only new clauses each time a trace ω or a set of partitions Π is augmented.

Theorem 1. Given an FSM *M* with *n* states, Algorithm 2 returns a checking trace ω.

Sketch of the proof. When Algorithm 2 terminates the resulting trace ω is indeed a checking one, since by the post-condition of *Infer_conjecture* no conjecture exists that is distinguishable from the given FSM *M*, after having executed ω. Note that all complete conjectures equivalent to *M* are excluded because as soon as one is found (including possibly *M* itself), according to the Lemma, its partition is added to Π. Algorithm 2 always terminates, because the number of all possible conjectures with the fixed input alphabet within a given bound on the number of states *n* is finite.

Example. Consider the FSM in Fig.1, it has no distinguishing sequence, its characterization set is {*a*, *b*}.

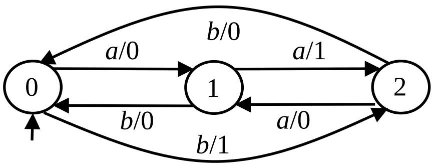
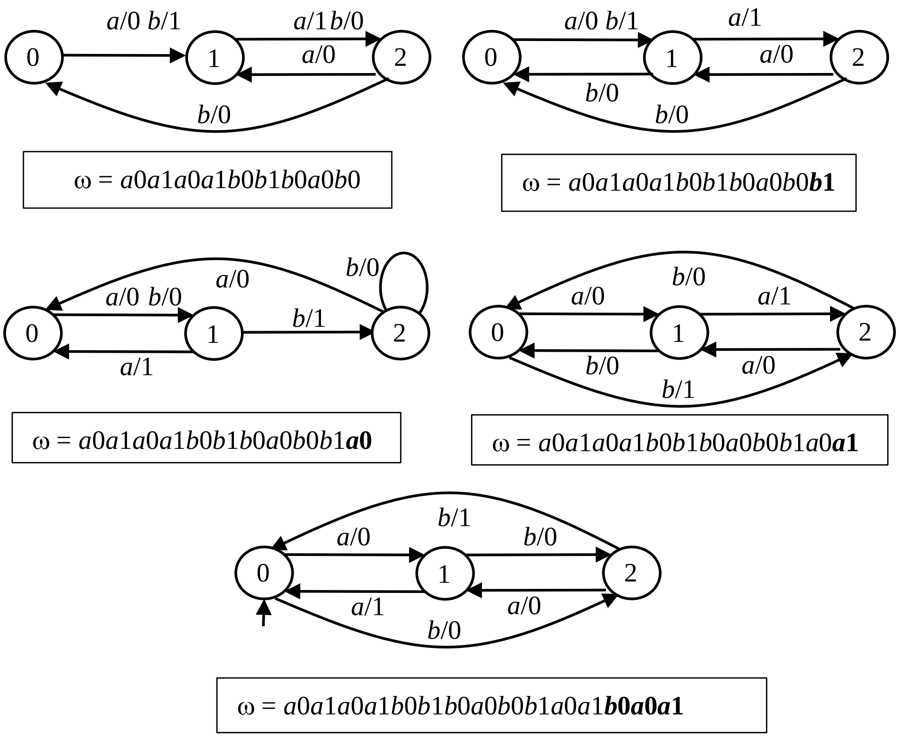


Fig.1. The FSM *M*

This example is used in [18], where a method for checking sequence generation from a minimal FSM without distinguishing sequence is proposed. Using this example the authors of [18] compare their method with those of [10][19] and report that the length of checking sequence obtained by their method is 120, while that of [10] is 171 and 248 of [19].

Algorithm 1 implemented in a prototype tool presented in Section 6 returns the checking sequence $\omega = a0a1a0a1b0b1b0a0b0b1a0a1b0a0a1$ of length 15. Fig. 2 shows intermediate complete ω-conjectures obtained executing Algorithm 1. Notice that the last but one conjecture is actually the FSM *M*, though, the same trace is also accepted by another conjecture, which is eliminated using the suffix *b*0*a*0*a*1.



$\omega = a0a1a0a1b0b1b0a0b0$

$\omega = a0a1a0a1b0b1b0a0b0\boldsymbol{b1}$

$\omega = a0a1a0a1b0b1b0a0b0b1\boldsymbol{a0}$

$\omega = a0a1a0a1b0b1b0a0b0b1a0\boldsymbol{a1}$

$\omega = a0a1a0a1b0b1b0a0b0b1a0a1\boldsymbol{b0a0a1}$

Fig. 2. The ω-conjectures generated by Algorithm 1 along with their versions of ω; suffixes in bold show how ω grows.

Note that the algorithm does not require the FSM to be minimal, moreover, it can be adapted to accept even a partial FSM. We are not aware of any method for checking sequence construction for FSMs which are partially defined and have compatible states, i.e., machines without characterization set. The only existing method which deals with such machines is [17], but it relies on the usage of the reset operation, as opposed to the approach proposed here.

6 Active Inference Approach

The iterative approach of Algorithm 2, which relied on computing a checking experiment for an ω-conjecture that was consistent with the current prefix trace can be adapted to active inference. The trick is to find a checking experiment not between the reference FSM *M* and the ω-conjecture, but between two possible ω-conjectures and retain the one that is consistent with the observations on the black box.

Given a black box BB, which behaves as an unknown minimal complete strongly connected FSM with the input alphabet *I* and a number of states equal to *n*, Algorithm 3 infers the FSM and constructs its checking trace.

Algorithm 3. Inferring BB and determining its checking trace

Input A black box BB, input set *I* and integer *n*

Output A minimal complete ω-conjecture with *n* states and a checking trace ω

```
$\omega := \varepsilon$
$\Pi := \varnothing$
$C := Infer\_conjecture(\omega, n, \Pi)$
while an ω-conjecture $D$ is returned by $Infer\_conjecture(\omega, n, \Pi)$ do
    if $D/D$-after-$\omega \times C/C$-after-$\omega$ is complete in the input set $I$ then
        $\Pi := \Pi \cup \{\pi_D\}$
    else
        Determine an input sequence $\beta a$ such that $\beta$ is a shortest transfer sequence from
        the state $C$-after-$\omega$ to a state with the undefined input $a$ in $D/D$-after-$\omega \times C/C$-
        after-$\omega$
        $\omega := \omega tr(\beta a)$, where $tr(\beta a)$ is the trace obtained by applying $\beta a$ to BB
            if $\omega \notin Tr_C$ then
                $C := Infer\_conjecture(\omega, n, \Pi)$
            end if
    end if
end while
return $C$ and $\omega$
```

Example. Consider that the FSM *M* in Fig.1 is a BB. Six intermediate complete ω-conjectures shown in Fig. 3 are obtained executing Algorithm 3. The last two conjectures both accept ω = *a*0*a*1*a*0*b*0*b*1*b*0*b*1*a*0*a*1*b*0*a*0*a*1*a*0*a*1. Both end up in state 2 from which they cannot be distinguished. The algorithm returns ω as a checking trace and the last but one conjecture which is isomorphic to FSM *M*/2. The last conjecture is isomorphic to FSM *M*/0. Indeed, this trace is accepted by *M* in two states, 0 and 2.

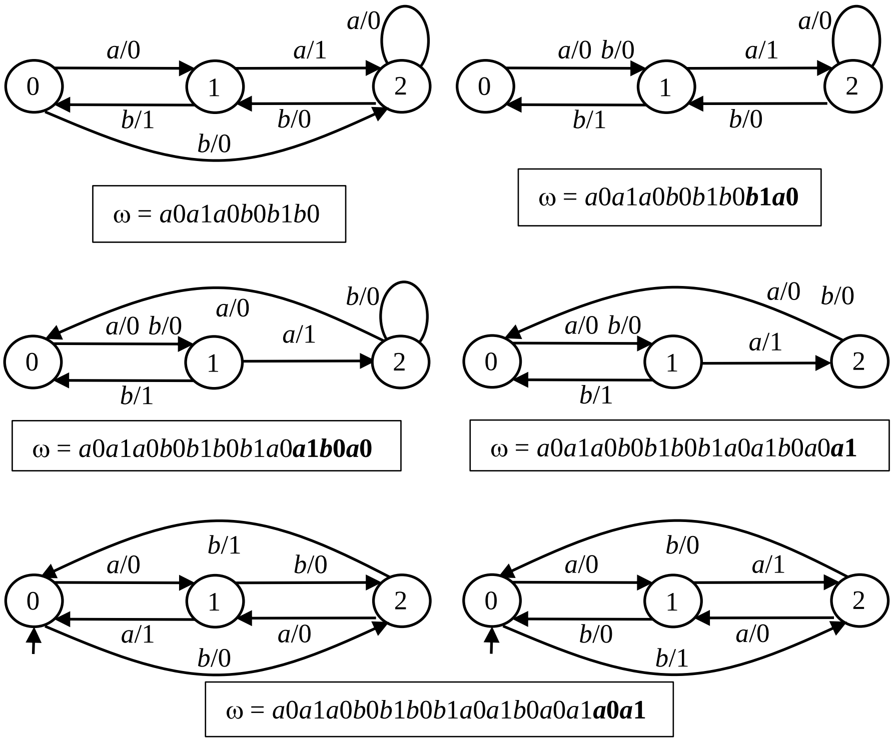


Fig. 3. The ω-conjectures generated by Algorithm 3 along with their versions of ω; suffixes in bold show how ω grows.

Theorem 2. If a black box behaves as a minimal complete strongly connected FSM with the input alphabet *I* and the number of states equal to *n*, Algorithm 3 infers it and constructs a checking sequence and trace for it.

Sketch of the proof. Algorithm 3 follows the steps of Algorithm 2, just replacing the FSM *M* by a current conjecture. This does not influence its termination since it only occurs when no more distinguishable conjecture can be found. And at some point, because the black box behaves as an FSM with *n* states, it will be returned by *Infer_conjecture*, so that the remaining conjecture is equivalent to the FSM of the back box initialized in some state. The resulting trace accepted by that state is a checking one, as in Theorem 1.

The expected complexity of the proposed approach could be estimated by viewing it as a mutation-based technique which kills mutants. In our approach at each iteration only a mutant surviving a current trace can be generated and then killed, drastically reducing the complexity of mutation-based techniques. A naive worst-case estimation based on number of (potential) mutants would be thus grossly overestimated. This explains why we provide in the next section experimental results on the observed complexity with random machines.

7 Experiments

The prototype was developed on C++ depending only on a SAT Solver Cryptominisat [24], as a backend. All the experiments were performed on a virtual machine (VirtualBox) with 8 GiB of RAM and one CPU used. The computer has the processor 7-3770 and 16 GiB of RAM.

Table 1 presents experimental results on randomly generated FSMs. The numbers of inputs as well as outputs are fixed to two, while the number of state is varied. For each state number, 101 complete strongly connected machines are generated; they are not necessarily minimal, since the approach does not require any state distinguishability. Each generated machine playing role of a specification FSM is used to construct checking sequence, and acting as a black box system is used for inference. Median values are collected for the length of resulting checking sequences |ω|, the number of times the solver is called (#solver), and execution time in seconds.

n	**RANDOM FSMs**					
	Checking			**Inferring**		
	|ω|	#solver	time	|ω|	#solver	time
1	2	3	0.01	2	3	0.01
2	7	8	0.01	7	9	0.01
3	17	14	0.01	18	18	0.01
4	26	20	0.01	30	24	0.01
5	37	26	0.01	43	32	0.02
6	47	32	0.03	57	39	0.05
7	63	39	0.07	69	45	0.13
8	76	44	0.17	83	54	0.32
9	100	53	0.59	107	72	2.4
10	118	58	1.7	119	70	9.0
11	146	69	18.5	146	83	161

Table 1. Experimental results with randomly generated FSMs with two inputs and outputs.

The prototype scales for up to a dozen of states. This matches the state-of-the-art, we quote “For the set of problems studied, the most efficient algorithms described in this paper find the exact solution in very little time in all problems that have solutions up to 11–12 states, and become progressively less effective as the number of states increases.” [16].

To assess the performance of the prototype to various numbers of inputs and outputs, another series of experiments reported in Table 2 were performed for machines with five states. Our experiments by varying their numbers separately show, unsurprisingly, that increasing number of inputs or outputs have opposite effects on the effectiveness, the more inputs the more complex the solutions (the search space is larger) but the more outputs the easier the solutions (more outputs increase distinguishability).

In addition, we performed another series of experiments using randomly generated lock machines (Table 3). A lock FSM (aka Moore lock) with *n* states has a unique "unlocking" input sequence of length *n* which executes the "remotest" transition, the transitions not covered by this sequence all lead to the initial state resetting the lock. We consider lock machines as ultimate test for active inference and checking sequence generation methods. As before for each number of states we generate 101 random locks with two inputs and two outputs and collect the same parameters as above. Clearly, for a fixed number of states, locks differ only in labelling of unlocking sequences, which effects the performance of the prototype, since it chooses inputs completing and distinguishing conjectures following the lexicographical order.

#inputs = #outputs	**RANDOM FSMs**					
	Checking			**Inferring**		
	|ω|	#solver	time	|ω|	#solver	time
2	37	26	0.01	43	32	0.02
3	51	40	0.03	57	47	0.05
4	68	53	0.04	70	58	0.09
5	74	62	0.05	81	71	0.12
6	88	73	0.07	95	85	0.2
7	101	83	0.09	109	99	0.3
8	113	95	0.12	121	111	0.38
9	121	102	0.12	127	122	0.5
10	138	114	0.18	145	136	0.72
20	257	212	0.63	276	261	3.2
30	377	312	1.3	425	391	10
40	525	412	2.5	517	571	22

Table 2. Experimental results with randomly generated arbitrary FSMs with five states.

n	**RANDOM LOCKs**					
	Checking			**Inferring**		
	|ω|	#solver	time	|ω|	#solver	time
1	2	3	0.01	2	3	0.01
2	7	8	0.01	7	7	0.01
3	22	16	0.01	23	22	0.01
4	57	28	0.04	58	61	0.05
5	110	40	0.41	127	164	0.79
6	255	58	7.8	269	514	21
7	488	456	870	456	2202	970

Table 3. Experimental results with randomly generated lock FSMs.

It is interesting to notice that active inference and checking sequence construction have comparable lengths of the resulting input sequences. After all, in both cases a checking sequence for the same machine is generated.

We observe that the length of resulting sequences grows polynomially, the number of times the solver is called linearly and time exponentially with the number of states.

8 Conclusions

We have presented a method that can infer a model of a non-resettable black box FSM for which we only know an upper bound *n* on the number of states. It produces the model along with the input sequence that was used for inferring it. The algorithm terminates on a final model that is equivalent to the black-box FSM up to initialization, and since it identifies a unique machine such that the input sequence is a checking sequence for this FSM.

The main benefit of this approach is that it only requires a bound on the number of states, no other assumption is needed, and the system does not have to be reset. This implies that it may have a wide spectrum of applications. The performance of active inference methods is usually assessed through the number of interactions with a system that are needed to infer it. Experiments have shown that the length of the input sequence implied by our approach is quite good. Another issue comes from the internal computations needed by the inference algorithm to build the model of the system. The method relies on a SAT solver to propose conjecture FSMs that are consistent with an observed trace. Unfortunately, this induces an exponential growth in the number of states, and this has been the limiting factor in our experiments. However, being able to infer state machines of up to a dozen states is in itself interesting for a large range of applications (many systems have relatively small state-space for the control part of their computations).

The approach seems promising, and can be improved in several directions. First, we have encoded the constraints for passive inference in a straightforward way, which puts a high burden on the constraint solver. It should be possible to encode the problem with more elements from the trace and FSM-structure to help the solver, with some guidance. Another direction we are investigating is to extract more information from previous conjectures and observations so as to reduce the number of calls to the solver to a minimum. In many cases, the calls to the solver can be avoided because it is possible to derive further checking experiments from the structure of the past conjecture(s). Instead of calling the solver to identify a new conjecture, it could be possible to refine the current conjecture.

References

1. Abel, A., Reineke, J. MeMin SAT-based exact minimization of incompletely specified mealy machines. In IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 94-101 (2015)
2. Biermann, A. W., Feldman, J. A. On the synthesis of finite-state machines from samples of their behavior. IEEE transactions on Computers, 100(6), 592-597 (1972)
3. Carbonnel, C., Cooper, M.C. Tractability in constraint satisfaction problems: a survey. Constraints, 21(2), 115-144 (2016)
4. Boute, R.T. Distinguishing sets for optimal state identification in checking experiments. IEEE Transactions on Computers, 23, 874–877 (1974)
5. De la Higuera, C. Grammatical inference: learning automata and grammars. Cambridge University Press (2010)

6. Gold, E. M. Complexity of automaton identification from given data. Information and control, 37(3), 302-320 (1978)
7. Gonenc, G. A method for the design of fault detection experiments. IEEE Transactions on Computers 19, 551–558 (1970)
8. Groz, R., Li, K., Petrenko, A., Shahbaz, M. Modular system verification by inference, testing and reachability analysis. In Testing of Software and Communicating Systems, Springer Berlin Heidelberg, 216-233 (2008)
9. Groz, R., Simao, A., Petrenko, A. and Oriat, C. Inferring finite state machines without reset using state identification sequences. In IFIP International Conference on Testing Software and Systems. Springer Berlin Heidelberg, 161-177 (2015)
10. Hennie, F.C.: Fault-detecting experiments for sequential circuits. Proc. Fifth Annual Symp. On Circuit Theory and Logical Design, pp. 95–110 (1965)
11. Heule, M.J., Verwer, S. Exact DFA identification using SAT solvers. In International Colloquium on Grammatical Inference. Springer Berlin Heidelberg, 66-79 (2010)
12. Hierons, R.M., Ural, H. Optimizing the length of checking sequences. IEEE Transactions on Computers, 55, 618–629 (2006)
13. C. de la Higuera, Grammatical Inference: Learning Automata and Grammars, Cambridge University Press, 2010.
14. Kella, J. Sequential machine identification. IEEE Transactions on Computers, 100(3), 332-338 (1971)
15. Moore, E.F. Gedanken experiments on sequential machines. In: Automata Studies. Annals of Mathematical Studies, vol. 34, pp. 129–153 (1956)
16. Oliveira, A. L., João PM Silva. Efficient algorithms for the inference of minimum size dfas. Machine Learning 44, no.1, 93-119 (2001)
17. Petrenko A., Yevtushenko. N. Testing from partial deterministic FSM specifications, IEEE Transactions on Computers, 54(9), 1154-1165 (2005)
18. Porto, F.R., Endo, A.T., Simao, A. Generation of checking sequences using identification sets. In Formal Methods and Software Engineering. ICFEM 2013. LNCS, vol. 8144. Springer Berlin Heidelberg (2013)
19. Rezaki, A., Ural, H.: Construction of checking sequences based on characterization sets. Computer Communications 18, 911–920 (1995)
20. Rivest, R.L., Schapire, R.E. Inference of finite automata using homing sequences. In Machine Learning: From Theory to Applications. Springer Berlin Heidelberg, 51-73 (1993)
21. Sabnani, K., Dahbura, A. A protocol test generation procedure. Computer Networks 15: 285-297 (1988)
22. Simao, A.S., Petrenko, A. Generating checking sequences for partial reduced finite state machines. In TestCom/FATES. LNCS, vol. 5047, Springer Heidelberg, 153–168 (2008)
23. Simao, A., Petrenko, A. Checking sequence generation using state distinguishing subsequences. In The IEEE International Conference on Software Testing, Verification, and Validation Workshops, 48–56 (2009)
24. Soos, M. CryptoMiniSat - a SAT solver for cryptographic problems. http://www.msoos.org/cryptominisat (2009).
25. Vasilevski, M. P. Failure diagnosis of automata. Cybernetics, Plenum Publishing Corporation, New York, No 4, 653-665, (1973)
26. Yannakakis, M. and Lee, D. Testing finite state machines: fault detection. Journal of Computer and System Sciences, 50, 209-227 (1995)
27. Yao, M., Petrenko, A. and Bochmann, G.V. Conformance testing of protocol machines without reset. In The IFIP Thirteenth International Symposium on Protocol Specification, Testing and Verification. North-Holland Publishing Co. 241-256 (1993)