FSM Inference from Long Traces

Florent Avellaneda and Alexandre Petrenko

CRIM, Montral, Canada,
{florent.avellaneda, alexandre.petrenko}@crim.ca

Abstract. Inferring a minimal finite state machine (FSM) from a given set of traces is a fundamental problem in computer science. Although the problem is known to be NP-complete, it can be solved efficiently with SAT solvers when the given set of traces is relatively small. On the other hand, to infer an FSM equivalent to a machine which generates traces, the set of traces should be sufficiently representative and hence large. However, the existing SAT-based inference techniques do not scale well when the length and number of traces increase. In this paper, we propose a novel approach which processes lengthy traces incrementally. The experimental results indicate that it scales sufficiently well and time it takes grows slowly with the size of traces.

Keywords: Machine inference, Machine identification, SAT solver

1 Introduction

Occam's razor is a problem-solving principle attributed to William of Ockham. Also known as the law of parsimony, this principle states that among competing hypotheses, the one with the fewest assumptions should be selected. This simple and natural principle is the base of a lot of work in various areas.

A typical area where this principle is used is the model inference problem. Model inference is the process of building a model consistent with a given set of observations. Since there exists generally an infinite number of consistent models, we choose the simplest following the law of parsimony. When the model to infer is a finite states machine (FSM), we generally use the number of states as the unit of measurement for the complexity. So, the inference in this context consists in finding a minimal FSM consistent with a given set of observation.

Model inference problem has several useful applications such as model-based testing when a model inferred from traces produced by a system executing tests is used to assess the test quality, generate additional tests and model check properties confirmed by the executed tests. The FSM inference from a set of traces is a very active research domain which can be divided into two categories: passive learning (learning from examples) [12,13] and active learning (learning with queries) [4]. In the first category, we have only a set of examples and use it to infer an FSM consistent with this set. Passive FSM inference problem is stated by Kella in 1971 [16] as sequential machine identification. In the second category, we use an oracle to ask queries and infer FSM incrementally. The work

by Walkinshaw et al. [22] has shown how passive inference algorithms can be used to perform active inference. Based on these results, Smetsers et al. [19] employ SMT solvers to infer DFAs, Mealy machines and register automata.

This paper belongs to the first category: "Given a sample $\mathcal{T}$ and $n \in \mathbb{N}$, does an FSM with n states consistent with $\mathcal{T}$ exist?". Bierman and Feldman address this question [7] by proposing to use a CSP (constraint satisfaction problem) formulation. Later Gold [13] proves that the problem is NP-complete. More than 20 years later Oliveira and Silva [17] develop an algorithm using generic CSP or SMT solvers. Then Grinchtein, Leucker and Piterman [14] present a SAT formulation which allows to solve the problem more efficiently, later enhanced by Heule and Verwer [15] by adding auxiliary variables. The method of Grinchtein, Leucker and Piterman is less efficient than that of Heule and Verwer, but their incremental approach is interesting, as the time it takes to find a solution grows slowly with the length of a given trace.

It combines the algorithms of Angluin [4] and Biermann and Feldman [7]. As a result, SAT clauses need to be completely rewritten each time new tables as proposed by Angluin are modified. The approach can hardly scale on lengthy traces.

The problem of dealing with long traces is that to infer an adequate model of a component from a set of its execution traces, this set must cover numerous use cases. Intuitively, the more and longer traces are collected from a component under observation the higher the confidence that it is sufficiently representative. Note that in the context of passive inference, we are not controlling the component, as opposed to query learning, aka active inference of FSMs. Unfortunately, multiple lengthy traces pose a problem for the model inference because they significantly increase the time necessary to build a model.

Differently from the existing approaches, our approach does not use the Angluin's tables and builds clauses incrementally, just adding them when a new trace is considered. This allows to use SAT solvers in an incremental way [9]. In incremental SAT solving, the solver processes only newly added formulas, as its state is memorized to accelerate solving.

To process a set of traces incrementally we consider one trace at a time, generate an FSM and verify that it is consistent with the remaining traces. If it is not, choose a trace which is not in the FSM, i.e., a counterexample, and use it to refine the model.

Our incremental inference approach includes in fact two methods for refining conjectures. One is using a prefix and another a suffix instead of processing the whole counterexample trace.

The paper is organized as follows. Section 2 contains definitions. Section 3 provides an overview of passive inference of an FSM from a set of traces based on SAT-solving. In Section 4 we present our incremental inference approach together with preliminary experimental results. Section 5 briefly reports on our experience in applying inference in industrial context and Section 6 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 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') is a transition.

M is *completely specified* if for each tuple $(s, a) \in S \times I$ there exists a transition $(s, a, o, s') \in T$, otherwise M is *incompletely* specified. We use $\Delta(s, a)$ to denote s' and $\lambda(s, a)$ to denote o. M 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 from state s is a sequence of transitions forming a path from s in the state transition diagram of M. The machine M is *strongly connected*, if the state transition diagram of M is a strongly connected graph.

A *trace* of M in state 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 in state s and Tr_M denote the set of traces of M in the initial state. Let $\mathcal{T}$ be a set of traces, we say that M is *consistent* with $\mathcal{T}$ if $\mathcal{T} \subseteq Tr_M$. We also say that M is a *conjecture* for $\mathcal{T}$. If all FSMs with fewer states than M are not consistent with $\mathcal{T}$, then we say that M is a *minimal* FSM consistent with $\mathcal{T}$.

We say that two states $s_1, s_1' \in S$ are *incompatible*, if for every two transitions $(s_1, a, o, s_2), (s_1', a, o', s_2') \in T$ it holds that: $o \neq o'$ or s_2 and s_2' are incompatible, denoted $s_1 \ncong s_1'$. If s_1 and s_1' are not incompatible, then they are *compatible*, denoted $s_1 \cong s_1'$.

3 Passive Inference

Two types of methods solving the problem of learning an automaton from a set of sample traces can be distinguished.

One group constitutes heuristic methods derived from the algorithm of Gold [13] which try to merge states in polynomial time. They are often used in practice because of their efficiency, however, they provide no guarantee for the optimality, since there may exist another way of state merging which provides a resulting machine with a fewer states. Numerous existing heuristic methods allow to infer Mealy machines [21], Moore machines [11] as well as DFA[18].

Another group includes exact algorithms to determine an FSM model with a minimal number of states. This is a much more complicated problem, as it is NP-complete [13], but the minimality may prove to be essential in certain cases. Among existing algorithms for finding a minimal solution, we could mention the algorithm of Heule and Verwer [15] which in our opinion, can be considered as the most efficient currently existing method.

In this paper, we elaborate an approach for the exact FSM inference in an incremental way. It is SAT-solving based and has the advantage of having a low sensibility to the length and number of sample traces.

We rely on the SAT encoding of Heule and Verwer, which we overview in this section, though any other SAT formulation could also be used in our approach.

3.1 Problem statement

Given a set of traces $\mathcal{T}$ generated by an unknown deterministic FSM, we want to find a minimal FSM M consistent with $\mathcal{T}$, i.e., $\mathcal{T} \subseteq Tr_M$.

Given $\mathcal{T}$, let $W = (X, x_0, I, O, T)$ be a deterministic acyclic FSM such that $Tr_W = \mathcal{T}$. Clearly, W is incompletely specified because the FSM is acyclic. To find an FSM with at most n states consistent with $\mathcal{T}$ amounts to determine a partition π on the set of states X into compatible states such that the number of blocks does not exceed n. Clearly, n should be smaller than $|X|$.

This problem can be cast as a constraint satisfaction problem (CSP) [7]. The set of states X is represented by integer variables $x_0, ..., x_{|X|-1}$, such that

$$\begin{aligned} \forall x_i, x_j \in X: \ & \text{if } x_i \ncong x_j \text{ then } x_i \neq x_j \\ & \text{if } \exists a \in I : \lambda(x_i, a) = \lambda(x_j, a) \text{ then} \\ & \quad (x_i = x_j) \Rightarrow (\Delta(x_i, a) = \Delta(x_j, a)) \end{aligned} \tag{1}$$

Let $B = \{0, ..., n-1\}$ be a set of integers where each integer represents a block of a partition π. Assuming that the value of x_i is in B for all $i \in \{0, ..., |X|-1\}$, we need to find a solution, i.e., an assignment of values of variables in $\{x_0, ..., x_{|X|-1}\}$ such that (1) is satisfied. Each assignment implies a partition of n blocks and thus an FSM with n states consistent with $\mathcal{T}$.

3.2 Encoding as a SAT problem

The previous CSP formulas can be translated to SAT using unary coding for each integer variable $x \in X$: x is represented by n Boolean variables $v_{x,0}, v_{x,1}, ..., v_{x,n-1}$. To identify the initial state we have the clause:

$$v_{x_0,0} \tag{2}$$

It means that the state x_0 should be in the first block.

For each state $x \in X$, we have the clause:

$$v_{x,0} \vee v_{x,1} \vee ... \vee v_{x,n-1} \tag{3}$$

These clauses mean that each state should be in at least one block.

For each state x and $\forall i, j \in B$ such that $i \neq j$, we have the clauses:

$$\neg v_{x,i} \vee \neg v_{x,j} \tag{4}$$

These clauses mean that each state should be in at most one block.

The clauses 3 and 4 encode the fact that each state should be in exactly one block.

For every incompatible states $x, x' \in X$ and $\forall i \in B$, we have the clauses:

$$\neg v_{x,i} \vee \neg v_{x',i} \tag{5}$$

These clauses mean that two incompatible states should not be in the same block.

For every states $x, x' \in X$ such that $\lambda(x, a) = \lambda(x', a)$, and $\forall i, j \in B$, we have a Boolean formula (which can be translated trivially into clauses):

$$(v_{x,i} \wedge v_{x',i}) \Rightarrow (v_{\Delta(x,a),j} \Rightarrow v_{\Delta(x',a),j}) \tag{6}$$

These clauses enforce determinism.

Note that the clauses (5) encode the first line of the CSP constraint (1) and the clauses (6) encode the second line. An existing SAT solver [5,6,10,20] can be used to check satisfiability of the obtained formula.

3.3 Auxiliary variables

Heule and Verwer [15] propose to use auxiliary variables, replacing formula (6) and add some additional clauses. They provide experimental results which indicate that their encoding is sufficiently efficient. Namely, for $a \in I$ and $0 \leq i, j < n$, variable $y_{a,i,j}$ is introduced for True value means that for any state in block i, the next state reached with input a is in the block j. These variables are used to form the following clauses.

For each transition $(x, a, o, x') \in T$ and for every $i, j \in B$:

$$y_{a,i,j} \vee \neg v_{x,i} \vee \neg v_{x',j} \tag{7}$$

This means that blocks i and j are related for input a if state x is in the block i and its successor x' on input a is in the block j.

For each input symbol $a \in I$, for every $i, h \in B$ and for each $j \in \{h+1, n-1\}$:

$$\neg y_{a,i,h} \vee \neg y_{a,i,j} \tag{8}$$

This means that each block relation can include at most one pair of blocks for each input to enforce determinism.

For each input symbol $a \in \Sigma$ and each $i \in B$:

$$y_{a,i,0} \vee y_{a,i,1} \vee ... \vee y_{a,i,n-1} \tag{9}$$

This means that each block relation must include at least one pair of blocks for each input to enforce determinism.

For each transition $(x, a, o, x') \in T$ and for every $i, j \in B$:

$$\neg y_{a,i,j} \vee \neg v_{x,i} \vee v_{x',j} \tag{10}$$

This means that once blocks i and j are related for input a and state x is in the block i then its successor x' on input a must be in the block j.

Among these clauses, some are redundant. Nevertheless, their use improves the performance of FSM inference as work in [15] suggests.

3.4 Symmetry Breaking

It is possible that for certain formulations of a SAT formula, some assignments are equivalent, i.e., represent a same solution. In this case, we say that we have a symmetry. A good practice is to break this symmetry [2,3,8] by adding constraints such that different assignments satisfying the formula represent different solutions.

The above formulation can result in a significant amount of symmetry because any permutation of the blocks is allowed. This fact has already been noticed in the literature and the strategy adopted in [1,15] consists in placing each state in a certain subset to a fixed distinct block. To this end, we can use the state incompatibility graph which has $|X|$ nodes and two nodes are connected iff the corresponding states of W are incompatible. Clearly, each state of a clique (maximal or smaller) must be placed in a distinct block. Hence, we can add to the SAT formula clauses for assigning initially each state from the clique to a separate block.

Table 1. Summary for encoding passive inference from ISFSM $W = (X, x_0, I, O, T)$ into SAT. n is the maximal number of states in an FSM to infer, $B = \{0, ..., n-1\}$.

Ref	Clauses	Range
(2)	$v_{x_0,0}$	
(3)	$(v_{x,0} \vee v_{x,1} \vee ... \vee v_{x,n-1})$	$x \in X$
(4)	$(\neg v_{x,i} \vee \neg v_{x,j})$	$x \in X; 0 \leq i < j < n$
(5)	$(\neg v_{x,i} \vee \neg v_{x',i})$	$x \not\cong x'; i \in B$
(7)	$(y_{a,i,j} \vee \neg v_{x,i} \vee \neg v_{x',j})$	$(x, a, o, x') \in T; i, j \in B$
(8)	$(\neg y_{a,i,h} \vee \neg y_{a,i,j})$	$a \in I; h, i, j \in B; h < j$
(9)	$(y_{a,i,0} \vee y_{a,i,1} \vee ... \vee y_{a,i,n-1})$	$a \in I; i \in B$
(10)	$(\neg y_{a,i,j} \vee \neg v_{x,i} \vee v_{x',j})$	$(x, a, o, x') \in T; i, j \in B$

4 Incremental Inference

To alleviate the complexity associated with large sets containing lengthy traces, we propose an approach which, instead of attempting to process all the given traces in the set $\mathcal{T}$ at once, iteratively infers an FSM from their subset (initially it is an empty set) and uses active inference to refine it when it is not consistent with one of the given traces. While active inference usually uses a black box as an oracle capable of judging whether or not a trace belongs to the model, we assign the role of an oracle to a set of traces $\mathcal{T}$. Even if this oracle is restricted since it cannot generate traces for all possible input sequences, nevertheless, as we demonstrate, it leads to an efficient approach for passive inference from execution traces.

The proposed approach is elaborated in two methods performing different refinements of a conjecture inconsistent with a given set of traces. Refinement needs to be performed when the shortest prefix ω of a trace in $\mathcal{T}$ which is not a trace of the conjecture is found. The first type of refinement consists in adding ω to the conjecture's initial state which is achieved by formulating the corresponding constraints. We present this method in Section 4.1. The second type of refinement consists in adding not ω but its shortest suffix ω' which is not a trace of any state of the conjecture. The suffix ω' is added to some state of the conjecture which is achieved by formulating the corresponding constraints. This method is elaborated in Section 4.2.

We provide the results of experimental evaluation of the two methods and discuss them in Section 4.3.

4.1 Prefix-based method

Let $\mathcal{T}$ be a set of traces (generated by a deterministic FSM). We want to find a minimal FSM consistent with $\mathcal{T}$ iteratively. To do that, we search for an FSM M with at most n states satisfying a growing set of constraints (initially we do not have any constraints). If no solution is found, it means that the state number n is too low. In this case we increase n and start again. If a solution is found and M is consistent with $\mathcal{T}$, then we return this solution. Otherwise, we find the shortest prefix of a trace ω in $\mathcal{T}$ not accepted by M. Then, we use SAT encoding described in the previous section to formulate the constraint that ω has to be a trace of the conjecture.

The approach is formalized in Algorithm 1.

Theorem 1. *Given a set of traces $\mathcal{T}$, Algorithm 1 returns an FSM consistent with $\mathcal{T}$ if it exists or $false$ otherwise.*

Proof. If line 5 is reached, then $\mathcal{T} \subseteq Tr_M$. So M is consistent with $\mathcal{T}$. If line 10 is reached, then C encoding the fact that all traces of $\mathcal{T}$ have to be included in an FSM with at most n states is not satisfiable. So, there is no FSM with at most n states consistent with $\mathcal{T}$. The termination is assured because in each while loop, additional trace of $\mathcal{T}$ is considered. When all traces are considered

Algorithm 1 Infer an FSM from a set of traces

Input: A set of traces $\mathcal{T}$ and an integer n.
Output: An FSM with at most n states consistent with $\mathcal{T}$ if it exists.

```
1: C := ∅
2: while C is satisfiable do
3:     Let M be an FSM of a solution of C
4:     if T ⊆ Tr_M then
5:         return M
6:     end if
7:     Let ω be the shortest trace in T\Tr_M
8:     C := C ∧ C_ω, where C_ω is clauses encoding the fact that ω ∈ Tr_M using Table 1
9: end while
10: return false
```

then either there is a solution and $\mathcal{T} \subseteq Tr_M$ terminates the function, or there is no solution and the while condition is no longer respected.

Corollary 1. *Let $\mathcal{T}$ be a set of traces. If we call Algorithm 1 incrementally by increasing n from $n = 1$ until an FSM consistent with $\mathcal{T}$ is obtained, then it is a minimal FSM consistent with $\mathcal{T}$.*

4.1.1 Example

We illustrate Algorithm 1 with a simple example of a small program, see Algorithm 2.
Let w = *ping/pong.pause/pause.ping/pause.ping/pause.pause/pong .pause/pause.ping/pause.pause/pong.ping/pause.ping/pause.pause/pause .ping/pause.ping/pause.pause/ping.ping/pong.ping/pong...* be the only trace in $\mathcal{T}$ obtained by random execution of the program.

Algorithm 2

```
1: while true do
2:     Event msg = receive();
3:     if msg == ping then
4:         send(pong);
5:     end if
6:     if msg == pause then
7:         send(pause);
8:         while receive() ≠ pause do
9:             send(pause);
10:        end while
11:        send(pong);
12:    end if
13: end while
```

Initially, we consider as a conjecture the trivial FSM with the empty trace. The shortest prefix trace inconsistent with this conjecture is *ping/pong*, and so, some clauses are added to ensure that the trace *ping/pong* is accepted. A new conjecture is an FSM with a single state having self-looping transition labeled *ping/pong*. This time, the shortest prefix inconsistent with this conjecture is *ping/pong.pause/pause*. This trace yields new constraints leading to a next conjecture with a single state having two self-looping transitions labeled *ping/pong* and *pause/pause*. This conjecture is consistent with the two considered traces *ping/pong* and *ping/pong.pause/pause* but still not with the whole w. The process continues while the constraints are satisfiable. All the executed steps are illustrated in Figure 1. A trace beneath a conjecture is a prefix used to obtain the conjecture.

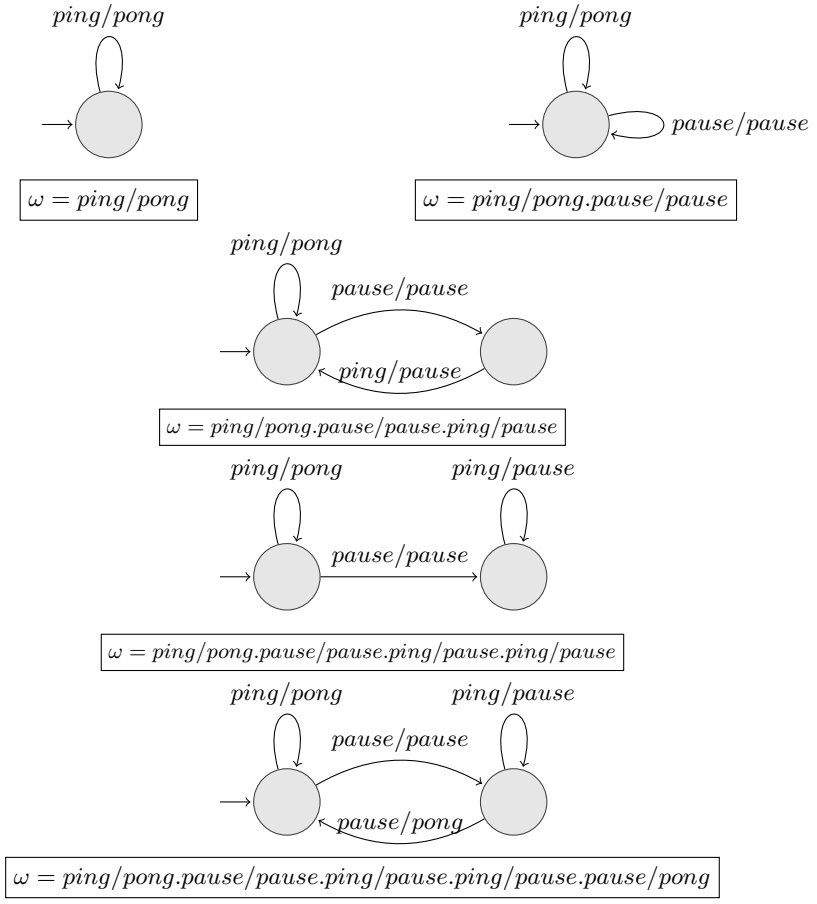

Fig. 1. Inferring an FSM from trace *ping/pong.pause/pause.ping/pause.ping/pause .pause/pong.pause/pause.ping/pause.pause/pong.ping/pong.ping/pong.pause/pause .ping/pause.ping/pause.pause/pong.ping/pong.ping/pong...* with Algorithm 1.

4.1.2 Evaluation

To the best of your knowledge, the approach of Heule and Verwer [15] is currently the most efficient encoding of the FSM inference into SAT. In this section, we provide results of experimental comparison of their approach with ours.

We have implemented the encoding of the inference problem to SAT using the Heule and Verwer's formulas as described in Table 1. We use **H&V** and **Prefix-based** to refer to the method of Heule and Verwer and Algorithm 1, respectively.

The prototype was implemented in C++ calling the SAT solver Cryptominisat [20]. The experiments were carried out on a machine with 8 GB of RAM and an i7-3537U processor.

We randomly generate FSMs with seven states, two inputs a and b, and two outputs 0 and 1. Each state s_i is linked to the state $s_{i+1 \ mod \ 8}$ by a transition with input a and a random output to ensure that machines are strongly connected. Then we complete an FSM in a random way. Fig. 2 shows an example of such a construction.

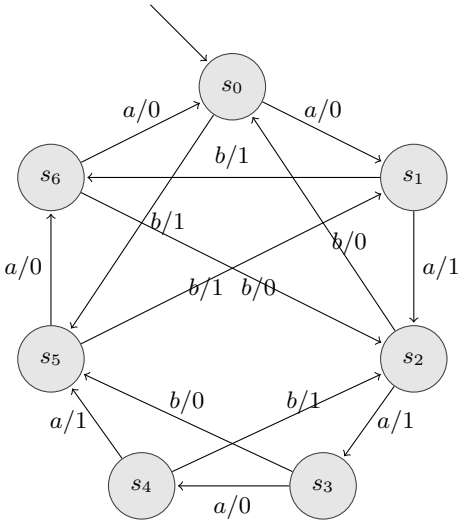

Fig. 2. Example of a random FSM.

Given an FSM, traces of various length are randomly generated. Table 2 and Table 3 show time used to infer an FSM from a single trace and 100 traces, respectively. For each length of traces, we calculate the average time used to infer a machine over ten instances.

Table 2. Seconds to infer an FSM from a trace.

Length	H&V	Prefix-based
1k	2.5	0.2
2k	7.3	0.2
4k	20	0.2
8k	53	0.2
16k	190	0.3
32k	Out of Memory	0.5
64k	Out of Memory	1.5

Table 3. Seconds to infer an FSM from 100 traces.

Length	H&V	Prefix-based
100	31	< 0.1
200	140	< 0.1
400	590	< 0.1
1k	Out of Memory	0.1
10k	Out of Memory	3.8

The results in Table 2 and 3 indicate that the proposed approach performs much better than that of [15]. Moreover, they show that time used by Algorithm 1 grows very slow when the size of traces increases. This is due to the fact that the approach of [15] uses all the traces at once for inference while our approach is incremental and requires a minimal prefix of a single trace among the given traces in each iteration.

Table 4. Inferring FSM from a single trace with length 100k by Algorithm 1.

# States	Checking $\mathcal{T} \subseteq Tr_M$	SAT Solving	Total
5	3.33 sec.	0.003 sec.	3.46 sec.
6	3.28 sec.	0.02 sec.	3.42 sec.
7	3.28 sec.	0.076 sec.	3.48 sec.
8	3.27 sec.	0.57 sec.	3.98 sec.
9	3.28 sec.	7.23 sec.	10.7 sec.
10	3.28 sec.	28.6 sec.	32.0 sec.

Table 5. Inferring FSM from a single trace with 10 states by Algorithm 1.

Length	Checking $\mathcal{T} \subseteq Tr_M$	Solving SAT	Total
25k	0.17 sec.	22.9 sec.	23.2 sec.
50k	0.79 sec.	25.1 sec.	26.0 sec.
75k	1.83 sec.	26.4 sec.	28.3 sec.
100k	3.28 sec.	28.6 sec.	32.0 sec.

In Table 4 and Table 5, we show the results obtained when we push our algorithm to its limits.

In Table 4, we increase the state number and set the length of generated traces to 100k. In Table 5, we set the state number to 10 and increase the trace length generated by this FSM.

We can see that the time (an average of 100 instances) used by the SAT solver depends on the number of states of the machine to infer, but very little on the length of the traces used. On the other hand, the time to test the trace inclusion depends on the length of a trace, but not on the number of states in the FSM to infer.

It is not surprising that overall time grows rapidly with the number of FSM states to infer (because the problem remains NP-Complete), on the other hand, it is interesting to notice that time grows almost linearly with the length of traces.

4.2 Suffix-based method

In the prefix-based method, when a generated conjecture M is inconsistent with $\mathcal{T}$, we use the shortest prefix of a trace to refine the conjecture. Clearly, the longer the prefix the more clauses are added to the constraints. This observation motivates our second method which is using a different refinement.

The idea is that when we find the shortest trace ω which is in $\mathcal{T}$ but not in M, we determine a suffix ω' of ω such that ω' is not a trace of any state of the conjecture M. Then if we add the constraint that there exists a state s such that ω' must be accepted by M in state s, thus M is refuted and will be refined. If a refined conjecture which accepts w' does not accept the whole w yet then a longer suffix is considered in the next iteration until w is in Tr_M. When the suffix ω' is shorter than ω, the number of added clauses can be smaller compared to the use of ω.

Theorem 2. *Given a set of traces $\mathcal{T}$, Algorithm 3 returns an FSM consistent with $\mathcal{T}$ if it exists or false otherwise.*

Proof. If line 5 is reached, then $\mathcal{T} \subseteq Tr_M$ and so M is consistent with $\mathcal{T}$. If line 14 is reached, then all the traces of $\mathcal{T}$ cannot be represented by an FSM with n states. The termination is assured because in each while loop, additional suffix of a trace of $\mathcal{T}$ is considered. When all suffixes of all traces are considered then either there is a solution and $\mathcal{T} \subseteq Tr_M$ terminates the function, or there is no solution and the while condition is no longer respected.

Corollary 2. *Let $\mathcal{T}$ be a set of traces. If we call Algorithm 3 incrementally by increasing n from $n = 1$ until an FSM consistent with $\mathcal{T}$ is obtained, then it is a minimal FSM consistent with $\mathcal{T}$.*

Algorithm 3 Infer an FSM from a set of traces.

Input: A set of traces $\mathcal{T}$ and an integer n.
Output: An FSM with a most n states consistent with $\mathcal{T}$ if it exists.

1: $C := \emptyset$
2: **while** C is satisfiable **do**
3: Let M be an FSM of a solution of C
4: **if** $\mathcal{T} \subseteq Tr_M$ **then**
5: **return** M
6: **end if**
7: Let ω be the shortest trace in $\mathcal{T} \backslash Tr_M$
8: **if** $\exists \omega'$ the shortest suffix of ω such that $\forall s, \omega' \notin Tr(s)$ **then**
9: $C := C \wedge C'_\omega$, where C'_ω is clauses encoding the fact that $\exists s : \omega' \in Tr(s)$ using Table 1 without the first constraint.
10: **else**
11: $C := C \wedge C_\omega$, where C_ω is clauses encoding the fact that $\omega \in Tr_M$ using Table 1.
12: **end if**
13: **end while**
14: **return** $false$

4.2.1 Example

We illustrate the suffix-based method with the example from Section 4.1.1. We use the same trace $w = ping/pong.pause/pause.ping/pause.ping/pause$ $.pause/pong.pause/pause.ping/pause.pause/pong.ping/pause.ping/pause$ $.pause/pause.ping/pause.ping/pause.pause/ping.ping/pong.ping/pong...$ as the only trace in $\mathcal{T}$ obtained by executing Algorithm 2. Fig. 3 shows intermediate conjectures with the suffixes added to obtain them.

4.2.2 Evaluation

Comparing the execution of the two methods on the same trace in Sections 4.1.1 and 4.2.1, one can notice that the suffix-based method uses instead of a long trace with an event making a conjecture inconsistent just its much shorter suffix with that event. An example of such an event in the trace is *ping/pause*. Intuitively, all things being equal, a suffix could be shorter than a prefix when an event causing inconsistence occurs seldom.

To check this hypothesis, we decided to extend the experiments reported in the last row of Table 2, where a trace of the length 64000 belongs to an FSM randomly generated as explained in Section 4.1.2. This time we vary the chances for input b to appear. Table 6 contains the averages of ten instances for each value of probability.

We can see that when all inputs are equiprobable (Line 1 of the Table 6), the second algorithm is a little slower. On the other hand, time used by the prefix-

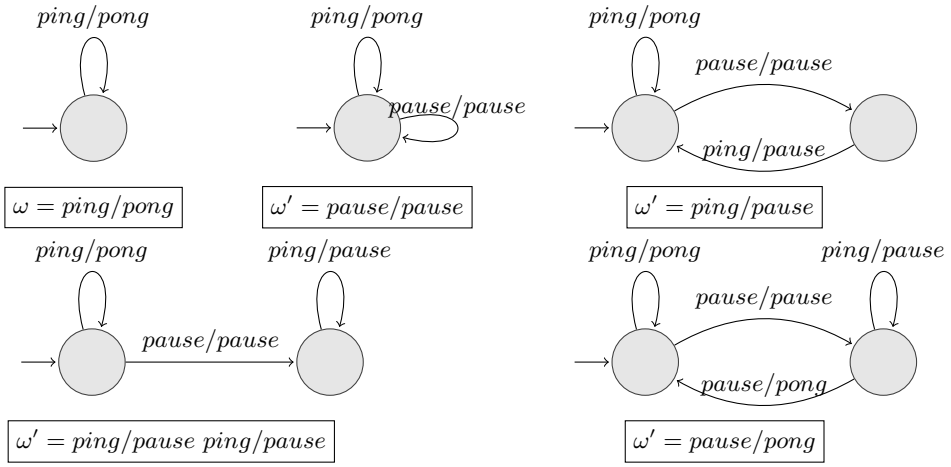

Fig. 3. Inferring an FSM from trace *ping/pong.pause/pause.ping/pause .ping/pause.pause/pong.pause/pause.ping/pause.pause/pong.ping/pong.ping/pong .pause/pause.ping/pause.ping/pause.pause/pong.ping/pong.ping/pong...* with Algorithm 3

.

Table 6. Seconds to infer an FSM from a trace with different probabilities of input b.

Probability of b	**Prefix-based**	**Suffix-based**
50 %	1.5	2.4
25 %	1.4	1.4
10 %	1.4	1.4
1 %	3.5	1.4
0.5 %	6.0	1.5
0.3 %	16	1.4
0.2 %	90	1.5
0.1 %	Out of Memory	1.6

closed method grows when an input rarely appears, but it remains constant with the suffix-closed method, as expected in the hypothesis.

4.3 Discussion

We presented two methods for incremental inference of a minimal FSM consistent with a given set of traces. As could be expected, experimental results indicate that inference from an incrementally growing subset of traces has a big advantage compared to the classical inference from all the given traces at once, as in the method of Heule and Verwer [15] when the traces are rather long and numerous. The reason for that is the proposed approach avoids as long as possible to generate constraints from the whole initial traces, and tries to find instead

their appropriate portions, prefixes, as in the prefix-based method or suffixes, as in the suffix-based method.

Comparing the two proposed methods, we understand that their efficiency depends on intricate properties of given traces. Preliminary experiments confirm our hypothesis that rare key events in a trace may create favorable conditions for the suffix-based method to perform more efficiently than its counterpart, the prefix-based method.

5 Industrial Case Study

Our industrial partner provided us with logs of a flight simulator expecting us to produce state machine models of some components involved in the logged executions of the simulator. Models are considered as an important part of documentation, especially for legacy components and components from the third party. They facilitate change impact analysis, regression testing and other tasks of simulator development and maintenance.

The logs are normally collected while executing flight scenarios defined by experts and come in the form of time series of at least 12000 steps.

Clearly, using traditional inference methods directly on time series is out of the question; preprocessing we performed includes their partitioning into smaller time series caused by inputs and replacing time series which are "close" to others such that a limited number of time series become outputs in a state machine model. Vectors of values of input variables present in a log become inputs in the model. In the processed traces their number reaches a dozen. The inferred FSMs have up to ten states.

Flight simulator experts consider the resulting models sufficiently adequate and useful. In this case study log preprocessing turns out to be more challenging and time consuming than the model inference with the prototype we developed. The scalability of the whole approach may, however, be an issue for processing logs resulting from long flight scenarios. The latter need more aggressive preprocessing, e.g., excluding parts where "not much happening", this would favor our suffix-based method which looks for turning events.

6 Conclusion

In this paper we considered the problem of inferring a minimal FSM consistent with a set of long traces. Although this problem has extensively been studied, the efficiency of the existing methods deteriorates quickly with the size of the given traces. We proposed in this paper an approach aimed at dealing with long traces. The need for it comes from the observation that the more and longer traces are collected from a component under observation, the higher the confidence that they are sufficiently representative and would yield an adequate model.

Addressing the scalability issue, we proposed an approach which does not process all the given traces at once, instead it does this incrementally. The idea of processing a set of traces incrementally is to consider one trace at a time, generate an FSM and verify that it is consistent with the remaining traces. If it is not, choose a trace which is not in the FSM, i.e., a counterexample, and use it to refine the model.

Our incremental inference approach includes in fact two methods for refining conjectures. One is using a prefix and another a suffix instead of processing the whole counterexample trace. The approach is SAT-solving based and has the advantage of having a lower sensibility to the length and number of sample traces compared to the existing approaches.

The experimental results indicate that the proposed approach is sufficiently efficient especially for long traces where some inputs occur rather rarely. We plan to perform more experiments to find other ways of improving efficiency.

Acknowledgements

This work was partially supported by MESI (Ministre de lconomie, Science et Innovation) of Gouvernement du Qubec, NSERC of Canada and CAE.

References

1. Andreas Abel and Jan Reineke. Memin: SAT-based exact minimization of incompletely specified mealy machines. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 94–101. IEEE, 2015.
2. Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th annual Design Automation Conference*, pages 731–736. ACM, 2002.
3. Fadi A Aloul, Karem A Sakallah, and Igor L Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006.
4. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.
5. Gilles Audemard and Laurent Simon. The glucose SAT solver, 2013.
6. Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.
7. Alan W Biermann and Jerome A Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE transactions on Computers*, 100(6):592–597, 1972.

8. Cynthia A Brown, Larry Finkelstein, and Paul Walton Purdom Jr. Backtrack searching in the presence of symmetry. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 99–110. Springer, 1988.
9. Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.
10. Niklas Een and Niklas Sörensson. Minisat: A SAT solver with conflict-clause minimization. *Sat*, 5:8th, 2005.
11. Georgios Giantamidis and Stavros Tripakis. Learning moore machines from input-output traces. In *FM 2016: Formal Methods: 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings 21*, pages 291–309. Springer, 2016.
12. E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.
13. E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.
14. Olga Grinchtein, Martin Leucker, and Nir Piterman. Inferring network invariants automatically. *Automated Reasoning*, pages 483–497, 2006.
15. Marijn JH Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.
16. Jehuda Kella. Sequential machine identification. *IEEE Transactions on Computers*, 100(3):332–338, 1971.
17. Arlindo L Oliveira and João PM Silva. Efficient algorithms for the inference of minimum size DFAS. *Machine Learning*, 44(1):93–119, 2001.
18. José Oncina and Pedro García. Identifying regular languages in polynomial time. *Advances in Structural and Syntactic Pattern Recognition*, 5(99-108):15–20, 1992.
19. Rick Smetsers, Paul Fiterău-Broştean, and Frits Vaandrager. Model learning as a satisfiability modulo theories problem. In *International Conference on Language and Automata Theory and Applications*, pages 182–194. Springer, 2018.
20. Mate Soos. Cryptominisat 2.5. 0. *SAT Race competitive event booklet*, 2010.
21. LPJ Veelenturf. Inference of sequential machines from sample computations. *IEEE Transactions on Computers*, 2(C-27):167–170, 1978.
22. Neil Walkinshaw, John Derrick, and Qiang Guo. Iterative refinement of reverse-engineered models by model-based testing. In *International Symposium on Formal Methods*, pages 305–320. Springer, 2009.