Actively learning to verify safety for FIFO automata

Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, Gul Agha
Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, USA
{vardhan,ksen,vmahesh,agha}@cs.uiuc.edu

Abstract. We apply machine learning techniques to verify *safety* properties of *finite state machines* which communicate over *unbounded FIFO channels*. Instead of attempting to iteratively compute the reachable states, we use *Angluin's L^* algorithm* to learn these states. The learnt set of reachable states is then used either to prove that the system is safe, or to produce a valid execution of the system that leads to an unsafe state (*i.e.* to produce a counterexample). Specifically, we assume that we are given a model of the system and we provide a novel procedure which answers both *membership* and *equivalence* queries for a representation of the reachable states. We define a new *encoding* scheme for representing reachable states and their witness execution; this enables the learning algorithm to analyze a larger class of FIFO systems automatically than a naive encoding would allow. We show the upper bounds on the running time and space for our method. We have implemented our approach in Java, and demonstrate its application to a few case studies.

1 Introduction

Infinite state systems often arise as natural models for various software systems at the design and modeling stage. In particular, finite state machines that communicate over unbounded first-in-first-out channels, called *FIFO automata*, are a popular model for various communication protocols, languages such as Estelle and SDL (Specification and Description Language) in which processes have infinite queue size, distributed systems and various *actor* systems. A generic task in the automated verification of safety properties of any system is to compute a representation for the set of reachable states. For finite state systems, this is typically accomplished by doing an exhaustive exploration of the state-space. However, for infinite state systems, except in a few special classes [15], exhaustive exploration of the state space is impossible; and in fact the verification problem in general can shown to be undecidable.

In the Lever (LEarning to VERify) project, we are pursuing the goal of using *machine learning* for verification of infinite state systems. The idea is as follows. Instead of computing the reachable states by iteratively applying the transition relation until a fixpoint is reached (which may not be possible in a finite number of iterations), we view the identification of the reachable states as a machine learning problem. Naturally, in order for a learner to be able to learn the reachable region, we have to provide it with some information about the reachable states. We can easily find examples of reachable states by executing some sample sequence of transitions. Moreover, given a set of states as the supposed reachable region, we can check if this set is a fixpoint under the transition relation. If it is not a fixpoint then clearly it is not the correct reachable region. However, typically learning algorithms also require either negative examples of the concept being learned or the ability to make membership and equivalence queries. In order to provide this information, we learn what we call *annotated trace language* representing not only the reachable states but also system executions witnessing the reachability of these states.

Another crucial problem in the application of the learning based verification approach is to identify the point when we have sufficient information to provide an answer to the verification problem. If the learning algorithm outputs a set of traces that is closed under the transition relation of the system and does not reach any of the unsafe states, then clearly the system can deemed to be correct. On the other hand, unsafe states in the set output by the learning algorithm can be used to obtain executions (counter-examples) leading to the unsafe state. Notice that this relies on the fact that we learn traces which provide witnesses along with the reachable states. Spurious counterexamples can be used by the learner to refine the hypothesis and the process repeated until either a valid counterexample is found or the system shown to be correct. Finally, based on the practical success enjoyed by *regular model checking* [9], we assume that the set of annotated traces to be learnt is in fact regular. Our main observation is that this learning based approach is a *complete verification* method for systems whose annotated trace language is regular (for a precise condition see Section 3). In other words, for such systems we will eventually either find a buggy execution that violates the safety property, or will successfully prove that no unsafe state is reachable. We have previously applied the RPNI[20] algorithm for verification of safety properties (see Section 6 on related work).

This paper presents two main new ideas. Firstly, we give a new scheme for the *annotations* on traces. With this annotation scheme, many more practical FIFO systems have regular annotated trace languages, thus enlarging the class of systems that can be provably verified by our method. Secondly and more significantly, we provide a method to devise a *knowledgeable teacher* which can answer membership (whether a string belongs to the target) as well as equivalence-queries (given a hypothesis, whether it matches the concept being learnt). In the context of learning annotated traces, equivalence queries can be answered only to a limited extent. However, we overcome our limitation to answer equivalence queries exactly and present an approach that is still able to use the powerful query-based learning framework. As mentioned earlier, we assume the annotated traces of the system to form a regular language and use Angluin's L* algorithm [3] which is a well-known algorithm for learning regular sets. Using the L* algorithm gives us significant benefits. First, the number of samples we need to consider is polynomial in the size of the automaton representing the annotated traces. Second, we are guaranteed to learn the *minimal* automaton that represents the annotated traces. Finally, we can show that the running time is bounded by a polynomial in the size of the minimal automaton representing the annotated traces and the time taken to verify if an annotated trace is valid for the FIFO system.

We have implemented our algorithm in Java, and demonstrated the feasibility of this method by running the implementation on simple examples and network protocols such as the alternating-bit protocol and the sliding window protocol. Our approach is complementary to previous methods for algorithmic verification that have been proposed, and there are examples of FIFO automata that our method successfully verifies but on which other approaches fail (see Section 6 on related work). We also give the requirements under which classes of infinite state systems other than FIFO automata can be verified using the learning approach.

2 Preliminaries

In this section, we describe machine learning framework that we use and recall the definition of FIFO automata.

2.1 Learning framework

A learning algorithm is usually set in a framework which describes the types of input data and queries available to the learner. We use the active learning framework in which the learner is allowed to make both membership and equivalence queries to a *teacher*. As in [24], we focus on learning of regular languages; based on the experience of *regular model checking* [9], regular languages are often sufficient to capture the behavior of an interesting class of infinite state systems. The learning algorithm that we use for regular sets is Angluin's L* algorithm [3].

Angluin's L* algorithm requires what is called a *Minimally Adequate Teacher* which provides an oracle for membership (whether a given string belongs to a target regular set) and equivalence queries (whether a given hypothesis matches the target regular set). In case the teacher answers *no* to an equivalence query, it also provides a string in the symmetric difference of the hypothesis and the target sets. The main idea behind Angluin's L* algorithm is to systematically explore strings in the alphabet for membership and create a DFA with minimum number of states to make a conjecture for the target set. If the conjecture is incorrect, the string returned by the teacher is used to make corrections, possibly after more membership queries. The algorithm maintains a prefix closed set S representing different possible states of the target DFA, a set SA for the transition function consisting of strings from S extended with one letter of the alphabet and a suffix closed set E denoting *experiments* to distinguish between states. An *observation table* with rows from $(S \cup SA)$ and columns from E stores results of the membership queries for strings in $(S \cup SA).E$ and is used to create the DFA for a conjecture. Angluin's algorithm is guaranteed to terminate in polynomial time with the minimal DFA representing the target set.

2.2 FIFO Automata

A FIFO automaton [14] is a 6-tuple $(Q, q_0, C, M, \Theta, \delta)$ where Q is a finite set of *control states*, $q_0 \in Q$ is the initial control state, C is a finite set of *channel names*, M is a finite alphabet for contents of a channel, Θ is a finite set of transitions names, and $\delta : \Theta \rightarrow Q \times ((C \times \{?, !\} \times M) \cup \{\tau\}) \times Q$ is a function that assigns a *control transition* to each transition name. For a transition name θ, if the associated control transition $\delta(\theta)$ is of the form $(q, c?m, q')$ then it denotes a *receive* action, if it is of the form $(q, c!m, q')$ it denotes a *send* action, and if it is of the form (q, τ, q') then it denotes an *internal* action. The channels are considered to be perfect and messages sent by a sender are received in the order in which they were sent. The formal operational semantics, given by a labelled transition systems, is defined below.

A FIFO automaton $F = (Q, q_0, C, M, \Theta, \delta)$ defines a labelled transition system $\mathcal{L} = (S, \Theta, \rightarrow)$ where

- The set of states $S = Q \times (M^*)^C$; in other words, each state of the labelled transition system consists of a control state q and a C-indexed vector of words w denoting the channel contents.
- If $\delta(\theta) = (q, c?m, q')$ then $(p, w) \xrightarrow{\theta} (p', w')$ iff $p = q$, $p' = q'$ and $w = w'[c \mapsto m \cdot w'[c]]$
- If $\delta(\theta) = (q, c!m, q')$ then $(p, w) \xrightarrow{\theta} (p', w')$ iff $p = q$, $p' = q'$ and $w' = w[c \mapsto m \cdot w[c]]$
- If $\delta(\theta) = (q, \tau, q')$ then $(p, w) \xrightarrow{\theta} (p', w')$ iff $p = q$, $p' = q'$ and $w' = w$.

Here $w[i \mapsto s]$ stand for the C-indexed vector which is identical to w for all channels except i, where it is s; $w[i]$ denotes the contents of the channel i. We say $(p, w) \rightarrow (p', w')$

provided there is some θ such that $(p, w) \xrightarrow{\theta} (p', w')$. As usual, $\rightarrow^*$ will denote the reflexive transitive closure of $\rightarrow$. For $\sigma = \theta_1\theta_2 \cdots \theta_n \in \Theta^*$, we say $(p, w) \xrightarrow{\sigma} (p', w')$ when there exist states $(p_1, w_1) \ldots (p_{n-1}, w_{n-1})$ such that $(p, w) \xrightarrow{\theta_1} (p_1, w_1) \xrightarrow{\theta_2} \cdots (p_{n-1}, w_{n-1} \xrightarrow{\theta_n} (p', w')$. The trace language of the FIFO automaton is

$$L(F) = \{\sigma \in \Theta^* \mid \exists s = (p, w).\ s_0 \xrightarrow{\sigma} s\}$$

where $s_0 = (q_0, (\epsilon, \ldots, \epsilon))$, i.e., the initial control state with no messages in the channels.

3 Verification procedure

We assume that we are given a model of the FIFO automata which enables us to identify the transition relation of the system. The central idea in our approach is to learn a representation for the set of reachable states instead of computing it by iteratively applying the transition relation. Once the set of reachable states is learned, we can verify if the safety property is violated by checking if an unsafe state is among the set of reachable states. To use Angluin's L* algorithm for learning, we need to answer both membership and equivalence queries for the reachable set. However, there is no immediate way to verify if a certain state is really reachable or not. A solution to this problem is to also keep a candidate witness (in terms of the transitions of the system) to a reachable state. It can then be checked if this witness can ever be exhibited by the system. Using this, given a purported reachable state and its witness, we can answer a query about its membership in the actual reachable region. Therefore, instead of learning the set of reachable states directly, we learn a language which allows us to identify both the reachable states and their witnesses.

For equivalence queries, we can provide an answer in one direction. We will show that the reachable region with its witness executions can be seen as the least fixpoint of a relation derived from the transitions. Hence, an answer to the equivalence query can come from checking if the proposed language is a fixpoint under this relation. If it is not a fixpoint then it is certainly not equivalent to the target; but if it is a fixpoint, we are unable to tell if it is also the least fixed point. However, we are ultimately interested in only checking whether a given safety property holds. If the proposed language is a fixpoint but does not intersect with the unsafe region, the safety property clearly holds and we are done. On the other hand, if the fixpoint does intersect with unsafe states, we can check if such an unsafe state is indeed reachable using the membership query. If the unsafe state is reachable then we have found a valid counterexample to the safety property and are done. Otherwise the proposed language is not the right one since it contains an invalid trace.

Figure 1 shows the high level view of the verification procedure. The main problems we have to address now are:

- What is a suitable representation for the reachable states and their witnesses?
- Given a language representation, we need to answer the following questions raised in Figure 1:
  - (Membership Query) Given a string x, is x a valid string for a reachable state and its witness?
  - (Equivalence Query(I)) Is a hypothetical language L a fixpoint under the transition relation? If not, we need a string which demonstrates that L is not a fixpoint.

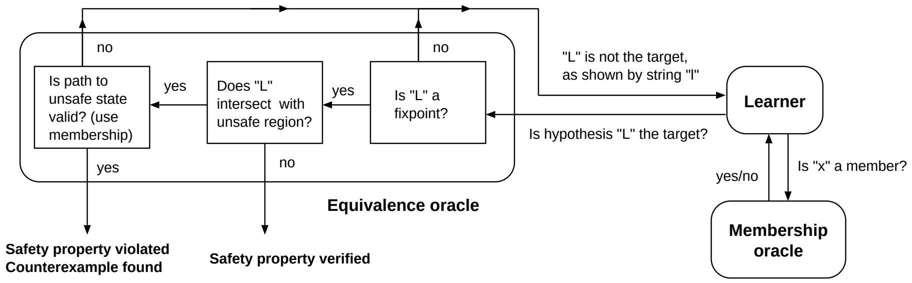
no
no
Is path to unsafe state valid? (use membership)
yes
Does "L" intersect with unsafe region?
yes
Is "L" a fixpoint?
"L" is not the target, as shown by string "l"
Learner
Is hypothesis "L" the target?
yes
no
Equivalence oracle
yes/no
Is "x" a member?
Membership oracle
Safety property violated Counterexample found
Safety property verified

Fig. 1. Verification procedure

- (Equivalence Query(II)) Does any string in L witness the reachability of some "unsafe" state?

3.1 Representation of the reachable states and their witnesses

Let us now consider the language which can allow us to find both reachable states and their witnesses. The first choice that comes to mind is the language of the traces, $L(F)$. Since each trace uniquely determines the final state in the trace, $L(F)$ has the information about the states that can be reached. While it is easy to compute the state s such that $s_0 \xrightarrow{\sigma} s$ for a *single* trace σ, it is not clear how to obtain the set of states reached, given a *set of traces*. In fact, even if $L(F)$ is regular, there is no known algorithm to compute the corresponding set of reachable states of the labelled transition system.[1] The main difficulty is that determining if a receive action can be executed depends non-trivially on the sequence of actions executed before the receive.

In [24], we overcame this difficulty by annotating the traces in a way that makes it possible to compute the set of reachable states. We briefly describe this annotation scheme before presenting the actual scheme used in this paper. Consider a set $\overline{\Theta}$ of *co-names* defined as follows:

$$\overline{\Theta} = \{\overline{\theta} \mid \theta \in \Theta \text{ and } \delta(\theta) \neq \tau\}$$

Thus, for every send or receive action in our FIFO automaton, there is a new transition name with a *bar*. The intuition of putting the annotation of a bar on some transitions of a trace is to indicate that the message sent or received as a result of this transition does not play a role in the channel contents of the final state. In other words, a *barred* transition $\overline{\theta}$ in an annotated trace of the system denotes either a message sent that will later be consumed, or the receipt of a message that was sent earlier in the trace. Annotated traces of the automaton are obtained by marking send-receive pairs in a trace exhibited by the system.

The above annotation scheme allowed us to calculate the reachable set for any regular set of annotated traces by a simple homomorphism. However, one difficulty we encountered is that for some practical FIFO systems, the annotated trace language is not regular; the nonregularity often came from the fact that a receive transition has to be matched to a send which could have happened at an arbitrary time earlier in the past. To alleviate this problem, we use a new annotation scheme in which only the send part of the send-receive pair is kept. This gives an annotated trace language which is

[1] This can sometimes be computed for simple loops using meta-transitions.

regular for a much larger class of FIFO systems (although we cannot hope to be able to cover all classes of FIFO systems since they are Turing expressive). We now describe this annotation in detail.

As before, we have a new set of barred names but this time only for the send transitions:

$$\overline{\Theta} = \{\overline{\theta} \mid \theta \in \Theta \text{ and } \delta(\theta) = c_i!a_j \text{ for some } c_i, a_j\}$$

We also define another set of names $T_Q = \{t_{\mathrm{q}} \mid q \in Q\}$ consisting of a symbol for each control state in the FIFO.

Now let the alphabet of *annotated traces* Σ be defined as $(\Theta - \Theta_r) \cup \overline{\Theta} \cup T_Q$ where Θ_r is the set of receive transitions $\{\theta_r \mid \delta(\theta_r) = c_i?a_j$ for some c_i,a_j $\}$.

Given a sequence of transitions l in $L(F)$, let $\mathcal{A}$ be a function which produces an annotated string in Σ^*. $\mathcal{A}$ takes each receive transition θ_{r_i} in l and finds the matching send transition θ_{s_i} which must occur earlier in l. Then, θ_{r_i} is removed and θ_{s_i} replaced by $\overline{\theta_{s_i}}$. Once all the receive transitions have been accounted for, $\mathcal{A}$ appends the symbol $t_q \in T_Q$ corresponding to the control state q which is the destination of the last transition in l. Intuitively, for a send-receive pair which cancel each other's effect on the channel contents, $\mathcal{A}$ deletes the received transition and replaces the send transition with a barred symbol. As before, a barred symbol indicates that the message sent does not play a role in the channel contents of the final state. Notice that in the old annotation scheme both the send and the receive were replaced with a barred version; here the receive transition is dropped altogether. The reason we still keep the send transition with a bar is, as we will show shortly, that this allows us to decide whether any given string is a valid annotated trace. The symbol t_q is appended to the annotated trace to record the fact that the trace l leads to the control state q.

As an example, consider the FIFO automaton shown in Figure 2. For the following traces in $L(F)$: $\theta_1\theta_2\theta_3$, $\theta_1\theta_2\theta_3\theta_1\theta_2$, the strings output by $\mathcal{A}$ are respectively: $\overline{\theta_1}\theta_3 t_{q_0}$, $\overline{\theta_1\theta_3}\theta_1 t_{q_2}$.

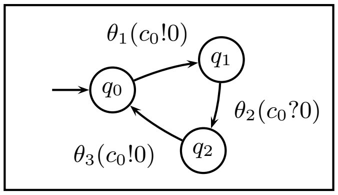

Fig. 2. Example FIFO automata

Let the language of annotated traces be $AL(F) = \{\mathcal{A}(t) \mid t \in L(F)\}$ which consists of all strings in Σ^* that denote correctly annotated traces of F. Let $AL^{\mathrm{old}}(F)$ be the annotated trace language corresponding to the old annotation scheme described earlier (in which we keep both parts of a send-receive pair). The following proposition (for proof see the Appendix) shows that the new annotation scheme has regular annotated trace language for more FIFO automata than the old scheme.

Proposition 1. *The set of FIFO automata for which $AL(F)$ is regular is strictly larger than the set of FIFO automata for which $AL^{\mathrm{old}}(F)$ is regular.*

$AL(F)$ can be seen to represent both the reachable states of the FIFO system and the annotated traces which in some sense witness the reachability of these states. Thus, $AL(F)$ is a suitable candidate for the language to use in the verification procedure shown in Figure 1.

Given a string l in Σ^*, we say that l is well-formed if l ends with a symbol from T_Q and there is no other occurrence of symbols from T_Q. We say that a language L is well-formed if all strings in L are well-formed. For a well-formed string l ending in symbol t_q, let $\mathcal{T}(l)$ denote the prefix of l without t_q and let $\mathcal{C}(l)$ denote the control state q.

3.2 Answering membership queries

In order to answer a membership query for $AL(F)$, given a string l in Σ^* we need to verify if l is a correct annotation for some valid sequence of transitions l' in $L(F)$. Let $\mathcal{A}^{-1}(l)$ be a function which gives the set (possibly empty) of all sequences of transitions l' for which $\mathcal{A}(l') = l$. First, if l is not well-formed, $\mathcal{A}^{-1}(l) = \emptyset$ since all valid annotations are clearly well-formed. Assuming l is well-formed, if we ignore the bars in $\mathcal{T}(l)$, we get a string l'' which could potentially be in $\mathcal{A}^{-1}(l)$ except that the transitions corresponding to any receives are missing. We can identify the possible missing receive transitions by looking at the barred symbols in $\mathcal{T}(l)$; each barred send can potentially be matched by a receive transition that operates on the same channel and has the same letter. However, we do not know the exact positions where these receive transitions are to be inserted in l''. We can try all possible (finitely many) positions and simulate each resulting transition sequence on the fly on the FIFO system. Any transition sequence which is valid on the FIFO and gives back l on application of $\mathcal{A}$ is then a member of $\mathcal{A}^{-1}(l)$. If $\mathcal{A}^{-1}(l) \neq \emptyset$ then l is a valid annotated trace.

For illustration, let us consider a membership query for the string $\overline{\theta_1\theta_3}\theta_1 t_{q_2}$ for the FIFO automata shown in Figure 2. We identify the possible missing receive transitions as two instances of θ_2. Since a receive can only occur after a send for the same channel and letter, the possible completions of the input string with receives are $\{\theta_1\theta_2\theta_3\theta_2\theta_1, \theta_1\theta_2\theta_3\theta_1\theta_2, \theta_1\theta_3\theta_2\theta_2\theta_1, \theta_1\theta_3\theta_2\theta_1\theta_2, \theta_1\theta_3\theta_1\theta_2\theta_2\}$. Of these, $\theta_1\theta_2\theta_3\theta_1\theta_2$ can be correctly simulated on the FIFO system and gives back the input string $\overline{\theta_1\theta_3}\theta_1 t_{q_2}$ on application of $\mathcal{A}$. Therefore, the answer to the membership query is *yes*. An example for a negative answer is $\overline{\theta_1}t_{q_0}$.

3.3 Answering equivalence queries

For learning $AL(F)$ in the active learning framework, we need a method to verify whether a supposed language L of annotated traces is equivalent to $AL(F)$. If not, then we also need to identify a string in the symmetric difference of $AL(F)$ and L to allow the learner to make progress.

Given a string $l \in L$ and a transition θ in the FIFO, we can find if it is possible to *extend* l using θ. More precisely, we define a function $Post(l, \theta)$ as follows. If l is well-formed, let $source(\theta)$ and $target(\theta)$ be the control states which are respectively the source and the target of θ.

$$Post(l,\theta) = \begin{cases} \emptyset & \text{if } l \text{ not well-formed or if } \mathcal{C}(l) \neq source(\theta) \\ \{\mathcal{T}(l)\theta\, t_{\mathrm{target}(\theta)}\} & \text{otherwise if } \delta(\theta) = \tau \text{ or } \delta(\theta) = c_i!a_j \\ \{deriv(\mathcal{T}(l),\theta)\, t_{\mathrm{target}(\theta)}\} & \text{otherwise if } \delta(\theta) = c_i?a_j \end{cases}$$

$deriv(\mathcal{T}(l), \theta)$ checks the first occurrence of a send θ_s in $\mathcal{T}(l)$ for channel c_i and if the send is for the character a_j, replaces θ_s with $\overline{\theta_s}$. $deriv(\mathcal{T}(l), \theta)$ is empty if no such θ_s could be found or if θ_s outputs a character other than a_j. Intuitively, *deriv* is similar to the concept of the derivative in formal language theory, except that we look at only the channel that θ operates upon.

Let $Post(l)$ be $\bigcup_{\theta \in \Theta} Post(l, \theta)$ and $Post(L)$ be $\bigcup_{l \in L} Post(l)$.

Theorem 1. *Let $\mathcal{F}(L) = Post(L) \cup \{t_{q_0}\}$ where q_0 is the initial control state. $\mathcal{F}(L)$ is a monotone set operator,* i.e. *it preserves set-inclusion. Moreover, $AL(F)$ is the least fixpoint of the functional $\mathcal{F}(L)$.*

The proofs can be found in the Appendix. Theorem 1 gives us a method for answering equivalence queries for $AL(F)$ in one direction. If L is not a fixpoint, it cannot be equivalent to $AL(F)$. In this case, we can also find a string in $L \oplus AL(F)$ as required for Angluin's algorithm. Here, $A \oplus B$ denotes the symmetric difference of two sets. Consider the following cases:

1. $\mathcal{F}(L) - L \neq \emptyset$. Let l be some string in this set. If l is t_{q_0} then it is in $AL(F) \oplus L$. Otherwise, we can check if l is a valid annotation using the procedure described in Section 3.2. If yes, then l is in $AL(F) \oplus L$. Otherwise, it must be true that $l \in Post(l')$ for some $l' \in L$. If l is not valid, l' cannot be valid since $Post()$ of a valid annotation is always valid. Hence $l' \notin AL(F)$ or $l' \in AL(F) \oplus L$.
2. $\mathcal{F}(L) \subsetneq L$. From standard fixpoint theory, since $AL(F)$ is the least fixed point under $\mathcal{F}$, it must be the intersection of all prefixpoints of $\mathcal{F}$ (a set Z is a prefixpoint if it *shrinks* under the functional $\mathcal{F}$, *i.e.* $\mathcal{F}(Z) \subseteq Z$). Now, L is clearly a prefixpoint. Applying $\mathcal{F}$ to both sides of the equation $\mathcal{F}(L) \subsetneq L$ and using monotonicity of $\mathcal{F}$, we get $\mathcal{F}(\mathcal{F}(L)) \subsetneq \mathcal{F}(L)$. Thus, $\mathcal{F}(L)$ is also a prefixpoint. Let l be some string in the set $L - \mathcal{F}(L)$. Since l is outside the intersection of two prefixpoints, it is not is in the least fixpoint $AL(F)$. Hence, l is in $AL(F) \oplus L$.
3. $\mathcal{F}(L) = L$. Let $\mathcal{W}(L)$ be the set of annotated traces in L which can reach unsafe states (We will describe how $\mathcal{W}(L)$ is computed in the next section). If $\mathcal{W}(L)$ is empty, since L is a fixpoint, we can abort the learning procedure and declare that the safety property holds. For the other case, if $\mathcal{W}(L)$ is not empty then let l be some annotated trace in this set. We check if l is a valid annotation using the procedure described in Section 3.2. If it is valid, we have found a valid counterexample and can again abort the whole learning procedure since we have found an answer (in the negative) to the safety property verification. Otherwise, l is in $AL(F) \oplus L$.

A subtle point to note is that although we attempt to learn $AL(F)$, because of the limitation in the equivalence query, the final language obtained after the termination of the verification procedure may not be $AL(F)$. It might be some fixpoint which contains $AL(F)$ or it might be simply some set which contains a valid annotated trace demonstrating the reachability of some unsafe state. However, this is not a cause for concern to us since in all cases the answer for the safety property verification is correct.

3.4 Finding annotated traces leading to unsafe states

In the previous section, we referred to a set $\mathcal{W}(L)$ in L which can reach unsafe states. We now show how this can be computed.

We assume that for each control state $q \in Q$, we are given a recognizable set [6] describing the unsafe channel configurations. Equivalently, for each q, the unsafe channel contents are given by a finite union of products of regular languages: $\bigcup_{0 \leq i \leq n_q} P_{q,i}$ where

$P_{q,i} = \prod_{0 \leq j \leq k} U_q(i, c_j)$ and $U_q(i, c_j)$ is a regular language for contents of channel c_j. For each $P_{q,i}$, an unsafe state s_u is some $(q, u_0, u_1, \dots u_k)$ such that $u_j \in U_q(i, c_j)$.

For a channel c, consider a function $h_c : \Sigma \to M^*$ defined as follows:

$$h_c(t) = \begin{cases} m \text{ if } t \in \Theta \text{ and } \delta(t) = c!m \\ \epsilon \ \ \text{otherwise} \end{cases}$$

Let h_c also denote the unique homomorphism from Σ^* to M^* that extends the above function.

Let L_q be the subset of an annotated trace set L consisting of all well-formed strings ending in t_q, *i.e.* $L_q = \{l \mid l \in L \text{ and } \mathcal{C}(l) = q\}$.

If an unsafe state $s_u = (q, u_0, u_1, \dots u_k)$ is reachable, then there must exist a sequence of transitions $l_\theta \in \Theta^*$ such that $s_0 \xrightarrow{l_\theta} s_u$, where s_0 is the initial state. In l_θ, if the receives and the sends which match the receives are taken out, only the remaining transitions which are sends can contribute to the channel contents in s_u. Looking at the definition of h_c, it can be seen that for each channel content u_j in s_u, $u_j = h_{c_j}(\mathcal{A}(l_\theta))$ (recall that $\mathcal{A}$ converts a sequence of transitions into an annotated trace). Thus, for s_u to be reachable, there must be some annotated trace $l \in AL(F)$ such that $s_u = (\mathcal{C}(l), h_{c_0}(l), h_{c_1}(l), \dots, h_{c_k}(l))$.

Let $h^{-1}_{c_j}(U_q(i, c_j))$ denote the inverse homomorphism of $U_q(i, c_j)$ under h_{c_j}. For each $P_{q,i}$, $\bigcap_{0 \leq j \leq k} h^{-1}_{c_j}(U_q(i, c_j))$ gives a set of annotated strings which can reach the unsafe channel configurations for control state q. Intersecting this with L_q verifies if any string in L can reach these set of unsafe states. If we perform such checks for all control states for all $P_{q,i}$, we can verify if any unsafe state is reached by L. Thus, the set of annotated traces in L that can lead to an unsafe state is given by:

$$\mathcal{W}(L) = \bigcup_{q \in Q} (\bigcup_{0 \leq i \leq n_q} (L_q \cap \bigcap_{0 \leq j \leq k} h^{-1}_{c_j}(U_q(i, c_j))))$$

We summarize the verification algorithm in Figure 3.

Theorem 2. *For verifying safety properties of FIFO automata, the learning to verify algorithm satisfies the following properties:*

1. *If an answer is returned by algorithm, it is always correct.*
2. *If $AL(F)$ is regular, the procedure is guaranteed to terminate.*
3. *The number of membership and equivalence queries are at most as many as needed by Angluin's algorithm. The total time taken is bounded by a polynomial in the size of the minimal automaton for $AL(F)$ and linear in the time taken for membership queries for $AL(F)$.*

For the proof and the details of the complexity analysis, the reader is referred to the Appendix.

4 Generalization to other infinite state systems

The verification procedure described for FIFO automata can be easily generalized to other infinite state systems. The challenge for each class of system is to identify the alphabet Σ which provides an annotation enabling the following:

```
algorithm learner
begin
Angluin's $L^*$ algorithm
end

algorithm isMember
Input: Annotated trace $l$
Output: is $l \in AL(F)$?
begin
  if $l$ not well-formed return no
  else
    find receives matching barred symbols
    find possible positions for receives
    simulate resulting strings on FIFO
      system on the fly
    if any string reaches $\mathcal{C}(l)$ with
      correct annotation, return yes
  return no
end
```

```
algorithm Equivalence Check
Input: Annotated trace set $L$
Output: is $L = AL(F)$?
If not, then some string in $L \oplus AL(F)$
begin
  $\mathcal{F}(L) = Post(L) \cup \{t_{q0}\}$
  if $\exists l \in (\mathcal{F}(L) - L)$
    if isMember($l$)
      return (no, $l$)
    else
      return (no, $l'$ where $l = Post(l')$)
  else if $\mathcal{F}(L) \subsetneq L$
    return (no, $l \in (L - \mathcal{F}(L))$)
  else if $\exists l \in \mathcal{W}(L)$
    if isMember($l$)
      Print (safety prop. does not hold, $l$); stop
    else
      return (no, $l$)
  else
    Print (safety prop. holds); stop
end
```

Fig. 3. Learning to verify algorithm

- membership query for the annotated trace language
- function to compute $Post()$ for a given annotated set
- function to find if a string in an annotated set can reach an unsafe state

Notice that the verification procedure does not assume anything else about FIFO automata other than the above functions. In fact, the learning algorithm does not have to be limited to regular languages; any suitable class of languages can be used if the needed decision procedures are available.

5 Implementation

We have updated the Lever (LEarning to VERify) tool suite first introduced in [24] with the active learning based verification procedure for FIFO automata. The tool is written in Java and is available from [17]. For general automata related decision procedures, we use the Java package `dk.brics.automata` available from [18].

We have used Lever to analyze some canonical FIFO automata verification problems: *Producer Consumer, Alternating bit protocol* and *Sliding window protocol* (window size and maximum sequence number 2). The above examples are fairly well-known in the FIFO research community; for details the reader is referred to [22]. Table 1 shows the results obtained. We compare the number of states of the final automaton (Size) and the running times (T) using the verification procedure in this paper with the procedure we used earlier in [24] (columns Size_{old} and T_{old}). It can be seen that there is an improvement using the new procedure (although the comparison of Size should be taken with the caveat that the annotation in the two procedures is slightly different). All executions were done on a 1594 MHz notebook computer with 512 MB of RAM using Java virtual machine version 1.4.1 from Sun Microsystems. We also report the time taken (T_{rmc}) by the regular model checking tool [19] on the same examples. Although a complete comparative analysis with all available tools remains to be done, it can be seen the running time of Lever is slightly better than the regular model checking tool.

	Size	T	Size_{old}	T_{old}	T_{rmc}
Producer Consumer	7	0.3s	20	0.4s	3.3s
Alternating Bit	33	2s	104	4.1s	24.7s
Sliding Window	133	54s	665	81.2s	78.4s

Table 1. Running time

6 Related Work

Verification of infinite state systems: For automatic verification of infinite state FIFO systems, the state space has to be represented by symbolic means. Some common representations are: regular sets [9, 1], Queue Decision Diagrams [7], semi-linear regular expressions [14] and constrained QDDs [8]. Since an iterative approach of computing the fixpoint for reachability does not terminate for most cases, various mechanisms are used for finding the reachable set. We now discuss some of these techniques and show their relation to our learning approach.

In the approach using *meta-transitions* and *acceleration* [7, 8, 14], a sequence of transitions, referred to as a *meta-transition*, is selected and the effect of its infinite iteration calculated. This is complementary to our learning approach, since meta-transitions can be also be incorporated into our learning algorithm. Another popular approach is that of *regular model checking* [9, 1]. A regular set is used to represent the states and a transducer is used to represent the transition relation. The problem is reduced to finding a finite transducer representing the the infinite composition of this relation. However, there are some examples for which even if such a finite transducer exists, the procedure may not be able to converge to it. One such FIFO automaton is shown in Figure 2. We used the regular model checking tool from [19] to analyze this example, but the tool failed to terminate even after two hours. On the other hand, our learning-based tool is able to automatically find the reachable set in less than half a second. It is certainly possible that in other examples, transducer construction may be able to find the reachable region faster. Thus, our approach can be seen as complementary and seen to extend the range of systems that can be automatically analyzed.

An approach for computing the reachable region that is closely related to ours is *widening*. In this approach, the transition relation is applied to the initial configuration some number of times and then by comparing the sets thus obtained, the limit of the iteration is guessed. A simple widening principle in the context of regular model checking is given in [9] which is extended in [23] for parametric systems. Bultan [10] and Bartzis *et al.* [5] present widening procedures for Presburger formulas and arithmetic automata respectively. At a very high level, both *widening* and our approach use similar ideas. In both methods, based on certain sample points obtained using the transitions, a guess is made for the fixpoint being searched for. One important difference between widening and our approach is that widening (except for certain special contexts where it can be shown to be exact) is a mechanism to prove the correctness of a system and cannot be used to prove a system to be incorrect. On the other hand, the approach presented here allows one to both prove a system to be correct and to detect bugs.

Use of machine learning for verification: We introduced the learning to verify approach in [24]. In that work, we learned the reachable set from sample runs of the FIFO system using an algorithm for learning regular sets called RPNI [20]. We were restricted to a learning framework which cannot answer queries but instead simply presents positive and negative examples of the concept to be learnt. The active learning framework that we have used in this paper has some advantages over the framework used earlier. First, we need only polynomial number of samples as opposed to exponential that we needed before, significantly reducing the space requirements of the algorithm. Second, we learn the optimal automaton for the annotated traces. Our experiments also show that the new method uses less running time. Finally, an important advantage of the present work is in the annotation scheme used to collect traces witnessing reachability; with the new annotation scheme many more FIFO systems have regular set of annotated traces, thus enlarging the class of systems that can be provably verified.

In other work applying machine learning to verification, Peled *et al.* [21] give a method called *Black Box Checking* which is extended by Groce *et al.* [16] as *Adaptive Model Checking*. Briefly, in this method, one starts with a possibly inaccurate model and incrementally updates it using Angluin's [3] query based learning of regular sets. Cobleigh *et al.* [11] also use a variant of Angluin's algorithm to learn the assumptions about the environment to aid compositional verification. Boigelot *et al.* [4] present a technique for constructing a finite state machine that simulates all observable operations of a given reactive program. Ammons *et al.* [2] use machine learning to discover formal specifications of the protocols that a client of an application program interface must observe. Edelkamp *et al.* [12] consider the problem of finding "bad" states in a model as a directed search problem and use AI heuristic search methods to attempt to find these states. Ernst *et al.* [13] have developed a system called *Daikon* which attempts to discover likely invariants in a program by analyzing the values taken by its variables while the program is exercised in a test suite.

Our approach in using the machine learning techniques for verification is unique in that we are not trying to learn an unknown system model but rather the behavior of a system which is already fully described. This is closest in spirit to Ernst *et al.* [13], although the domain of application and objective are completely different.

7 Conclusion

We have presented a machine learning based approach to verify finite state machines communicating over unbounded FIFO channels. We use Angluin's L* algorithm for learning a regular representation of the reachable states and their witnesses. We show that the verification procedure is sound and also complete if the annotated trace language is regular and provide bounds for the running time and space.

The *learning to verify* approach may be applied to other infinite systems such as: automata with unbounded integers; real-time and hybrid systems; parameterized systems; counter automata; probabilistic systems; and push-down automata with multiple stacks. Another interesting direction is to extend this approach to verifying liveness properties.

References

1. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Algorithmic improvements in regular model checking. In *Computer-Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 236–248. Springer, 2003.

2. G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. *ACM SIGPLAN Notices*, 37(1):4–16, Jan. 2002.
3. D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, Nov. 1987.
4. B. Boigelot and P. Godefroid. Automatic synthesis of specifications from the dynamic observation of reactive programs. In E. Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 321–334, Enschede, The Netherlands, 1997. Springer Verlag, LNCS 1217.
5. C. Bartzis and T. Bultan. Widening arithmetic automata. In *Computer Aided Verification'04 (to appear)*, 2004.
6. J. Berstel. *Transductions and Context-Free-Languages.* B.G. Teubner, Stuttgart, 1979.
7. B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Collection des Publications de la Faculté des Sciences Appliquées de l'Université de Liége, 1999.
8. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2):211–250, June 1999.
9. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 403–418. Springer, 2000.
10. T. Bultan. *Automated symbolic analysis of reactive systems.* PhD thesis, Dept. of Computer Science, University of Maryland, College Park, Md., 1998.
11. J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 331–346, 2003.
12. S. Edelkamp, A. Lafuente, and S. Leue. Protocol verification with heuristic search. In *AAAI Symposium on Model-based Validation of Intelligence*, 2001.
13. M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *International Conference on Software Engineering (ICSE'99)*, pages 213–224, 1999.
14. A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems: Application to FIFO automata. *Information and Computation*, 181(1):1–31, 2003.
15. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
16. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 357–371, 2002.
17. LEVER. Learning to verify tool. http://osl.cs.uiuc.edu/~vardhan/lever.html, 2004.
18. A. Møller. `dk.brics.automaton`. http://www.brics.dk/~amoeller/automaton/, 2004.
19. M. Nilsson. `Regular model checking tool`. http://www.regularmodelchecking.com, 2004.
20. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, Singapore, 1992.
21. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. In *FORTE/PSTV, Beijing, China*, 1999.
22. A. S. Tanenbaum. *Computer Networks, 2nd Ed.* Prentice-Hall, Englewood Cliffs, NJ, 1989.
23. T. Touili. Regular model checking using widening techniques. In *ENTCS*, volume 50. Elsevier, 2001.
24. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. Technical Report UIUCDCS-R-2004-2445, UILU-ENG-2004-1747, http://osl.cs.uiuc.edu/docs/sub2004vardhan/cfsmLearn.pdf, Department of Computer Science, University of Illinois at Urbana-Champaign, 2004.

A Proofs

Proposition 1. *The set of FIFO automata for which $AL(F)$ is regular is strictly larger than the set of FIFO automata for which $AL^{\text{old}}(F)$ is regular.*

Proof (Sketch). If $AL^{\text{old}}(F)$ is regular, let D be the DFA for it. Now create a new DFA D' by making a copy of D and adding one more state s_{new}. Further, for any final state s_{final} in D' add a transition t_q from s_{final} to s_{new}. Here, q is the target control state for all transitions incoming on s_{final} (with *bars* ignored) and t_q is the symbol in T_Q for q. Make s_{new} the only final state in D'. It is easy to see that D and D' are essentially the *same* except that we have explicitly added symbols for the control state that any trace accepted by D ends with. We can now create a finite automaton for $AL(F)$ by replacing all barred receives in D' with ϵ transitions. This shows that $AL(F)$ is regular.

It can be shown that $AL(F)$ for the automaton in Figure 2 is regular while $AL^{\text{old}}(F)$ is not. Thus, the set of FIFO automata for which $AL(F)$ is regular is strictly larger.

Lemma 1. *Each string in $AL(F)$ is either t_{q_0} or in $Post(l)$ for some $l \in AL(F)$*

Proof (Sketch). A string l' is in $AL(F)$ because it is the annotation of at least one sequence of transitions ρ in $L(F)$, *i.e.* $l' = \mathcal{A}(\rho)$. If ρ is the empty string then $l' = \mathcal{A}(\epsilon) = t_{q_0}$. Otherwise, let ρ_{-1} be the prefix of ρ without the last transition. Consider $l = \mathcal{A}(\rho_{-1})$. From the definition of $Post()$, it is easy to see that $l' = Post(l)$.

Theorem 1. *Let $\mathcal{F}(L) = Post(L) \cup \{t_{q_0}\}$ where q_0 is the initial control state. $\mathcal{F}(L)$ is a monotone set operator,* i.e. *it preserves set-inclusion. Moreover, $AL(F)$ is the least fixpoint of the functional $\mathcal{F}(L)$.*

Proof (Sketch). Since $Post(L)$ is simply the union of $Post()$ of all strings in L, monotonicity of $\mathcal{F}$ is immediate.

From the definition of $Post()$, we can see that if $l \in AL(F)$, it is also true that $Post(l) \in AL(F)$. This implies $\mathcal{F}(AL(F)) \subseteq AL(F)$ since a string in $\mathcal{F}(AL(F))$ is either t_{q_0} or $Post(l)$ for some $l \in AL(F)$. By Lemma 1, $\mathcal{F}(AL(F)) \supseteq AL(F)$ since any string in $AL(F)$ has to be $Post(l)$ for some other $l \in AL(F)$. Thus, $AL(F)$ is a fixpoint for $\mathcal{F}$.

To see that $AL(F)$ is also the least fixpoint, by way of contradiction assume a (strictly) smaller fixpoint L'. Applying $Post()$ to some string either increases its length by one or increases the number of barred symbols in it. Therefore, given a finite string l it is not possible to have an infinite chain $l_0, l_1, l_2, \ldots$ with $l = l_0$ such that $Post(l_{i+1}) = l_i$ Let l be some string in $AL(F)$ which is not in L'. By Lemma 1, there must be some l_1 such that $Post(l_1) = l$. Now l_1 can be t_{q_0} or be $Post(l_2)$ some l_2. Since this chain of $l_1, l_2 \ldots$ cannot be infinite, it has to end in t_{q_0}. Clearly, t_{q_0} is in any fixpoint, hence $t_{q_0} \in L'$. Consider the smallest i for which l_i in the chain is not in L'. But since $l_{i-1} = Post(l_i)$, l_{i-1} has to be in L' giving a contradiction. Hence, $AL(F)$ is the least fixpoint of $\mathcal{F}$.

Theorem 2. *For verifying safety properties of FIFO automata, the learning to verify algorithm satisfies the following properties:*

1. *If an answer is returned by algorithm, it is always correct.*
2. *If $AL(F)$ is regular, the procedure is guaranteed to terminate.*
3. *The number of membership and equivalence queries are at most as many as needed by Angluin's algorithm. The total time taken is bounded by a polynomial in the size of the minimal automaton for $AL(F)$ and linear in the time taken for membership queries for $AL(F)$.*

Proof (Sketch). Soundness of the procedure is straightforward. We declare that the safety property holds only when we have found a fixpoint L which does not intersect with the "unsafe" traces. Since L is a fixpoint, it must be larger or equal to $AL(F)$ which is the least fix point. If $\mathcal{W}(L)$ is empty then $\mathcal{W}(AL(F))$ must also be empty implying that no execution of the system can reach an unsafe state.

If we say that the safety property does not hold and provide a path leading to an unsafe state, this counterexample has to be valid since we always first check it against the FIFO system.

For showing completeness, assuming that $AL(F)$ is regular, we can rely on the termination guarantees of Angluin's algorithm. The only caveat is our limited ability to answer equivalence queries. Consider a hypothetical teacher which can answer all membership and equivalence queries for $AL(F)$ correctly. For an equivalence query, whenever our teacher says *no* the hypothetical teacher must also say *no*; however our teacher is unable to decide when to say *yes*. Notice that a *yes* answer to an equivalence query is only given once and marks the end of the algorithm. Imagine a session of a learner with the hypothetical teacher and a parallel session of another learner with our limited teacher. Further, assume that in case the answer to the equivalence query is *no*, the hypothetical teacher returns the same string (for the symmetric difference) as our teacher. Both learners start off by making identical queries and proceed in lock step since both teachers provide the same answers. Let us say that at some point, our teacher declares that it has solved the verification problem and aborts the learning procedure. Consider the two cases possible:

- Our teacher finds some fixpoint which does not intersect with the unsafe states. In this case, the hypothetical teacher might still continue if the fixpoint is not the least fixpoint ($AL(F)$) or say *yes* if is the least fixpoint. The hypothetical teacher could not have said *yes* earlier, since if the learner proposed $AL(F)$, our teacher would have found it as a fixpoint which does not intersect with the unsafe states.
- Our teacher finds a valid counterexample to the safety property. Again the hypothetical teacher could not have said *yes* earlier. If $AL(F)$ had been proposed by the learner, since the safety property does not hold, some string in $AL(F)$ is a valid counterexample to the safety property and our teacher would have found it.

Thus, in all cases our teacher will end the learning procedure sooner or at the same time as the hypothetical teacher.

Complexity analysis: As shown in [3], Angluin's algorithm makes $O(|\Sigma|mn^2)$ membership queries and $O(n)$ equivalence queries. Here m is the size of the longest string returned by the teacher in a negative answer to an equivalence query and n is the size of the minimal automaton representing $AL(F)$ (assuming $AL(F)$ is regular). When we answer an equivalence query for a hypothesis L, we search for a string in $L \cap (\neg Post(L))$, $Post(L) \cap (\neg L)$ and $\mathcal{W}(L)$. The size of $Post(L)$ is bounded by $O(|\Theta|n)$, hence $L \cap (\neg Post(L))$ and $Post(L) \cap (\neg L)$ are at most of size $O(|\Theta|n^2)$. Assuming that the unsafe states are described by an automaton smaller than the minimal one for $AL(F)$, $|\mathcal{W}(L)|$ is bounded by $O(n^2)$. Therefore, the longest string that can be returned as an answer to the equivalence query is $O(|\Theta|n^2)$. Hence, $m = O(|\Theta|n^2)$.

Let $T(l,k)$ be the time taken for a membership query for a string of length l on a FIFO automata with k receive transitions. The running time for the verification procedure is dominated by the cost of equivalence and membership queries. Let us now consider these in turn.

1. Equivalence queries: Each equivalence query can also result in a membership query. Following the reasoning in the previous paragraph, the cost of one equivalence query is bounded by $O(m + T(m,k))$ which can be simplified to $O(T(|\Theta|n^2, k))$. For maximum of n such queries, the total cost is $O(nT(|\Theta|n^2, k))$
2. Membership queries from learner: For $O(|\Sigma|mn^2)$ membership queries with the maximum length of a string being $O(m+n)$, the bound on the cost is $O(|\Sigma||\Theta|n^4T(|\Theta|n^2, k))$

The cost of answering the membership queries from learner clearly dominates the total cost. Thus, the running time is $O(|\Sigma||\Theta|n^4T(|\Theta|n^2, k))$ which is a polynomial in the size of the minimal automata for $AL(F)$ and the time needed for a membership query for $AL(F)$. The longest string for which membership may need to be checked is quadratic in the size of the minimal automata.

The space requirements for the verification procedure consist of the observation table and the space requirement for membership queries. As shown in [3], the observation table needs $O(|\Theta|(m^2n^2 + mn^3))$ space which reduces to $O(|\Theta|^3n^6)$.

Let us now consider $T(l,k)$, the cost of a membership query for a string of length l. This depends strongly on the annotation scheme used. For instance, in the old annotation scheme (which keeps both parts of a send-receive pair) this is simply $O(l)$. For the new annotation scheme, we drop the receive part to allow more FIFO systems to have regular annotated trace languages (making them amenable to automatic analysis). However, this forces us to do more work for the membership query. There can be at most l receive transitions that have to be inserted in the queried string to get a trace that can be simulated on the FIFO system. A trivial upper bound for $T(l,k)$ can be derived as follows. For a query string of length l, we may have to add l more receives. The receives can be put in the $l+1$ positions in $(l+1)^l$ or $O(l^l)$ ways. For each place, the number of choices for receives is at most equal to the minimum of k and l. Let $p = \min(k,l)$. Then, the cost of a membership query is $O(p^l l^l)$. The bound could possibly be improved but this is deferred for future work. It is also easy to see that the space requirement for a membership query is simply $O(\log(p^l l^l))$.