Inferring Finite State Machines Without Reset Using State Identification Sequences

Roland Groz[1(✉)], Adenilso Simao[2], Alexandre Petrenko[3], and Catherine Oriat[1]

[1] Université Grenoble Alpes, Saint-Martin-d'Hères, France
{Roland.Groz,Catherine.Oriat}@imag.fr
[2] Universidade de São Paulo, Saint-Martin-d'Hères, São Paulo, Brasil
adenilso@icmc.usp.br
[3] CRIM, Montréal, Canada
Alexandre.Petrenko@crim.ca

Abstract. Identifying the (finite state) control structure of a black box system from the traces observed in finite interaction is of great interest for many model-based activities, such as model-based testing or model-driven engineering. There are several inference methods, but all those methods assume that the system can be reset whenever necessary. In this paper, we address the issue of inferring a finite state machine (FSM) that cannot be reset; we propose a method, inspired by FSM-based testing generation methods. We assume classical testing hypotheses, namely that we are given a bound n on the number of states and a set W of characterizing sequences to distinguish states. To the best of our knowledge, this is the first model inference method that does not require resetting the system, and does not require an external oracle to decide on equivalence. The length of the test sequence is polynomial in n and the exponent depends on the cardinal $|W|$ of the characterization set.

Keywords: Finite state machines · Model inference · Testing

1 Introduction

For model-driven software engineering, model-based testing including, it is important to rely on models of the software artefacts. It is also essential that the models are up-to-date. In many contexts, however, such models are not available. In the last decade, interest has risen on methods to retrieve models from software artefacts, see, e.g., [1, 2, 4, 7, 10, 12, 17]. Depending on the context, goal and assumptions, various types of techniques have been considered for specification mining, reengineering or model inference. When source code is available, automated analysis of its structure can yield adequate models [5, 11].

Various algorithms have been proposed for inference of finite state machines. Such methods have been used to retrieve finite state behavioural models of black box components by testing them [16, 17]. Typically, such components could be accessed over a network, so that we do not even assume that the executable can be scrutinized: the system can only be observed at its interfaces [1]. This corresponds to a typical black

© IFIP International Federation for Information Processing 2015

K. El-Fakih et al. (Eds.): ICTSS 2015, LNCS 9447, pp. 161–177, 2015.
DOI: 10.1007/978-3-319-25945-1_10

box testing scenario, where the tester would send inputs to a system and observe its outputs. This scenario arises in many practical situations. For instance, a component has to be treated as a black box, when its internals cannot be available due to intellectual property constraints.

We assume that the System Under Test (SUT) can be modelled, at some level of abstraction on its inputs and outputs, as a Finite State Machine (FSM). FSM-based testing theory has shown that an FSM can be identified, i.e., the SUT can be tested to be proven equivalent to it, with the help of state identifying (distinguishing) sequences, constituting, e.g., a characterization set (W-set) of input sequences.

Existing inference algorithms assume that the black box can be reliably reset to its initial state. This makes the inference task somehow easier, because it is possible to start each new experiment from a known state, and explore progressively the neighbouring states. But there are cases where a black box cannot be reset, or where it is unsure that we can rely on it being restored to the same initial state. There are also cases where restarting the system completely might be very costly requiring a lot of time to reset the whole configuration (e.g., rebooting a machine, with possibly many software components to configure and reinitialize). Actually, our research was triggered by a case study from the SPaCIoS European project where we had to infer a software provided (as is common now) as a virtual machine. Although an i/o interaction in HTTP with it over the local network would require less than a millisecond, resetting the application would need more than a minute, around 10^5 more than an i/o.

In this paper, we propose an algorithm that can infer a black-box implementation without resetting it. This problem has similarities with classical testing methods, in particular the DS-method [6] and the W-method [18]. In classical testing methods, we are provided with a characterization set, and we derive a checking sequence or experiment that identifies a black box. The key difference is that classical testing methods start from a known specification machine, and just check whether the black box is equivalent (or conformant) to this specification. Therefore, those methods heavily rely on transfer sequences that make it possible to test a new state through a path known to transfer to the right state in the specification. In our context, since no specification is available, we cannot rely on known transfer sequences. Although the absence of reset had already been addressed by Hennie [6] and others in the case of a single distinguishing sequence, the task is made much harder in our context since we cannot return to a known state to compare the responses to a state of the specification. Moreover, we consider the case where the black box may not have a distinguishing sequence. This problem has been investigated in [6, 8, 13, 14], where a characterization set is used. It turns out that generating a checking sequence (without reset) from a known specification FSM with a characterization set is very costly, due to the fact that the sequences of this set have to be applied a number of times which is proportional to some exponential on the number of states in the specification [14]. Recent work [8, 13] has reduced the effort by some constant factor, which is a practical improvement; nonetheless, the complexity remains exponential.

As with other identification problems in FSM based testing, we rely on assumptions on the black box. First, we assume that an upper bound n on the number of states of the black box is known. Second, we are provided with a set W of input sequences that characterize the different states of the black box: each state produces a different set of output responses to these input sequences. Such a set could be derived from a previous version of the software or from domain-specific knowledge.

Rivest and Schapire [15] pioneered the inference of automata without reset, but their method was based on Angluin's L^* algorithm [3] which assumes that an oracle can answer equivalence queries. In a typical software testing context, such an oracle cannot be provided. Our method does not require any equivalence query, and there is no oracle apart from the black box. In order to guarantee equivalence, we just assume we know an upper bound on the number of states, which is a much weaker assumption. Identifying states by their i/o responses to sequences of W is somehow similar to the set of suffixes in L^* although we do not need suffix-closure contrary to L^*.

Our method relies on a localizer, a procedure that can reliably bring the black box to the same state all over again to make sure that responses to all sequences from the W-set are observed from a single state. Therefore, it is possible to extend progressively the knowledge of the transitions starting from that state, while being sure that we actually come back to a known (previously learnt) state. This procedure is inspired by the construction of locating sequence used in [6, 14]. The algorithm uses the localizer to start from a known state, which becomes the first learnt state. Thus, it does not require the black box to be initially in a particular state. It then tries to characterize progressively the tail state reached after the application of the localizer by repeatedly applying the sequences from W. By doing so, it will often end up in unknown states, therefore the algorithm alternates between applications of sequences from W to get more knowledge and applications of the localizer to restart from some previously learnt state. The algorithm identifies a correct model in $O(p(f + p)\ 2^p\ n^{p+2})$ inputs, where n is the number of states of the machine, f is the size of the input set and p is the size of the W-set. Hence, the length of the identification sequence is polynomial in the number of states, although the exponent depends on the number of characterizing sequences.

The paper is organized as follows. Section 2 provides definitions and notations for our method. Section 3 describes the inference procedure, while Sect. 4 is dedicated to the localizer subroutine. Section 5 illustrates the algorithm on a small example. Proofs of termination and correctness, as well as an upper bound on complexity are in Sect. 6. Section 7 discusses assumptions and Sect. 8 concludes.

2 Definitions

2.1 Basic Definitions

A Finite State Machine is a complete deterministic Mealy machine. Formally, a Finite State Machine (FSM) M is a 6-tuple $(S, s_0, I, O, \delta, \lambda)$, where

- S is a finite set of states with the initial state s_0,
- I is a finite set of inputs, and O is a finite set of outputs,
- δ: $S \times I \rightarrow S$ is a transition function, and
- λ: $S \times I \rightarrow O$ is an output function.

δ and λ are actually mappings, i.e., $dom(\delta) = dom(\lambda) = S \times I$ since we only consider complete machines. As M is deterministic, a tuple $(s, x) \in S \times I$ uniquely determines a *transition* of M. For simplicity we use (s, x) to denote the transition, thus omitting its output and final state. We extend the transition and output functions from input symbols to input sequences, including the empty sequence ε, as usual: $\delta(s, \varepsilon) = s$ and $\lambda(s, \varepsilon) = \varepsilon$, for $s \in S$; for $\alpha x \in I^+$, $\delta(s, \alpha x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$. An FSM M is said to be *strongly connected*, if for each pair of states $s, s' \in S$, there exists an input sequence $\alpha \in I^*$, such that $\delta(s, \alpha) = s'$; α is called a *transfer* sequence.

Two states $s, s' \in S$ are *distinguishable*, if there exists $\gamma \in I^*$, such that $\lambda(s, \gamma) \neq \lambda(s', \gamma)$. We say that γ distinguishes s and s'. Given a set $H \subseteq I^*$, states s and s' are *H-equivalent*, if $\lambda(s, \gamma) = \lambda(s', \gamma)$ for all $\gamma \in H$. Otherwise, i.e., if there exists $\gamma \in H$ such that $\lambda(s, \gamma) \neq \lambda(s', \gamma)$, the states are *H-distinguishable*. We define H-distinguishability and H-equivalence of machines as a corresponding relation between their initial states. An FSM M is *minimal*, if all states are pairwise distinguishable. In this paper, the machines are assumed to be minimal and strongly connected. A set of inputs W is a *characterization* set for an FSM M if each pair of states is W-distinguishable.

A sequence of input/output pairs $\alpha \in (IO)^*$ is a *trace*. Given traces α, β, and ω, such that $\omega = \alpha\beta$, we write $\omega \setminus \beta$, when we need to refer to the result of deleting the suffix β from the trace ω, resulting in trace α. The notation $\alpha \leq \omega$ is used to refer to a prefix α of the trace ω; $\omega{\downarrow}A$, where $A \subseteq I \cup O$, denotes the projection of ω to A, obtained by removing symbols that are not in A from the trace ω. Formally, $\varepsilon{\downarrow}A = \varepsilon$; if $x \in A$, then $(\alpha x){\downarrow}A = (\alpha{\downarrow}A)x$; if $x \notin A$, then $(\alpha x){\downarrow}A = (\alpha{\downarrow}A)$.

Given a machine $M = (S, s_0, I, O, \delta, \lambda)$ with a state s_i and an input sequence $\alpha \in I^*$, we use $tr_i(\alpha)$ to denote the trace from state s_i such that $tr_i(\alpha){\downarrow}I = \alpha$ and $tr_i(\alpha){\downarrow}O = \lambda(s_i, \alpha)$. Given a characterization set W, rather than naming or numbering states (such as "s_i"), we may refer to a state by its *state characterization* $Tr_i(W) = \{tr_i(w) \mid w \in W\}$. A state characterization Tr_i can be seen as a mapping q_i from W to $(IO)^*$ such that $q_i(w) = tr_i(w)$. The set of all mappings $Q = \{q_1, q_2, \ldots, q_n\}$ corresponds to the set of states of the machine. Namely, for $q \in Q$ and $s \in S$, we write $q \leftrightarrow s$ if $\forall w \in W$, $q(w){\downarrow}O = \lambda(s, w)$. Inferring an unknown FSM with the same characterization set, each mapping $q \in Q$ can then be considered as its state. In the sequel, we use B (instead of the general M) to refer to a black-box machine, P (instead of S) to refer to its states, whereas the inferred model (also called "conjecture", in line with L*) will have states denoted by Q.

2.2 Definitions for the Inference Method

Let $B = (P, p_0, I, O, \delta, \lambda)$ denote an FSM modelling a given black-box (BB) with the characterization set W. As the BB cannot be reset, it is possible to observe only a single trace from it. We introduce the notion of *labelling* function C: $(IO)^* \rightarrow Q \cup \{\perp\}$, which maps each prefix of an observed trace to a state q of Q, if it can be inferred that the machine B will be in state p such that $q \leftrightarrow p$ after the observed trace; otherwise C maps the prefix to an unknown state, denoted $\perp$.

Let ω be an observed trace. A labelling C is *deterministic* for ω if the following property holds:

$$\begin{aligned}&\text{For } \alpha, \alpha' \in (IO)*, \text{ such that } \alpha \leq \omega, \text{ and } \alpha' \leq \omega, \text{ if } C(\alpha) = C(\alpha') \neq \perp,\\ &\text{then for any } \beta, \beta' \in (IO)*, \text{ such that } \alpha\beta \leq \omega, \alpha'\beta' \leq \omega \text{ and } \beta \downarrow I = \beta' \downarrow I,\\ &\qquad\qquad\qquad \text{we have } \beta = \beta' \text{ and } C(\alpha\beta) = C(\alpha'\beta').\end{aligned}$$

Obviously, a labelling should be deterministic. It should also be *consistent* with the state characterizations, i.e., the following property should also hold:

$$\text{For all}\, \alpha, \gamma \in (IO)*\ \text{ such that}\, \alpha\gamma \leq \omega,\ \text{if}\, C(\alpha)\ = q \neq \bot \text{and} \gamma \downarrow I = w \in W, \\ \text{then}\, q(w)\ = \gamma.$$

We capture the notion of known (or inferred) state by stating that a labelling C should be revealing. Let $B = (P, p_0, I, O, \delta, \lambda)$, a labelling C is *revealing* w.r.t. B if, and only if $\forall \alpha \in (IO)^*$ ($\exists p \in P$ s.t. $\forall p' \in P$ $(\delta(p', \alpha{\downarrow}I)) = p) \Rightarrow \exists q$ s.t. $C(\alpha) = q$ and $q \leftrightarrow p$. In other words, this property forces C to be defined on any trace α that is "homing" [9], i.e., which leads unambiguously to a single state with a known state characterization in the FSM B, irrespective of the start state.

The labelling function and state characterizations are the main components of the proposed method. To ease the presentation and simplify the algorithm, we also introduce several auxiliary notations and definitions, derived from the labelling function.

A trace is *verified* if it is a subtrace of the observed trace ω and its start and end states are known (labelled by C). Each verified trace is the result of execution of several transitions between the known states of the FSM B. We define a set of tuples containing a verified trace together with its start and end states $V \subseteq Q \times (IO)^* \times Q$. Formally, $V = \{(q, \alpha, q') \mid \exists\sigma\alpha \leq \omega, C(\sigma) = q, C(\sigma\alpha) = q'\}$. Moreover, since we assume that C is deterministic, for all $(q, \alpha, q') \in V$, and all $\chi \leq \omega$, such that $C(\chi) = q$, we have $C(\chi\alpha) = q'$.

We will also use a subset of it to refer to single transitions, called *verified* transitions, and not to their sequences. Let $T \subseteq V$, $T = \{(q, xo, q') \mid x \in I, o \in O$ and $\exists\sigma xo \leq \omega, C(\sigma) = q, C(\sigma xo) = q'\}$ be the set of verified transitions. We will use as well the term of *input verified in state* when we do not need to refer to the end state of a verified transition. Thus $R = \{(q, x) \in Q \times I \mid \exists o \in O, q' \in Q, (q, xo, q') \in T\}$ is the set of inputs verified in corresponding states.

Moreover, since we will discover new states as the observed trace grows, we will have a set of known states associated with a current trace ω. Q_C denotes a set of states *discovered* by ω such that $q \in Q_C$ iff $C(\alpha) = q$, for some $\alpha \leq \omega$.

We shall be able to derive from ω a *complete* conjecture, i.e., the FSM B, when for all $q \in Q_C$, for all $x \in I$, $(q, x) \in R$.

We further assume that the set W is an ordered set of p sequences and use $tr(w_p)$ to denote a subtrace produced by the BB when the last sequence of W viz. w_p is applied.

Finally, $K \subseteq Q \times (IO)^+ \times (IO)^*$ is used to keep track of the applications of $w \in W$ in a state q followed by either a transition or a trace of a sequence from W. $(q, \alpha, \gamma) \in K$, if there exists a trace β, such that $\beta\alpha\gamma \leq \omega$, $C(\beta) = q$, $\alpha{\downarrow}I \in I \cup W$ and $\gamma{\downarrow}I \in W$.

3 Inference Procedure

A localizer procedure $L(\omega, W)$ is used to ensure that we continue learning transitions from states visited before. The localizer procedure, given a current trace ω, produces an updated trace ω', a set of traces $Tr(W)$, and the last appended trace $tr(w_p)$ to label the state which has produced it and is thus identified with $Tr(W)$. Recall that a state is defined by its answers to sequences from W, i.e., by $Tr(W)$. At the end of the localizer, we know the BB is in state $\delta(Tr(W), w_p)$, and the state reached after $\omega\backslash tr(w_p)$ can be labelled as $Tr(W)$.

Before defining the localizer, we present the main algorithm that calls it.

```
Inference procedure
Initialize: $K = R = V = \varnothing$
$(\omega, q_0, tr(w_p)) := L(\varepsilon, W)$                % Home into (or through) a known state
$C(\omega \setminus tr(w_p)) := q_0$
$Q_C := \{q_0\}$
while $\exists q' \in Q_C$ and $x' \in I$, such that $(q', x') \notin R$ do
    if $C(\omega) = q \neq \bot$, then                % If current state is known
                                                  % move to unverified transition
        Find a shortest $\alpha = \alpha_1\alpha_2\ldots\alpha_k$,
            s.t. for all $i$ $(q_i, \alpha_i, q_{i+1}) \in V$, $q_1 = q$, and $x \in I$ s.t. $(q_{k+1}, x) \notin R$
        Apply $\alpha\downarrow I$, observe $\alpha$
        $\omega := \omega\alpha$
        $\chi := \omega$
        Apply $x$, observe $xo$                   % observe transition
        $\sigma := xo$
        $\omega := \omega xo$
    else                                          % else use a previous known state
        Find the shortest $\gamma$, s.t. $C(\omega \setminus \gamma) \neq \bot$ % could be shorter than $w_p$
        $\chi := \omega \setminus \gamma$
        $\sigma := \gamma$
    end if
    $q := C(\chi)$                                % here $\omega$ is unknown
    Choose $w \in W$, such that there is no $tr(w)$ s.t. $(q, \sigma, tr(w)) \in K$
    Apply $w$, observe $tr(w)$                    % improve characterization of $(q, \sigma)$
    $\omega := \omega tr(w)$
    $K := K \cup \{(q, \sigma, tr(w))\}$
    if $\{w \in W \mid (q, \sigma, tr(w)) \in K\} = W$, then      % full characterization reached
        $C(\chi\sigma) := \{tr(w) \mid w \in W$ and $(q, \sigma, tr(w)) \in K\}$
        $Q_C := Q_C \cup \{C(\chi\sigma)\}$
        Update $V$, $R$, $K$ and $C$
    end if
    if $C(\omega) = \bot$ then                    % move to an identified state
        $(\omega, q', tr(w_p)) := L(\omega, W)$
        $C(\omega\backslash tr(w_p)) := q'$
        $Q_C := Q_C \cup \{q'\}$
        Update $V$, $R$, $K$ and $C$
    end if
end while
```

Build the conjecture from Q_C and T.

The sets K, V (hence T) and R should be updated to reflect the changes in C. The update is done by applying the following rules as long as possible (fix point iterations with monotonic increase of the sets):

Rule 1 If $C(\beta) = q$, $C(\beta\alpha) = q'$, $\beta\alpha \le \omega$, then $(q, \alpha, q') \in V$

Rule 2 If $C(\beta) = q$, $\beta\alpha \le \omega$ and $(q, \alpha, q') \in V$ then $C(\beta\alpha) = q'$

Rule 3 If $(q, x, q') \in V$, for $x \in I$, then $(q, x) \in R$

Rule 4 If $C(\beta) = q$, $\beta\alpha\gamma \le \omega$, $\alpha{\downarrow}I \in I \cup W$, $\gamma{\downarrow}I \in W$, then $(q, \alpha, \gamma) \in K$

Rule 5 If $\exists\alpha$ s.t. $\{w \in W \mid (q, \alpha, tr(w)) \in K\} = W$ then $\forall\beta\alpha \le \omega$ s.t. $C(\beta) = q$, we have $C(\beta\alpha) = \{\gamma \mid (q, \alpha, \gamma) \in K\}$

Notice that although in this definition of updates we derive V and K sets to be consistent with their definitions, we do not need to add all elements to them. Typically, V is transitively closed, but we only need a transitive reduction of it. If $(q, \alpha, q') \in V$ and $(q', \beta, q'') \in V$ then in the implementation of the procedure we would not store $(q, \alpha\beta, q'')$ in V because we use V to find a shortest concatenation of sequences that would themselves be in V or to verify transitions (sequences of length 1).

4 Localizer Procedure

In our inference procedure, the main use of the localizer procedure is to ensure that when the BB is in an unknown state q, we can restart a test from a state q' of the BB for which we can be certain that we know all its traces for all sequences from the W-set, and thus we can identify state q' of the BB. Actually, we present a generalized procedure that can work with any set of sequences Z, where Z may not be a fully characterizing set, i.e., there could be non-equivalent states in the BB that are Z-equivalent. The localizer procedure will be defined recursively with increasing subsets Z of W.

For the design of this localizer procedure, we extend an idea that had already been investigated by Hennie [6]. The key idea is that since the number of states is finite and bounded by n, by repeatedly applying a given input sequence α we can observe at most n different output sequences. This implies that in the state reached applying n times α the BB must have reached a cycle, coming to one of the states visited after some α^i. But we do not know which one because different states could still have the same output response to α. The length of the cycle is itself bounded by n. As the proof will show, it is enough to repeat α another $n - 1$ times (so $2n - 1$ in total) in the worst case to identify the cycle and be able to know what would be the response from the BB to the $2n$-th application and all subsequent applications of α.

This means that after $2n - 1$ applications of α, we are in a state p and we can apply another input sequence β, and be sure that we know the response of p to both α and β. If $\{\alpha, \beta\}$ is a W-set, we can continue a test from a state p that has been fully characterized, or at least from $\delta(p, \beta)$ which is a well-defined state. Jumping to more than two sequences is a bit more elaborate, but it is ensured by the recursive definition of the localizer. Actually, we build embedded cycles.

We illustrate the above discussion on the example in Fig. 1.

This 3-state machine has no distinguishing sequence. However, states can be fully characterized by their traces for sequences from the set $W = \{a, b\}$. Suppose we start

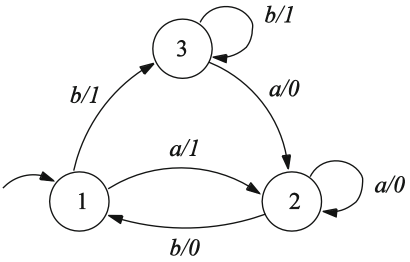


Fig. 1. An FSM to be inferred.

our experiment when the machine is in state 1 by sending an input *a*. We shall get an output 1. The localizer will repeatedly apply the input sequence *a*. The second and third outputs will be 0. Since we know the machine has no more than 3 states, after 3 applications of *a*, the fourth application will elicit an output which must be one of those previously seen, 0 or 1. In this case, it will be a 0. Now, since the last 3 outputs were 0s, we can be sure that the fifth application of *a* would elicit a 0. We can now apply a *b*, and we get 0 as output, since we were trapped in state 2. We know that just before applying the *b*, we were in a state that answers 0 to both *a* and *b*. Since we assume *W* to be a fully characterizing set, the state is distinguishable from all others. We shall call it the state $\{a0, b0\}$, which corresponds to state 2 in the example.

Note that at the end of the localizer procedure we are not in the state *p* that has been fully characterized, but past it: in the example of Fig. 1 we are in state 1 instead of 2, and more generally with $W = \{\alpha, \beta\}$ we are actually in $\delta(p, \beta)$. This will also be the case for our generalized localizer: at the end of the procedure, the BB will be in a state that follows the application of the last input sequence w_k of the *Z* set, and the state characterized with the set *Z* will be the state reached just before applying the last w_k sequence. This is not an issue because our inference procedure handles this situation. Note that this means that the sequence built by the localizer is not a homing sequence [9], although it could easily be adapted to provide a kind of homing sequence, modulo the fact that the states are initially unknown, as we discover state characterization while applying the localizer. To make a homing sequence, we would just have to add an empty sequence ε as the last sequence w_{k+1} of *Z*.

The localizer procedure is formulated for fixed alphabets *I* and *O*, and an assumed bound on the number of states in the black box *n*. The procedure uses a given sequence ω that has already been observed from the BB and has left it in an unknown state, and given a non-empty ordered set *Z* of input sequences. It returns an updated ω′ trace (ω concatenated with the new inputs applied by the localizer and observed outputs), the set of traces for *Z* that can be confirmed as the traces of the state *p* reached just before the last w_k sequence of *Z*, and the trace suffix observed from the state *p*.

Localizer procedure

Signature L: $((IO)^* \times (I^*)^+) \to ((IO)^* \times 2^{(IO)^*} \times (IO)^*)$

```
procedure $L(\omega, Z)$
if $|Z| = 1$, i.e., $Z = (w)$ then
    Apply $w$, observe $tr(w)$
    $\omega := \omega tr(w)$
    return $(\omega, \{tr(w)\}, tr(w))$
else let $Z = (w_1, \ldots, w_k)$, $(w_1, \ldots, w_{k-2}, w_{k-1}) = Z_1$, $(w_1, \ldots, w_{k-2}, w_k) = Z_2$
        for $i$ from 0 to $2n - 2$ do
            $(\omega', Tr(Z_1), \tau_i) := L(\omega, Z_1)$            % $\tau_i$ is $tr(w_{k-1})$
            $\omega := \omega'$
        end for
        Find greatest $j$, $0 \le j \le n - 1$ s.t. for all $m \in [0, n - 2]$ $\tau_{j+m} = \tau_{n+m}$
        $(\omega', Tr(Z_2), tr(w_k)) := L(\omega, Z_2)$
        return $(\omega', Tr(Z_2) \cup \{\tau_{j+n-1}\}, tr(w_k))$
end if
```

Note that if $k = 2$, the list $w_1, \ldots, w_{k-2}$ is empty, hence $Z_1 = (w_1)$ and $Z_2 = (w_2)$.

In the **for** loop, we repeatedly apply the same localizer sequence $L(\omega, Z_1)$. We store in an array τ_i the trace observed for the last element of Z_1, i.e., $tr(w_{k-1})$. After the n-th application, we must have reached a cycle, where τ_{n+1} must be equal to some previous τ_i. But it could be equal to several τ_i therefore we must continue another $n - 1$ times to ensure that we have completed the cycle. In any case, after $2n - 1$ applications, even if the length of the cycle is unknown, since $\tau_{j+n-2} = \tau_{n+n-2}$ is in the cycle, applying w_{k-1} instead of w_k would have produced τ_{j+n-1}. Now we know what would be the answer of the state to w_{k-1} and we can apply w_k instead.

We illustrate the localizer procedure on our example in Fig. 1. We start from initial state 1, with $\omega = \varepsilon$ and $Z = (a, b)$. We have $Z_1 = (a)$ and $Z_2 = (b)$. Since the **for** loop is executed $2n - 1 = 5$ times, we apply $w_1 = a$ to BB 5 times and reach the 5th step of the trace. We get the trace $a1a0a0a0a0$, with $\tau_0 = a1$, $\tau_1 = a0$, $\tau_2 = a0$, $\tau_3 = a0$ and $\tau_4 = a0$. At this point, we recognize the cycle $\tau_2 = \tau_3$ and $\tau_3 = \tau_4$ ($j = 2$) and we know that if we apply a, we would get 0. Subsequently we apply $w_2 = b$ and get 0.

We identify the state $q_0 = \{a0, b0\}$ The result of the localizer procedure is thus $(a1a0a0a0a0b0, q_0 = \{a0, b0\}, b0)$.

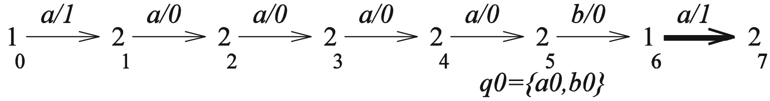
The input sequence used by the localizer is a fixed sequence. In our small example, it is a^5b. For a machine with at most n states and $W = \{w_1, w_2\}$, the input sequence is $w_1^{2n-1}w_2$; with $W = \{w_1, w_2, w_3\}$, it becomes $(w_1^{2n-1}w_2)^{2n-1}(w_1^{2n-1}w_3)$, and with $W = \{w_1, w_2, w_3, w_4\}$, it becomes $[(w_1^{2n-1}w_2)^{2n-1}(w_1^{2n-1}w_3)]^{2n-1}$ $(w_1^{2n-1}w_2)^{2n-1}(w_1^{2n-1}w_4)$ and so on.

5 Example of FSM Inference

We now illustrate the proposed method on the example in Fig. 1. As illustrated above, applying the localizer procedure allows us to identify a first state of the machine: $C(\omega \setminus b0) = q_0 = \{a0, b0\}$. Thus, the prefix obtained in the 5th step is labelled by q_0. C is updated in the third line of the inference procedure.

We then enter the main **while** loop in the inference procedure. As the last state of ω (6th step in the trace) is not yet labelled we proceed to the else part of the first **if** statement. Then $\gamma = b0$, $\chi = a1a0a0a0a0$ and $\sigma = b0$.

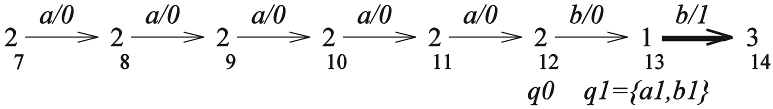
After the first **if** statement, $q = q_0$, K is empty, and we choose $w = a.$ We apply a, observe $a1$ and arrive at the 7th step. We get $K = \{(q_0, b0, a1)\}$, $V = R = \varnothing$.



Since the condition of the second **if** statement is false, we proceed with the third **if** statement, which is true. We apply again the localizer procedure, and label the state after $a0a0a0a0a0$ (12th step) with q_0: $C(12) = q_0$. We then restart the **while** loop.

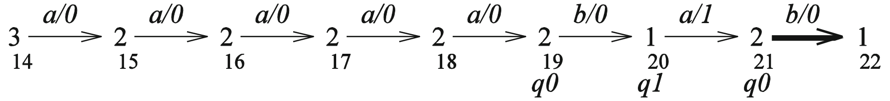
We enter the **else** part of the **if** statement, with $\gamma = \sigma = b0$. Then $q = q_0$, and we choose $w = b$. We apply b, observe $b1$ and arrive at the 14th step. We get $K = \{(q_0, b0, a1), (q_0, b0, b1)\}$.

The condition of the second **if** statement is now true, we have thus identified a new state $q_1 = \{a1, b1\}$ and $C(13) = q_1$. Then $V = \{(q_0, b0, q_1)\}$, $R = \{(q_0, b)\}$ and $C(6) = q_1$. Thus $K = \{(q_0, b0, a1), (q_0, b0, b1), (q_1, a1, a0)\}$.

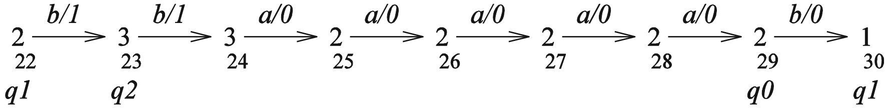


We now execute the third **if** statement: we apply the localizer procedure and arrive at the 20th step. $C(19) = q_0$ and $C(20) = q_1$. K is unchanged. We then restart the **while** loop. Since the last (20th) step is now labelled, we enter the **then** part of the first **if** statement. $\alpha = \varepsilon$ and we choose $x = a$. We apply a, observe $a1$ and arrive at step 21. We have $\sigma = a1$. Now $q = q_1$, and we choose $w = b$. We apply b, observe $b0$ and arrive at step 22. We update $K = \{(q_0, b0, a1), (q_0, b0, b1), (q_1, a1, a0), (q_1, a1, b0)\}$.

Since the condition of the second **if** statement is true, we identify $C(21) = q_0 = \{a0, b0\}$. We update V, R, C and K: $V = \{(q_0, b0, q_1), (q_1, a1, q_0)\}$, $R = \{(q_0, b), (q_1, a)\}$, $C(7) = q_0$, $C(22) = q_1$, $K = \{(q_0, b0, a1), (q_0, b0, b1), (q_1, a1, a0), (q_1, a1, b0), (q_0, a0, a0), (q_1, b1, a0)\}$. The trace at the last step (22) is labelled, and the condition of the third **if** statement is false.


3 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{b/0}$ 1 $\xrightarrow{a/1}$ 2 $\xrightarrow{b/0}$ 1
14 15 16 17 18 19 20 21 22
q0 q1 q0

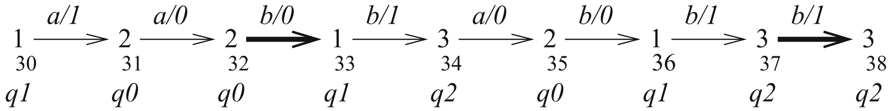
We then restart the **while** loop, which allows us to identify a new state q_2 = {a0, b1} and label C(23) = $q_{2.}$ We apply the localizer procedure and arrive at step 30. C(29) = q_0, C(30) = q_1. At this point, V = {(q_0, b0, q_1), (q_1, a1, q_0), (q_1, b1, q_2), (q_2, b1a0a0a0a0a0, q_0)}, K = {(q_0, b0, a1), (q_0, b0, b1), (q_1, a1, a0), (q_1, a1, b0), (q_0, a0, a0), (q_1, b1, a0), (q_1, b1, b1), (q_2, b1, a0)}, R = {(q_0, b), (q_1, a), (q_1, b)} and C(14) = q_2.


2 $\xrightarrow{b/1}$ 3 $\xrightarrow{b/1}$ 3 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{b/0}$ 1
22 23 24 25 26 27 28 29 30
q1 q2 q0 q1

We restart the **while** loop. As the last step (30) is labelled, we enter the **then** part of the first **if** statement. As all transitions from q_1 are now known, we must apply a transfer sequence to move to an unverified transition. We choose to go to q_0 with α = a1 and x = a. We apply a, observe a1, apply a again, observe a0 and arrive at step 32. σ = a0.

We choose w = b, apply b, observe b0, arrive at step 33 and add (q_0, a0, b0) to K. The condition of the second **if** statement is true, we have C(32) = q_0 and we add (q_0, a0, q_0) to V. Now R = {(q_0, b), (q_1, a), (q_1, b), (q_0, a)}, C(7) = C(7) = ... = C(11) = q_0, C(33) = q_1.

After applying the **while** loop two more times, we obtain the trace below, with V = {(q_0, b0, q_1), (q_1, a1, q_0), (q_1, b1, q_2), (q_0, a0, q_0), (q_2, a0, q_0), (q_2, b1, q_2)} and R = {(q_0, b), (q_1, a), (q_1, b), (q_0, a), (q_2, a), (q_2, b)}. All transitions are known and C(38) = q_2.


1 $\xrightarrow{a/1}$ 2 $\xrightarrow{a/0}$ 2 $\xrightarrow{b/0}$ 1 $\xrightarrow{b/1}$ 3 $\xrightarrow{a/0}$ 2 $\xrightarrow{b/0}$ 1 $\xrightarrow{b/1}$ 3 $\xrightarrow{b/1}$ 3
30 31 32 33 34 35 36 37 38
q1 q0 q0 q1 q2 q0 q1 q2 q2

We now exit from the **while** loop and V gives us the inferred finite state machine, which is isomorphic to the FSM in Fig. 1.

6 Proofs and Complexity

6.1 Proof of Localizer

The key theorem for our approach states that the localizer will ensure that the BB reached (before the application of the last sequence from W) a state that can be labelled with the mapping returned by the localizer.

Theorem 1. When $Z = W$, from any starting state, the localizer will return (ω', X, β) such that $C(\omega' \setminus \beta) = X$ and $X{\downarrow}I = W$.

To prove this theorem, we first introduce notations and two lemmas.

Let IL be the input projection of a trace added by a localizer $L(\omega, Z)$, $Z = (w_1, \ldots, w_k)$ returning ω' as the first element, i.e., $\omega'{\downarrow}I = \omega{\downarrow}I = IL(w_1, \ldots, w_k)$. We denote $P = IL \setminus w_k$, thus, $IL(w_1, \ldots, w_k) = P(w_1, \ldots, w_{k\text{-}1})\ w_k$.

Lemma 1. For all $k \geq 1$, we have that $P(w_1, \ldots, w_k) = [P(w_1, \ldots, w_{k-1})\ w_k]^{2n-1}P(w_1, \ldots, w_{k-1})$.

Proof. This lemma is proved by induction from $k = 1$ for P, accordingly we start with $IL(w_1, w_2)$. In the loop, $Z_1 = (w_1)$ and $Z_2 = (w_2)$ hence we apply w_1^{2n-1} then w_2. So $IL(w_1, w_2) = w_1^{2n-1}w_2$, $P(w_1) = w_1^{2n-1}$ and $P(w_1, w_0)$ is an empty sequence by definition of IL. Thus indeed $P(w_1) = [P(w_1, w_0)\ w_1\ ]^{2n-1}P(w_1, w_0)$.

Assuming it is true for some $k \geq 1$, then by definition of P we have:
$IL(w_1, \ldots, w_{k+2}) = P(w_1, \ldots, w_{k+1})w_{k+2}$.

And by the structure of the localizer, we have:
$IL(w_1, \ldots, w_{k+2}) = IL(w_1, \ldots, w_{k+1})^{2n-1}IL(w_1, \ldots, w_k, w_{k+2}) = ([P(w_1, \ldots, w_k)w_{k+1}]^{2n-1}P(w_1, \ldots, w_k)w_{k+2}$.

Using the induction hypothesis, we get:
$P(w_1, \ldots, w_{k+1}) = = ([P(w_1, \ldots, w_k)w_{k+1}]^{2n-1}P(w_1, \ldots, w_k)$. QED.

Lemma 2. Let ω' be a trace returned by the localizer for $Z = (w_1, \ldots, w_k)$ and $B = (S, s_0, I, O, \delta, \lambda)$, then for all $w \in Z$, $\lambda(s, w) = tr(w){\downarrow}O$, where $s = \delta(s_0, \omega' \setminus tr(w_k){\downarrow}I)$.

Proof. We prove this by induction on $|Z|$.

For $|Z| = 1$, $tr(w)$ is the β just observed on applying w at the end of ω which was $\omega' \setminus \beta$; thus $\lambda(s, w) = \beta{\downarrow}O = tr(w){\downarrow}O$.

Let us assume the property holds for $|Z| = k - 1 \geq 1$. Since Z_2 has k - 1 elements, we already have for all $w \in Z_2$, $\lambda(s, w) = tr(w){\downarrow}O$ (see how s is defined in Lemma 2). The only thing which we have to prove is that $\lambda(s, w_{k\text{-}1}) = tr(w_{k\text{-}1}){\downarrow}O = \tau_{j+n\text{-}1}{\downarrow}O_.$ Actually, $(\omega' \setminus tr(w_k)){\downarrow}I = (\omega{\downarrow}I).P(w_1, \ldots, w_{k\text{-}1})$ by definition of P. Let us denote $M = P(w_1, \ldots, w_{k\text{-}2})$, and $q = \delta(s_0, \omega{\downarrow}I)$. Using Lemma 1, we have $s = \delta(q, P(w_1, \ldots, w_{k\text{-}1})) = \delta(q, [Mw_{k\text{-}1}]^{2n-1}M) = \delta(q, M[w_{k\text{-}1}M]^{2n-1})$. Let $q_{\mathrm{i}} = \delta(q, M[w_{k\text{-}1}M]^{i})$, and $\lambda_i = \lambda(q_i, w_{k\text{-}1})$ for $i \in [0, 2n - 1]$. Note that $s = q_{2n\text{-}1}$ and according to the notations of the algorithm, for all $i \in [0, 2n$ - $2]$ $\lambda_i = \tau_i{\downarrow}O$. Since there are at most n distinct states, and we repeat the sequence $w_{k\text{-}1}M$, q_n must be equal to one of the $q_{j'}$ for some $j' < n$. Then, for all $m \in [0, n - 1]$, $\lambda_{j'+m} = \lambda_{n+m}$. Hence there is at least one such j, as required by the algorithm.

- If $q_n = q_p$ for some $p > 0$, then the length of the cycle is $n - 1$ at most and for any $j < n$ such that for all $m \in [0, n - 2]$, $\lambda_{j+m} = \lambda_{n+m}$ then we have: $\lambda_{j+n\text{-}1} = \lambda_{n+n\text{-}1} = \lambda(s, w_{k\text{-}1})$.
- If the greatest j is 0, this means there is no cycle of output of length less than n, then $q_n = q$, $q_n = q_0$ and $s = q_{2n\text{-}1} = q_{n\text{-}1}$ and $\lambda(s, w_{k\text{-}1}) = \lambda_{j+n\text{-}1}$ since $j = 0$.
- So in both cases we have $\lambda(s, w_{k\text{-}1}) = \lambda_{j+n\text{-}1} = \tau_{j+n-1}{\downarrow}O$.

QED.

It may be worth mentioning that taking the largest j means we identify the shortest cycle of outputs. The cycle of states entered by the implementation will have a length that is divisible by n–j: with the notation of the proof, $n - j$ divides $n - j'$.

We can now proceed to the proof of the Theorem 1. Recall that we denote states of the BB by their $Tr(W)$.

Actually, in all cases, $Tr(X)\downarrow I = Z$. This directly follows from the recursive definition: it holds for a singleton, and when we compute Tr for $(w_1, \ldots, w_k)$ we produce a set of traces that adds some $tr(w_{k\text{-}1})$ thus the set of traces is computed on $Z_2 = (w_1, \ldots, w_{k\text{-}2}, w_k)$. If $Z = W$ then by Lemma 2, the definition of C and the fact that C is revealing, we have $C(\omega' \setminus tr(w_k)) = X$ and $X\downarrow I = W$. QED.

6.2 Proof of the Method

To prove the correctness of the method we need to demonstrate that the mappings correspond to the states of the BB. We will use the following theorem.

Theorem 2. If $q \in Q_C$ then $\exists p \in P$ such that $q \leftrightarrow p$.

This theorem follows from the definitions of C and Q_C. The proof of the method itself (captured by Theorem 3) proceeds in several steps following the structure of the algorithm. In particular we must prove that all the steps that rely on an existence ("find", "choose") are well defined and will find an element, and we show that the algorithm increases monotonically the set of known states and transitions, and terminates when a conjecture can be built which is equivalent to the FSM $B = (P, p_0, I, O, \delta, \lambda)$.

1. On every entry in the **while** loop, either we are in a known state, i.e. the BB is in a state p such that $C(\omega) = q \leftrightarrow p$ or we are at the end of a localizer, i.e., $C(\omega \setminus tr(w_p))$ is defined (a known state). This is trivial on the first entry in the loop, since we just applied the localizer and did not move in the BB. And at the end of a cycle in the loop, we reach either after the case $C(\omega) = \perp$, in which case we reapply the localizer and are again at the end of it, or $C(\omega)$ is defined.
2. Find a shortest α: since we entered the while loop, we know that $\exists q' \in Q_\omega$ and $x \in I$, such that $(q', x) \notin R$. Moreover, we are in a state q. Then either $\exists x' \in I$, such that $(q, x') \notin R$, and we just have to pick $\alpha = \varepsilon$; or we know all successors of q. In that case, we build the connected graph reachable from q through verified transitions: if one of them has an unverified transition, we pick the one closest to q, i.e., that yields the shortest α. Otherwise, this would imply that there is a $q' \in Q_C$ such that there is a prefix $\exists \sigma \leq \omega$, $C(\sigma) = q'$ which would be distinct from all the successor states of q that constitute a strongly connected component. Since all successor states and this state q' are all paired (through the $\leftrightarrow$ relation) to states of the BB (by Theorem 2), this would contradict the assumption that the BB is strongly connected.
3. "Else find the shortest γ". In this else branch, we are in the case where $C(\omega) = \perp$. The existence of γ follows from 1: $\gamma = w_p$ in this case.
4. "Choose w": this comes one line after the "end if". If we followed the "else" branch, state $C(\chi\sigma) \notin dom(C)$ and there is a w for which $(q, \sigma, tr(w)) \notin K$. If we followed the "then" branch, then $(q_{k+1}, x) \notin R$ and $\sigma{:=}\,xo$ and by definition of R, we again have a w.
5. Progress: each loop iteration adds an element to K (by the structure of the loop), either at the end of a transition (xo) or at the end of a localizer.
6. Termination: there is a finite number of transitions in the BB (hence in the set Q_C by Theorem 2), each one is followed by a finite number of traces from the (finite)

set W. Hence the potential number of elements in K is finite, and the process will terminate.

7. Building the conjecture: based on Theorem 2, we can build a conjecture that is isomorphic to the BB by connecting all the transitions of R when the algorithm terminates.

Theorem 3. The inference procedure terminates and yields a conjecture that is isomorphic to the minimal FSM modelling the BB.

6.3 Complexity

As in all testing procedures, interacting with the system under test contributes mostly to the cost. Therefore we analyse the worst case complexity as the length (number of inputs) of the test sequence created by the procedure. The key element is the length of the localizer. Let L be the length of $L(\omega, W)$ and $|w_i|$ the length of an element of W. As in the notations of the algorithm, $p = |W|$ is the number of elements of the characterization set, n is the number of states, and f is the number of inputs (fan-out). Thus, L can be computed as follows.

$$L = \big(((|w_1|) \times (2n-1) + |w_2|) \times (2n-1) + \ldots + |w_{p-1}|\big) \\ \times (2n-1) + \big((|w_1|)\ldots \times (2n-1) + |w_{p-2}|\big) + |w_p|.$$

$O(L) = |w1|\ (2n–1)p$ since we can assume that for any i, we have $|wi| < n$: it is known that in a complete FSM with n states, any two states can be distinguished by a sequence of length no more than n. Thus, the length of the localizer is polynomial in n and exponential in p.

We apply the localizer at most once for each execution of the loop. The loop itself adds at least one element to K, thus the number of iterations of the loop is $n(f + p)$ p. Within the loop, the extension to ω would be $U = \mathbf{max}(|\alpha| + 1 + |w|, |\gamma| + |w|)$. $|\gamma| + |w|$ is bounded by $2n$, and $|\alpha|$ is bounded by n^2. Then the overall complexity would be $L + n$ $(f + p)p\ (L + U)$ bounded by $(n(f + p)p + 1)(L + n^2 + n + 1)$. As a coarse bound, we have $(n(f + p)p + 1)(n(2n - 1)^p + n^2 + n + 1)$ which is $O(p(f + p)\ 2^p n^{p+2})$, again polynomial in n and exponential in p.

Of course, this is a coarse upper bound for a worst-case complexity, and it is unsure it could be reached or even approached. As can be seen from the example, as soon as states and transitions are added to Q and V, the number of uses of the localizer decreases. Given the structure of the localizer, the sequences in W would be ordered by increasing length. And the localizer could be shortened because we may recognize cycles before reaching $2n - 1$ iterations.

Our experiments with hundreds of randomly generated machines show that for $n = 15, f = 10$, inference is achieved in around 10^4 steps for $p = 2$ (around 7000 when $|w_1| = 1$, and around 15000 when $|w_1| = 2$), i.e., around 500 times (3 orders of magnitude) less than the theoretical bound. To put this in perspective, we had initially tried classical inference methods [12, 16] on web applications, where a reset would typically take more than 1 min (restarting a virtual machine and restoring data configuration), whereas interacting with it over a local network would take around 1 ms per I/O pair. In such typical contexts, an inference sequence of length 10^5 would require no more time than a single reset.

7 Discussion

Our method assumes we know an upper bound n on the number of states of the BB and a characterization set W for it. We now discuss the impact of these assumptions. First let us remark that the localizer does not assume that W is characterizing. It can work with any set Z and just ensures that we characterize a state reached w.r.t. Z-equivalence [12]. However, an incorrect value for n could disrupt the localizer.

Actually, if either n or W is wrong (i.e., the assumptions for the BB are not satisfied), then two cases can occur:

1. The inference procedure runs to the end, but produces an FSM that is not isomorphic to a model of the BB.
2. The inference procedure detects an inconsistency (or fails to converge).

In the first case, we can either consider that the inference procedure provides a model that is approximate, and consistent with all observations: it will have produced an unrefined model of the BB that may be enough for the intended use of the model. Alternatively, we could now use the inferred model to check whether the BB conforms to this model, by some method of conformance testing, e.g., randomly walking through the model until we find a discrepancy or conclude on some level of confidence for the approximated model.

In the second case, let us first consider the case where n was an incorrect bound (too low), and this impacts the localizer. The only problem that can occur there is in the line "find the greatest j". We may fail to recognize a cycle in $2n - 1$ iterations. This can be easily addressed: first, we would continue iterating the **for** loop until we spot such an output cycle; then, we increase the value of n accordingly; at this point, we restart the whole inference procedure, because we cannot trust the previous equivalence of states.

Let us now consider the case where W would not be characterizing. This will not create any problem in the localizer, but in the main procedure. When we reapply a verified sequence α to go to a given state q_{k+1} or when we reapply an input x we may get a different output. This implies that the sequence we just applied distinguishes two states that we thought were equivalent. That sequence, or a part of it (up to the first divergence) could be added to W.

This can also be related to the initial assumption about W. The main issue is about how a characterizing set for a black box can be known. One possibility could be that we had such a set for a previous version of an implementation, and we take it as input of the method when we need to infer an updated implementation. It may also be possible to formulate domain-specific heuristics to identify candidate characterization sets.

8 Conclusion

We have presented a method that can infer a model of a non-resettable black box FSM for which we know an upper bound n on the number of states and a characterizing set W. The method is polynomial in n; the degree of the polynomial is bounded by the cardinality of W. It has now been implemented, and we are investigating its effectiveness under various settings. We are in particular interested in assessing the average complexity for various cases of characterization sets.

It might also be interesting to see how the algorithm proposed here could benefit from the improvements that have been proposed in the related problem of conformance testing for non-resettable machines. As mentioned in several parts of the paper, the algorithm lends itself to optimizations that could reduce the length of the sequence of inputs. In particular, the number of iterations in the localizer could be reduced when short cycles are recognized.

Another direction for research is on relaxing the assumptions, and providing adaptive heuristics. One direction could be to work towards inference of quotients [12].

Acknowledgments. The authors acknowledge the work of Laurent Anadon, an intern from Ecole Polytechnique with whom we started this problem on the DS case. The implementation of the algorithm was carried out by Nicolas Brémond, an intern from Enseirb-Matmeca. A. Simao had financial support of Brazilian Funding Agencies, FAPESP and CNPq. A. Petrenko acknowledges financial support of NSERC via Discovery grant RGPIN/194381-2012.

References

1. Aarts, F., Jonsson, B., Uijen, J.: Generating models of infinite-state communication protocols using regular inference with abstraction. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 188–204. Springer, Heidelberg (2010)
2. Ammons, G., Bodik, R., Larus, J.: Mining specifications. In: POPL 2002, pp. 4–16 (2002)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **2**, 87–106 (1987)
4. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: ESEC/FSE 2009, pp. 141–150 (2009)
5. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Laubach, S., Pasareanu, C.S., Robby, Zheng, H.: Bandera: extracting finite-state models from Java source code. In: 22nd ICSE, pp. 439–448 (2000)
6. Hennie, F.C.: Fault-detecting experiments for sequential circuits. In: Proceedings of Fifth Annual Symposium On Circuit Theory and Logical Design, pp. 95–110 (1965)
7. Irfan, M.N., Oriat, C., Groz, R.: Angluin style finite state machine inference with non-optimal counterexamples. In: Workshop on Model Inference In Testing 2010, ISSTA, 11–19 (2010)
8. Jourdan, G.V., Ural, H., Yenigün, H.: Reduced checking sequences using unreliable reset. Inf. Process. Lett. **115**(5), 532–535 (2015)
9. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - a survey. Proc. IEEE **84**, 1090–1123 (1996)

10. Meinke, K.: CGE: a sequential learning algorithm for Mealy automata. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 148–162. Springer, Heidelberg (2010)
11. Mihancea, P.F., Minea, M.: jModex: model extraction for verifying security properties of web applications. In: CSMR-WCRE, pp. 450–453 (2014)
12. Petrenko, A., Li, K., Groz, R., Hossen, K., Oriat, C.: Inferring approximated models for systems engineering. In: HASE 2014, pp 249–253 (2014)
13. Porto, F.R., Endo, A.T., Simao, A.: Generation of checking sequences using identification sets. In: Groves, L., Sun, J. (eds.) ICFEM 2013. LNCS, vol. 8144, pp. 115–130. Springer, Heidelberg (2013)
14. Rezaki, A., Ural, H.: Construction of checking sequences based on characterization sets. Comput. Commun. **18**(12), 911–920 (1995)
15. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. In: Hanson, S.J., Remmele, W., Rivest, R.L. (eds.) Machine Learning: From Theory to Applications. LNCS, vol. 661, pp. 51–73. Springer, Heidelberg (1993)
16. Shahbaz, M., Groz, R.: Inferring Mealy machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 207–222. Springer, Heidelberg (2009)
17. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 256–296. Springer, Heidelberg (2011)
18. Vasilevskii, M.P.: Failure diagnosis of automata. Cybernetics **9**, 653–665 (1973)