JMLR: Workshop and Conference Proceedings 57:30–41, 2016

Inferring Non-resettable Mealy Machines with n States

Roland Groz ROLAND.GROZ@IMAG.FR

Catherine Oriat CATHERINE.ORIAT@IMAG.FR

Nicolas Brémond NBREMOND@ENSEIRB-MATMECA.FR

LIG
Université Grenoble Alpes
F-38000 Grenoble, France

Abstract

Automata inference algorithms are used to extract behavioural models of software components. However, when the software system cannot be reset, inference must be done from a single trace. This paper proposes an active learning algorithm that can infer a Mealy model under the assumption that the number of the states of the machine is known and that a characterization set for it is provided. This algorithm improves on a previous paper that used a looser assumption on the number of states. The complexity is polynomial in the number of states of the Mealy machine.

Keywords: Query learning, FSM testing, Software Engineering

1. Introduction

Query learning has received growing interest for software engineering. It is used to retrieve finite state models of software systems or components. In this context, it makes sense to use active learning inference algorithms, because in most cases we can query the software system, by sending inputs to observe its outputs. The corresponding automata models are input-output machines, that is Mealy machines, often called FSM (Finite State Machines). Typical queries are usually not membership queries (checking whether a given input sequence is accepted), but output queries: sending sequences of inputs and observing the corresponding sequence of outputs. So the inference algorithm will exploit traces: sequences of input/output pairs.

Most algorithms used so far have considered that the system that is inferred can be reliably reset, so that it is possible to root the observed traces to a fixed known initial state. However, this is questionable. In many black box contexts, typically when a system is queried over a network, the system under inference (SUI) cannot be reset. It may also be the case that a system could be reset, but that it could take a very long time to restart. As a typical example, interacting over a local network for querying a web system on a virtual machine takes around a millisecond for a single input/output observation, whereas resetting a virtual machine takes typically almost over a minute, so the reset typically costs 10^5 more time than an I/O observation.

Rivest and Schapire (1993) had addressed this problem using a variant of the L^* algorithm with the assumption that a homing sequence was given, that is a fixed input sequence

© 2016 R. Groz, C. Oriat & N. Brémond.

such that the output observed completely determines the state reached at the end of the sequence. We have proposed a new approach in Groz et al. (2015). Instead of relying on a homing sequence, we use two classical assumptions from FSM testing. First, we assume a bound on the number of states of the SUI. And instead of a homing sequence, we start from a given characterization set, a.k.a. W-set, in reference to Vasilievskii (1973). A W-set is a set of input sequences that, when applied from a given state, make it possible to know what is this state. The advantage as compared to Rivest's approach is that this algorithm no longer requires an oracle. Implementing an oracle that can answer equivalence queries cannot be done for an unknown software system, so it is usually approximated. However there is a difficulty: since the W-set could contain several sequences, it implies that for a system which cannot be reset, the algorithm must be sure to return to a state where a previous sequence was applied, from the same state. This is achieved by a recursive procedure called a localizer.

Of course, the absence of an oracle is compensated by the fact that we assume we know a bound on the number of states as well as a characterization set (although the latter is a weaker compensation for a homing sequence). Such knowledge could be derived for instance from previous versions of the component to be inferred. Indeed, a frequent case is to use inference to get uptodate models of evolving components. In Groz et al. (2015) we discuss possible workaround to address inaccurate bounds or incorrect characterization sets. One possibility, as in Rivest and Schapire (1993) is to go for probabilistic algorithms, i.e. extending the characterization set (just as the homing sequence can be extended). An inadequate bound on the number of states could be spotted in the localizer procedure, and increased accordingly.

In this paper, we show how the algorithm from Groz et al. (2015) can be improved and can converge with a shorter observation trace when the assumption is that we have a strict bound, i.e. we are given the number of states of the machine to be inferred. It is based on the notion of compatibility, that has been used for a long time in passive inference, for instance for the use of automata minimization as done by Kella (1971). It is also possible to reduce the length of the sequence in the localizer procedure.

The rest of this paper is organized as follows. In section 2, we give the formal notations and definitions used in the paper to describe the algorithms. In section 3, we recall what is the base algorithm for the general case defined in Groz et al. (2015). Section 4 describes the improvements that can be done when the number of states if known. In section 5 we show how the algorithm works on an example, on which it reduces the length of the sequence necessary to infer from 71 steps (base algorithm) down to only 41 steps with the new algorithm. The following section provides preliminary statistical results on various kinds of machines. Finally, section 7 concludes.

2. Definitions

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

- Q is a finite set of states with the initial state q_0,

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

Notations δ and λ are lifted to sequences: $\delta(q, \epsilon) = q$, $\lambda(q, \epsilon) = \epsilon$ and for $q \in Q$, for $\alpha x \in I^*$, $\delta(q, \alpha x) = \delta(\delta(q, \alpha), x)$ and $\lambda(q, \alpha x) = \lambda(q, \alpha)\lambda(\delta(q, \alpha), x)$.

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

Two states $q, q' \in Q$ are *distinguishable* by $\gamma \in I^*$ if $\lambda(q, \gamma) \neq \lambda(q', \gamma)$. Two states are distinguishable by a set $H \subset I^*$ if there exists $\gamma \in H$ that distinguishes them. An FSM is *minimal* if all states are pairwise distinguishable. A set W of sequences of inputs (therefore conventionally called a W-set, following Vasilievskii (1973)) is a *characterization set* for an FSM M if each pair of states is distinguishable by W.

A sequence of input/output pairs $\alpha \in (IO)^*$ is called a *trace*. Given a trace composed of a prefix and suffix $\omega = \alpha\beta$, we will write $\alpha = \omega \setminus \beta$ and $\alpha \leq \omega$. $\omega \downarrow I$ will denote the projection of the trace on its input set, that is the sequence obtained by deleting all outputs from the trace. Given a machine $M = (Q, q_0, I, O, \delta, \lambda)$ and its current state q defined by the context, $tr(\alpha)$ will denote the trace from q such that $tr(\alpha) \downarrow I = \alpha$ and $tr(\alpha) \downarrow O = \lambda(q, \alpha)$. For a set of input sequences H, $Tr(H) = \{tr(h) \mid h \in H\}$. Finally, we overload the notation by defining $Tr(q)$ as the set of all traces of M from state q, i.e. $Tr(q) = \{tr(\alpha)\}_{\alpha \in I^*}$. And $Tr(M) = Tr(q_0)$. Two machines are equivalent: $M \approx M'$ iff $Tr(M) = Tr(M')$.

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

Given an FSM M and a set of possible implementations (aka fault domain) $\mathcal{F}$, a trace $\mathcal{T}$ is a *checking sequence* for M in $\mathcal{F}$ iff $\forall M' \in \mathcal{F}, \mathcal{T} \in Tr(M') \Leftrightarrow M' \approx M$. A typical fault domain could be the set of all FSM with a number of states bounded by some N. A checking sequence is a perfect test that identifies a correct machine and detects all faulty implementations.

3. Base algorithm

We assume we are provided with a Black-Box FSM $B = (P, p_0, I, O, \delta, \lambda)$, and a characterization set for it $W \subset I^*$. We are also provided with a bound n on the number of states of B, so $card(P) \leq n$. ω will be the currently observed trace.

Since we do not know B, we cannot compute a checking sequence for it using e.g. Vasilievskii's W-method (Vasilievskii, 1973). However, we can learn the structure of a minimal FSM equivalent to B using the algorithm described in Groz et al. (2015). This algorithm will in effect compute a checking sequence that will identify B in the fault domain of all machines with up to n states.

A complete definition and proof of the algorithm can be found in the original paper (Groz et al., 2015). In this section, we just provide the main description and results, on which the new algorithm, that will be presented in the next section, is based.

3.1. Data records used by the algorithm

The algorithm will record deduced information in the following sets:

- $Q \subset 2^{W \mapsto (IO)^*}$ will denote states, defined by their characterization. Each state is named by its traces recording its responses to the input sequences from W.
- $C : (IO)^* \mapsto Q \cup \{\bot\}$, actually defined on a subset of prefixes of ω will be used to label prefixes of the observed trace. For $\alpha \leq \omega$, we have $C(\alpha) = q \in Q$ when we have established that the machine B was in state q after observing α. $C(\alpha) = \bot$ when we do not know what is the state reached by B after α. C stands for "Characterized positions".
- $V \subset Q \times (IO)^* \times Q$ will record verified subtraces of ω, that is a subtrace is in V if its start and end "states" are labelled by C. Formally, $V' = \{(q, \alpha, q') \mid \exists \sigma\alpha \leq \omega, C(\sigma) = q, C(\sigma\alpha) = q'\}$. Actually, since the relation between states associated with V' would be transitively closed, the algorithm will maintain its transitive reduction V. V stands for "Verified sequences".
- R will record input verified transitions, those for which the start state is known but not necessarily the end state. $R = \{(q, x) \in Q \times I \mid \exists o \in O, \exists q' \in Q, (q, xo, q') \in V\}$.
- $K \subset Q \times (IO)^+ \times (IO)^*$ keeps track of the applications of elements from W in a given state, followed by either a single transition or a trace of a sequence from W. $(q, \alpha, \gamma) \in K$ if $\exists \beta$ s.t. $\beta\alpha\gamma \leq \omega$, $C(\beta) = q, \alpha \downarrow I \in I \cup W$ and $\gamma \downarrow I \in W$. K stands for "Known applications of characterizing sequences".

3.2. Localizer procedure

The key element for the base algorithm is the localizer procedure, which makes it possible to make sure that we can bring the machine to a state that can be identified because we can be sure of its responses to all sequences in W. The localizer procedure is formulated for fixed input and output alphabets I and O, and an assumed bound n on the number of states in the black box. It takes as input the current observed trace ω and W. ω will be extended and updated as output when exiting the procedure.

The structure of the sequence produced by a localizer is out of scope of this paper, and we refer the reader to Groz et al. (2015) for a presentation of a general localizer.

The key property of a localizer is that, regardless of the state in which the black-box machine B is, the localizer will ensure that just before applying the last w_k input sequence, B was in a state that can be characterized by the traces returned as second argument. This is captured by the following theorem from Groz et al. (2015).

Theorem 1 *From any starting trace ω, the localizer $L(\omega, W)$ will return (ω', X, β) such that $C(\omega' \setminus \beta) = X$ and $X \downarrow I = W$.*

In the example of section 5, we just need the simple case where the set W is reduced to a single element $W = \{w\}$, in which case we can apply w just once to know what is the state reached at the end of a trace (just before we apply w). However, designing a localizer when W has more than one element is much more complex. For instance, for a set $W = \{w_1, w_2, w_3\}$ and a bound N on the number of states, a localizer could apply the following sequence: $(w_1^{2N-1}w_2)^{2N-1}w_1^{2N-1}w_3$.

3.3. Main procedure

The inference procedure assumes it is given a black-box machine B whose number of states is less or equal to n, and for which it is known that a set which we will order as $W = (w_1, ...w_p)$ is a characterization set. It can query B through the procedure Apply by submitting an input sequence and getting the corresponding output sequence from B, in the state it was left in following previous queries. The rules for Update can be found in Groz et al. (2015).

```
procedure InferNoReset($B, W, n$)
    Initialize $K = R = V = \emptyset$  $(\omega, q_0, tr(w_p) \leftarrow L(\epsilon, W)$      // Home into a known state
    $C(\omega \setminus tr(w_p) \leftarrow q_0$  $Q_C \leftarrow \{q_0\}$  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 known, move to unverified transition
            Find a shortest $\alpha = \alpha_1, \alpha_2, ...\alpha_k$  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 $\alpha$  $\omega \leftarrow \omega\alpha$ ; $\chi \leftarrow \omega$  Apply $x$,
            observe $xo$                                        // Observe transition
            $\sigma \leftarrow xo$ ; $\omega \leftarrow \omega xo$
        else
                                        // Use the latest known previous state
            Find the shortest $\gamma$ s.t. $C(\omega \setminus \gamma) \neq \bot$        // Could be shorter than $w_p$
            $\chi \leftarrow \omega \setminus \gamma$ ; $\sigma \leftarrow \gamma$
        end
        $q \leftarrow C(\chi)$                                    // 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 $(q, \sigma)$
        $\omega \leftarrow \omega tr(w)$ ; $K \leftarrow K \cup \{(q, \sigma, tr(w))\}$  if $\{w \in W \mid (q, \sigma, tr(w)) \in K\} = W$ then
                                        // Full characterization reached
            $C(\chi\sigma) \leftarrow \{tr(w) \mid w \in W$ and $(q, \sigma, tr(w)) \in K\}$  $Q_C \leftarrow Q_C \cup \{C(\chi\sigma)\}$  Update
            $V, R, K, C$
        end
        if $C(\omega) = \bot$ then
                                        // Move to a characterized state
            $(\omega, q', tr(w_p)) \leftarrow L(\omega, W)$  $C(\omega \setminus tr(w_p)) \leftarrow q'$ ; $Q_C \leftarrow Q_C \cup \{q'\}$  Update
            $V, R, K, C$
        end
    end
    Build the conjecture from $Q_C$ and $V \cap (Q \times (IO) \times Q)$
end
```

We have kept the traditional terminology of "conjecture", even though in the case of this algorithm under our assumptions, we only get a single final machine.

3.4. Convergence and Complexity

The initial paper Groz et al. (2015) proved the following theorem and analyzed the complexity of the base algorithm.

Theorem 2 *When W is a characterization set for B and the number of states of B is less than or equal to n, the inference procedure terminates and yields a conjecture that is isomorphic to the minimal FSM modelling B.*

The complexity of active learning algorithms based on queries is usually assessed in terms of the number and the length of queries. This makes sense in particular in the context of learning black-box software components, especially when queried over a network or a bus, because remote interaction with the system takes much more time than the internal bookkeeping activities of the algorithm. In the case of algorithms that do not use reset and simply send inputs and observe outputs, the measure is quite simple: the total length of the trace until we can build the conjecture.

With $p = card(W)$, n the number of states and f the number of input symbols, a very coarse bound for the length of the trace is $O(p(f + p)2^p n^{p+2})$. Experiments showed that the average complexity for $p = 2$ (the most common case even for random machines with 2 inputs and 2 outputs) is $O((f + 2)n^{1.9})$. The interesting point worth mentioning is that the algorithm remains polynomial in n, although the degree of the polynomial depends on the number of sequences in W. Experiments with random and semi-random machines also showed that this algorithm outperforms Rivest and Schapire by an order of magnitude for $p = 2$.

4. Algorithm for known number of states

The base algorithm was designed with the view that n was an upper bound on the number of states. If n is actually the number of states, $card(P) = n$, the length of the inference sequence can be reduced, once we have found the characterization of all states. Note that state discovery can occur either through applications of the localizer, or through the deductions made from the set K of known applications of characterizing sequences.

From the outgoing sequences from a given state recorded in V we can check whether a given tail state (as yet unidentified) of a transition is compatible with any of the other states. If it is compatible with just one, and all the states are known, then we can identify that tail state. We detail this now.[1]

1. Actually, there could also be a reduction in the localizer procedure. Once we know all answers of all states to W, we could use an adaptive localizer that applies the most discriminating sequences from W first, and further ones only if needed. However, we do not address localizers in this paper, and in fact, the gain in our experiments was null or very small, because the deductions based on compatibility are strong enough to reduce drastically the need to call the localizer once all states are discovered.

4.1. Compatibility relation

For each sequence $(q, \alpha, q') \in V$, and each prefix $\sigma \in \mathit{Pref}(\alpha)$ of α, we define a node denoted by (q, σ). This node is associated with the state reached in B when σ is applied (or observed) from state q. Each node may correspond to several prefixes of ω, because the same verified subtrace α starting from a prefix labelled with q can occur several times in ω, but also because two verified sequences from V can share a common prefix, for instance if we have $(q, \alpha\beta, q') \in V$ and $(q, \alpha\gamma, q'') \in V$. Apart from nodes that appear in V, we also consider nodes in the tail of the observed trace, that are successors of the last labelled subtrace of the observed trace ω. The set of nodes is:

$$S = \{(q, \sigma) \mid q \in Q \;\&\; \exists \alpha \in (IO)^* \text{ s.t. } \alpha\sigma \leq \omega,\, C(\alpha) = q \;\&\; \forall \tau, \epsilon < \tau < \sigma \Rightarrow C(\alpha\tau) = \bot\}$$

Actually, when $(q, \alpha, q') \in V$, we consider in S that (q', ϵ) and (q, α) are two notations for the same node. So S is in fact the quotient for this equivalence relation. Nodes of S that are of the form (q, ϵ) will be called state nodes, and we may denote them just q. We organize V as a labelled input output transition system, viz. a directed graph with edges labelled by a couple of input and output. $\Gamma = (S, I, O, \rightarrow)$. The transition relation is defined as

$$(q, \sigma) \overset{x/o}{\rightarrow} (q, \sigma x o)$$

Nodes in Γ correspond to states of B, and edges represent the existence of a transition that has been traversed while observing ω.

We can now define the incompatibility relation $\nsim$ as the smallest fixpoint for the following relation.

$$(q, \epsilon) \nsim (q', \epsilon) \text{ iff } q \neq q' \tag{1}$$

$$(q, \sigma) \nsim (q', \sigma') \text{ if } \exists x \in I, (q, \sigma) \overset{x/o}{\rightarrow} (q, \sigma x o), (q', \sigma') \overset{x/o'}{\rightarrow} (q', \sigma' x o') \;\&\; o \neq o' \tag{2}$$

$$(q, \sigma) \nsim (q', \sigma') \text{ if } \exists x \in I, (q, \sigma) \overset{x/o}{\rightarrow} (q, \sigma x o), (q', \sigma') \overset{x/o}{\rightarrow} (q', \sigma' x o) \;\&\; (q, \sigma x o) \nsim (q', \sigma' x o) \tag{3}$$

The interpretation of this incompatibility w.r.t the states of B that correspond to the nodes of Γ is straightforward. Nodes (q, σ) and (q', σ') are incompatible when there is a sequence that distinguishes them, so they must be associated with different states of B. This is the stepping stone for refining the state identification. With the base algorithm, in order to identify the tail state of a transition, we need to come back to that transition as many times as there are sequences in W, and apply in turn each input sequence from W to identify the state. Whereas with the incompatibility relation, we may realize that we have enough information in the observed trace to identify the tail state without having to apply all elements of W to it.

4.2. Learning algorithm with compatibility analysis

Compatibility analysis can be weaved into the base algorithm as follows.

- A new data structure G based on Γ, which is the labelled transition system defined in Section 4.1 , enriches the set V. It supersedes V since it contains all the sequences from V plus additional nodes (those that occur after the last characterized trace) and additional information. G tags each node with the set of state nodes that are incompatible with it. Formally $G = (\Gamma, D) = ((S, I, O, \rightarrow), D)$ where $D \subset (2^Q)^S$ expresses the incompatibility relation with Q (D stands for difference).

- Whenever the observation trace ω is extended, G can be updated using the rules described above in 4.1. This occurs either when the localizer is used, because it consists of repeated applications of elements of W or whenever the algorithm applies an input x or a sequence α of w.
- G being based on V is also updated whenever a new state is discovered, either through the localizer or through further deductions either from K, or from the incompatibility relation.

Deduction: Whenever we update the incompatibility relation, one of the nodes in S may become incompatible wih all but one state from Q. When this occurs, C is updated as well as the depending structures: G, R, K, C.

With respect to the text of the base algorithm, this means that the only changes will occur in the Apply and Update procedures: both will update G with the extra rules from 4.1, and will check whether a **deduction** is possible. As soon as transitions are completely defined, for all states in Q and inputs in I, the algorithm can exit from the **while** loop and build the conjecture.

5. Example

We consider the following automaton, which is a product of a counter modulo 2 and a counter modulo 3 (fig. 1). $I = \{i, j\}$ and $O = \{0, 1\}$. We choose as W-set the singleton $W = \{jji\}$, which is a distinguishing sequence.

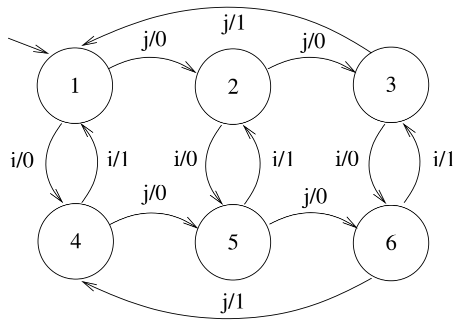

Figure 1: Two-counters automaton

5.1. Inferring until we “learn” all states

We start the algorithm by applying the localizer, which consists in just applying the sequence jji. We obtain the output sequence 000, and thus identify a new state at position 0 of the trace: $S_0 = \{jji \mapsto j0j0i0\}$ which we shall simply write $\{j0j0i0\}$. Positions are shown as indices of the states reached by the automaton. Of course, the inference algorithm only observes the positions and I/O pairs of transitions, it does not have access to the internal state: those are depicted for the reader to make it easier to follow the trace. The states that can be inferred are shown below the positions. They are called S_i with a numbering i that starts from 0.

$$\begin{array}{cccccccccc} 1_0 \xrightarrow{j/0} & 2_1 \xrightarrow{j/0} & 3_2 \xrightarrow{i/0} & 6_3 \xrightarrow{j/1} & 4_4 \xrightarrow{j/0} & 5_5 \xrightarrow{i/1} & 2_6 \xrightarrow{j/0} & 3_7 \xrightarrow{j/1} & 1_8 \xrightarrow{i/0} & 4_9 \\ S_0 & & & S_1 & & & S_2 & & & \end{array}$$

We enter the main *while loop*. We do not know the current state, so we enter the *else* part of the first *if* statement and get $\sigma = j0j0i0$. We choose $w = jji$, apply w and observe 101. So we get $K = \{(S_0, j0j0i0, j1j0i1)\}$ and discover a new state $S_1 = \{j1j0i1\}$ and get $V = \{(S_0, j0j0i0, S_1)\}$. Note that states are not discovered in the order of their "names" in the automaton. We first found state 1 which we called S_0, then we discover state 6 which we call S_1, and $V = \{(S_0, j0j0i0, S_1)$. We have the following incompatibilities: $S_0 \not\approx S_1$, $1 \not\approx S_1$, $4 \not\approx S_1$. Now, as the last position is unknown, we enter the last *if* statement and apply the localizer procedure.

We get $K = \{(S_1, j1j0i1, j0j1i0)\}$. A new state $S_2 = \{j0j1i0\}$ is discovered, $V = \{(S_0, j0j0i0, S_1), (S_1, j1j0i1, S_2)\}$ and we label position 6 of the trace with S_2. We can deduce that $6 \not\approx S_0$, $6 \not\approx S_1$ and $7 \not\approx S_0$, as well as a few other incompatibilities.

We restart the while loop and obtain the trace

$$\begin{array}{cccccccccc} 4_9 \xrightarrow{j/0} & 5_{10} \xrightarrow{j/0} & 6_{11} \xrightarrow{i/1} & 3_{12} \xrightarrow{j/1} & 1_{13} \xrightarrow{j/0} & 2_{14} \xrightarrow{i/0} & 5_{15} \xrightarrow{j/0} & 6_{16} \xrightarrow{j/1} & 4_{17} \xrightarrow{i/1} & 1_{18} \\ S_3 & & & S_4 & & & S_5 & & & \end{array}$$

At this point, we have identified the states $S_3 = \{j0j0i1\}$, $S_4 = \{j1j0i0\}$ and $S_5 = \{j0j1i1\}$. $V = \{(S_0, j0j0i0, S_1), (S_1, j1j0i1, S_2), (S_2, j0j1i0, S_3), (S_3, j0j0i1, S_4), (S_4, j1j0i0, S_5)\}$.

We can note that in this particular example, we have first identified all states and then we will identify all transitions, which is not the case in general. Although we were able to get partial incompatibility results, we are not yet able to use them for labelling positions, at this point, this will come later.

5.2. Learning transitions

We still do not know where we are, so we again apply the localizer.

$$\begin{array}{cccccccc} 1_{18} \xrightarrow{j/0} & 2_{19} \xrightarrow{j/0} & 3_{20} \xrightarrow{i/0} & 6_{21} \xrightarrow{i/1} & 3_{22} \xrightarrow{j/1} & 1_{23} \xrightarrow{j/0} & 2_{24} \xrightarrow{i/0} & 5_{25} \\ S_0 & & & S_1 & S_4 & & & S_5 \end{array}$$

We can now label positions 18 with S_0 (as returned by the localizer) and 21 with S_1 (because $(S_0, j0j0i0, S_1) \in V$). We can notice that for the first time, we know where we are (state S_1) at the end of the trace. We thus enter the *then* part of the first *if* statement in order to move to an unverified transition. We choose $\alpha = \epsilon$ and $x = i$, apply x and observe 1. At step 22, we choose $w = jji$, apply w and observe 100. We can now label step 22 with S_4. We have thus identified the transition $(S_1, i1, S_4)$. And we know from V that position 25 is S_5.

$$\begin{tikzcd}
5_{25} \arrow[r, "i/1"] & 2_{26} \arrow[r, "j/0"] & 3_{27} \arrow[r, "j/1"] & 1_{28} \arrow[r, "i/0"] & 4_{29} \arrow[r, "i/1"] & 1_{30} \arrow[r, "j/0"] & 2_{31} \arrow[r, "j/0"] & 3_{32} \arrow[r, "i/0"] & 6_{33} \\
S_5 & S_2 & & & S_3 & S_0 & & & S_1
\end{tikzcd}$$

So we now learn a new transition from S_5; choosing $\alpha = \epsilon$ and $x = i$, we get $(S_5, i1, S_2) \in V$, and similarly after another 4 inputs we learn transition $(S_3, i0, S_0)$. At this point, $V = \{(S_0, j0j0i0, S_1), (S_1, j1j0i1, S_2), (S_1, i1, S_4), (S_2, j0j1i0, S_3), (S_3, j0j0i1, S_4), (S_3, i1, S_0), (S_4, j1j0i0, S_5), (S_5, j0j1i1, S_0), (S_5, i1, S_2)\}$, and we cannot make any deduction from incompatibility. We are in state S_1 (position 33), and we already know the i transition, so we choose $\alpha = \epsilon$ and $x = j$, and we apply jji. So we learn transition $(S_1, j1, S_3)$.

$$\begin{tikzcd}
6_{33} \arrow[r, "j/1"] & 4_{34} \arrow[r, "j/0"] & 5_{35} \arrow[r, "j/0"] & 6_{36} \arrow[r, "i/1"] & 3_{37} \\
S_1 & S_3 & & & S_4
\end{tikzcd}$$

5.3. Deductions come in

We have now come to the point where the new algorithm from section 4 will make a difference, because *we can make a deduction* from G. Since we learnt a transition on input j, we are now able to split in V the verified sequences that comes from the localizing sequence $(S_1, j1j0i1, S_2)$ into two subsequences $(S_1, j1, S_3)$ and $(S_3, j0i1, S_2)$. And from this, we find that in Γ the node $(S_2, j0)$, that admits trace $j1i0$ (see position 7), is incompatible with states S_0, S_2, S_3, S_5 and now S_1 (S_1 admits trace $j1i1$). Therefore, we are able to deduce a new transition $(S_2, j0, S_4)$. In turn, this make it possible to split in V the sequence that went from S_2 to S_3, so that now we have $(S_4, j1i0, S_3) \in V$. From this, we now deduce that $(S_5, j0) \nsim \{S_0, S_2, S_3, S_4, S_5\}$, hence we deduce transition $(S_5, j0, S_1)$.

No further deduction can be made, so we continue the main loop from S_4 (position 37).

$$\begin{tikzcd}
3_{37} \arrow[r, "i/0"] & 6_{38} \arrow[r, "j/1"] & 4_{39} \arrow[r, "j/0"] & 5_{40} \arrow[r, "i/1"] & 2_{41} \\
S_4 & S_1 & S_3 & & S_2
\end{tikzcd}$$

We move to an unverified transition, with $\alpha = \epsilon$, $x = i$ and get $\sigma = i0$. We take $w = jji$, apply jji, observe 101 and reach position 41, so we just learnt transition $(S_4, i0, S_1)$. We have now enough information to conclude. $(S_4, j1)$ is incompatible with all states except S_0, so we learn transition $(S_4, j1, S_0)$, which enables to split the previous sequence $(S_4, j1j0i0, S_5)$, so that with incompatibility we deduce transitions $(S_3, j0, S_5)$, $(S_0, j0, S_2)$ and $(S_0, i0, S_3)$. At this point, $V = \{(S_0, i0, S_3), (S_0, j0, S_2), (S_1, i1, S_4), (S_1, j1, S_3), (S_2, j0, S_4), (S_3, i1, S_0), (S_3, j0i1, S_2), (S_3, j0j0i1, S_4)(S_4, i0, S_1), (S_4, j1, S_0), (S_5, i1, S_2), (S_5, j0, S_1)\}$. We only need to learn transition by i from S_2 and the tail state for $(S_3, j0)$. The second one is easily obtained by incompatibility: $(S_3, j0, S_5)$. And for the first one, it comes from the fact that we knew $(S_4, j1j0i0, S_5)$, which has been split in Γ as $(S_4, j1, S_0)$ and $(S_0, j0, S_2)$, hence $(S_2, j0, S_5)$. We have now learnt all transitions, and the algorithm stops on a conjecture that is equivalent to the black box machine.

Therefore, our algorithm is able to infer the automaton with a trace of length 41, whereas the base algorithm required 71 steps.

Would it be possible to infer the automaton earlier? It can be shown that in this case, the prefix of length 39 is a checking sequence, whereas for a prefix of length 38, there would not be a single solution. Actually, we applied jji from position 38, but the first j would have been enough; however the argument to conclude two steps earlier than we did uses more than incompatibility: it requires a specific reasoning on the number of outgoing and ingoing $i-$transitions.

6. Preliminary results

Figure 2 shows the median of the trace length outside localizers[2] as a function of the number of states for both algorithms. We used small input and output sets (2 or 3 letters) as this is the toughest case (lower distinguishability).

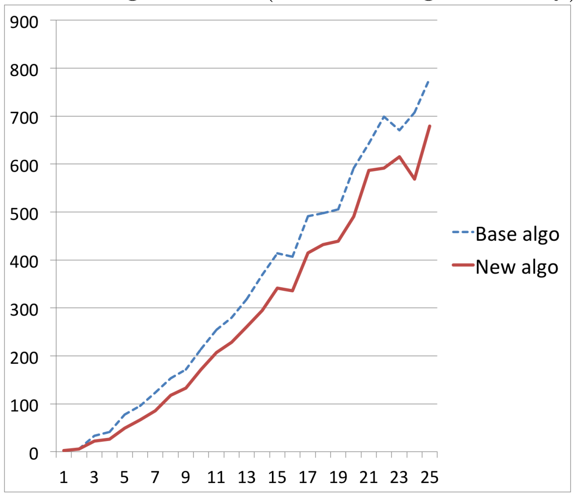

Figure 2

We conducted an experiment with 300 random FSM, with 2 to 25 states and $card(W) = 2$ (the most common case for random machines with small input and output sets, here 2 or 3 letters).
The new algorithm improves on the base algorithm on average by 20% (N.B. including general localizers from Groz et al. (2015), the overall gain is 8.1%). The gain seems stable over various state sizes. Another experiment with machines which are a product of a 3 state counter (as in the example) and a random FSM showed better results (gain around 30%).

The most striking result however is that the trace found with the new algorithm often corresponds or is close to the smallest prefix of the trace found with the base algorithm that would be a checking sequence. *In over 50% of experiments, the new algorithm actually finds exactly the smallest prefix. Any shorter prefix would not identify a unique machine.*

7. Conclusion

This paper shows that by better exploiting the information available from a trace built by the active learner for a non-resettable machine with a characterization set, we can reduce the length of the sequence needed for complete inference of an FSM provided the number of its states is known. Furthermore, the incompatibility relation defined is almost optimal in the sense that it often yields the shortest prefix that identifies a single solution machine.

Incompatibility information could also be used when the number of states is not known, to distinguish states, and get a minimal number of states, even though they might not be fully characterized. We are also working on other directions for better applicability of the algorithms presented in this paper. As noted in Groz et al. (2015), there are relatively easy

2. We excluded localizing sequences from this count, because the compatibility relations do not change the localizer, and there could be other types of localizers. Our implementation works with the general localizer from Groz et al. (2015), but more efficient localizers have been considered. Since the length of localizing sequences is the major contributor to the length of the trace, taking a non optimized localizer into account reduces the significance of the results.

approaches to deal with an incorrect bound on the number of states of a system. Most interesting would be the possibility to alleviate the other key assumption: the hypothesis that we know a characterization set of a black box FSM. We are considering heuristics to derive characterization sets for unknown machines.

The assumptions that we know n and W for a black box machine are strong. Under those assumptions, we propose algorithms that directly end up with a correct identification of the machine. Just as other grammatical inference methods, they could form the basis for deriving methods that converge in the limit, by establishing converging conjectures that can be refined with e.g. counterexamples that can then provide new values for n or W.

Acknowledgments

The authors are grateful to Adenilso Simao for providing the tool `Chico` based on Simao and Petrenko (2010) that checks whether a sequence is a checking sequence for a given machine.

References

R. Groz, A. Simao, A. Petrenko, and C. Oriat. Inferring finite state machines without reset using state identification sequences. In *Proceedings of the International Conference on Testing Software and Systems, ICTSS 2015*, Dubai, nov 2015.

J. Kella. Sequential machine identification. *IEEE Trans. Comput.*, 20(3), 1971.

R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. In *Machine Learning: From Theory to Applications*, pages 51–73, 1993.

A. Simao and A. Petrenko. Checking completeness of tests for finite state machines. *IEEE Trans. on Computers*, 59(8):1023–1032, 2010.

M. P. Vasilievskii. Failure diagnosis of automata. *Cybernetics*, 9:653–665, 1973.