Lower Bounds for Active Automata Learning*

Loes Kruger LOES.KRUGER@RU.NL
Bharat Garhewal B.GARHEWAL@CS.RU.NL
Frits Vaandrager F.VAANDRAGER@CS.RU.NL
Institute for Computing and Information Sciences, Radboud University, Nijmegen, the Netherlands

Abstract

We study lower bounds for the number of output and equivalence queries required for active learning of finite state machines, with a focus on $L^\#$, a new learning algorithm that requires fewer queries for learning than other state-of-the-art algorithms on a large collection of benchmarks. We improve the lower bound of Balcázar et al. (1997) on the combined number of output and equivalence queries required by any learning algorithm, and give a simpler proof. We prove that in the worst case $L^\#$ needs $n-1$ equivalence queries to learn an FSM with n states, and establish lower bounds on the number of output queries needed by $L^\#$ in the worst case. In practical applications, the maximum length of the shortest separating sequence for all pairs of inequivalent states (MS3) is often just 1 or 2. We present $L^\#_h$, a version of $L^\#$ with bounded lookahead h, which learns FSMs with an MS3 of at most h without requiring any equivalence queries, and give lower and upper bounds on its complexity.

Keywords: query learning; active automata learning; finite state machines; complexity; lower bounds; $L^\#$.

1. Introduction

Query learning was introduced by Angluin (1987a) and is currently one of the most important frameworks for learning finite state machines. In query learning, a *learner* obtains information about the concept to learn by making *queries* to some *teacher*, instead of passively receiving examples. In the setting considered by Angluin (1987a), the teacher knows a fixed regular language T over a given alphabet. The goal of the learner is to come up with a deterministic finite automaton (DFA) accepting T by asking two types of queries. In a *membership query*, the learner presents a word x and asks whether x is in T; the teacher answers YES or NO. In an *equivalence query*, the learner produces a DFA $\mathcal{H}$, and asks whether $L(\mathcal{H}) = T$; the teacher either answers YES if this is the case, or else returns a *counterexample*, i.e., any string witnessing $L(\mathcal{H}) \neq T$. Angluin (1987a) presented the L^* algorithm, which efficiently learns a DFA using a polynomial number of membership and equivalence queries. Angluin (1987b, 1990) also showed that neither membership queries alone nor equivalence queries alone are good enough to learn DFAs with a polynomial number of queries. Thus the L^* algorithm lives in a Goldilocks zone where efficient learning is possible. Balcázar et al. (1994, 1997) present lower bounds on the combined number of membership and equivalence queries needed for learning, but also leave several open problems.

* Research supported by NWO TOP project 612.001.852 "Grey-box learning of Interfaces for Refactoring Legacy Software (GIRLS)". This article is partly based on the MSc thesis of the first author.

Since 1987, many other learning algorithms have been discovered in the Goldilocks zone that improve upon L^*. The most efficient of these algorithms all have an asymptotic query complexity of $\mathcal{O}(kn^2 + n \log m)$ and pose at most $n - 1$ equivalence queries, where k is the number of input symbols, n is the number of states of the minimal DFA, and m is the length of the longest counterexample provided in response to an equivalence query; see e.g., Rivest and Schapire (1989, 1993); Howar (2012); Isberner et al. (2014); Isberner (2015); Frohme (2019); Vaandrager et al. (2022). Query learning has found numerous applications in the area of software and hardware analysis. We refer to Vaandrager (2017); Howar and Steffen (2018) for surveys and further references. Many of the recent learning algorithms and applications are presented in a setting of finite state machines (FSMs, a.k.a. Mealy machines), a variation of DFAs where instead of marking states as accepting we associate output symbols to transitions. In this setting, membership queries are replaced by *output queries* where the learner asks for the outputs in response to a sequence of inputs. However, algorithms and complexity results can be easily translated from one setting to the other.

There have been discussions about whether the number of queries is the right way to measure the complexity of learning algorithms. Some authors suggest to also measure the *symbol complexity*, which is the total number of input symbols and resets required to learn an automaton, see e.g. Isberner (2015). This is a relevant measure for practical learning scenarios, as the time required for realizing an output query asymptotically grows at least linearly in its length. Groz et al. (2020) recently proposed the hW-inference algorithm that is able to learn strongly connected FSMs with just a *single* long output query, in combination with a few equivalence queries en route. Clearly, the number of queries is not a meaningful measure of the complexity of hW-inference. Groz et al. (2020) use *adaptive* output queries in which the choice of an input symbol may depend on the outputs received in response to previous inputs. Whereas L^* and many older learning algorithms only use *preset* membership/output queries in which all the inputs are fixed in advance, adaptive output queries are effectively used in the fastest learning algorithms of today (Frohme, 2019; Vaandrager et al., 2022). However, we do not know if and how lower bounds on complexity are affected by the use of adaptive queries. In applications, equivalence queries are often approximated by using conformance testing algorithms, which generate a large number of tests (output queries) in order to find a counterexample for a hypothesis model. Typically, for larger benchmarks the number of output queries and symbols required for testing dominates those required for learning, and so the combined number of output queries needed for both learning and testing appears to be a sensible measure of complexity in these cases, see e.g. (Aslam et al., 2020; Vaandrager et al., 2022).

Given the significant advances in the theory of active automata learning during the last 25 years, the numerous applications of this theory, and discussions about how complexity should be measured, we believe it is important to revisit the open problems concerning lower bounds on query complexity of Balcázar et al. (1997), to also establish lower bounds for learning algorithms w.r.t. other complexity measures and types of queries, and to try to close the gap between these lower bounds and the asymptotic complexity of state-of-the-art algorithms. In this article, we report on some progress on these challenging problems. We describe our results in a setting of FSMs and the $L^\#$ algorithm of Vaandrager et al. (2022), but believe they can be transferred to the setting of DFAs and to other learning algorithms. More specifically, we present the following results:

1. We improve the lower bound of Balcázar et al. (1997) on the combined number of output and equivalence queries required for learning, and give a simpler proof.

2. We prove that in the worst case $L^\#$ needs $n-1$ equivalence queries to learn an FSM.

3. We establish lower bounds on the number of output queries needed by $L^\#$ in the worst case. Our bounds take the maximal length of counterexamples into account.

4. We present $L^\#_h$, a version of $L^\#$ that learns FSMs, with a bounded lookahead of at most h, without any equivalence queries, and analyze its complexity.

The rest of this article is organized as follows. We start with a preliminary Section 2 that introduces some basic concepts and the $L^\#$ algorithm. Section 3 presents our new lower bound results. Section 4 motivates and explains the new $L^\#_h$ algorithm, and presents some experimental results. Finally, Section 5 concludes the paper and outlines some directions for future work. Due to space limitations, all proofs have been deferred to Appendix A. The complete benchmark results can be found in Appendix B.

2. A Brief Introduction to $L^\#$

In this section, we briefly recall the $L^\#$ algorithm for learning (deterministic, input complete) FSMs, and illustrate it on a toy example of a coffee machine. For a detailed and formal description we refer to Vaandrager et al. (2022).

Figure 1 (right) shows a simple FSM $\mathcal{M}$ with three states q_0, q_1 and q_2, with q_0 designated as the initial state, two inputs 1 and coffee, and three outputs $\surd$, 1 and coffee. For each state and each input, the FSM generates an output and transitions to a target state. Intuitively, this FSM models a strange coffee machine that works as follows. If the

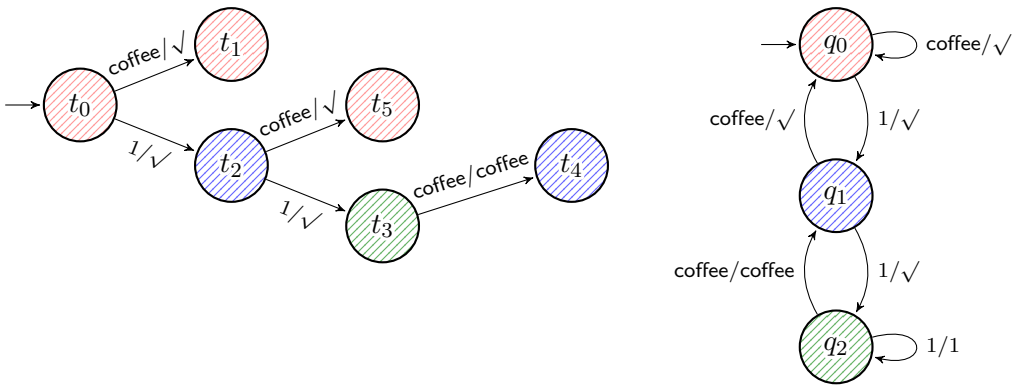

Figure 1: *An observation tree (left) for an FSM model of a coffee machine (right)*

user provides two coins and presses the coffee button, the machine produces coffee. If the user immediately provides another coin and presses the coffee button, the machine produces another coffee. If the user provides more coins than needed, the machine returns all extra coins. The machine produces no visible output, or $\surd$, when the user presses the coffee

button without providing enough coins or when the maximum number of coins for coffee has not been reached yet.

We consider a setting where a teacher knows an FSM, which a learner has to learn by posing two types of queries. Initially, the learner only knows the set of inputs. With an *output query*, the learner asks what the output is in response to an input sequence. In our example, if the learner poses output query "1 coffee" then the teacher will respond with "$\surd$ $\surd$". With an *equivalence query*, the learner may ask whether a hypothesized FSM $\mathcal{H}$ is correct. The teacher will respond with YES if the hypothesis is equivalent to the target FSM (generates the same outputs for any sequence of inputs). Otherwise, the teacher will answer NO and supply a counterexample, an input sequence that shows the difference between $\mathcal{H}$ and the target FSM. For instance, if the learner proposes a hypothesis with a single state and output $\surd$ in response to any input then the teacher will answer NO and may provide counterexample "1 1 1".

The $L^\#$ algorithm can be used by a learner to accomplish its task. $L^\#$ organizes the responses to output and equivalence queries in an *observation tree*, which is just a tree-shaped, partial FSM. Figure 1 (left) shows an observation tree for the example FSM, which stores the results of output queries coffee, 1 coffee, and 1 1 coffee. Each state of an observation tree corresponds to a unique state of the FSM (indicated by the state coloring) but initially, the learner does not know which one. The $L^\#$ algorithm deems two states r and r' in the observation tree *apart*, notation $r \# r'$, if there exists a sequence σ of inputs for which the known responses in both states are different. We write $\sigma \vdash r \# r'$ to indicate that σ witnesses the apartness of r and r'. In our example observation tree, we have coffee $\vdash t_0 \# t_3$, coffee $\vdash t_2 \# t_3$ and 1 coffee $\vdash t_0 \# t_2$. Thus states t_0, t_2 and t_3 are pairwise apart. Since state t_1 has no outgoing transitions it is not apart from any other state in the tree. The $L^\#$ algorithm partitions the states of observation tree $\mathcal{T}$ into three sets:

1. The set of *basis states* S. We require that S contains the initial state of $\mathcal{T}$, and that states from S form a subtree of $\mathcal{T}$. Moreover, states in S are required to be pairwise apart: $\forall p, q \in S, p \neq q : p \# q$.

2. The set of *frontier states* F, consisting of the immediate successors of basis states that are not contained in S.

3. The remaining states $Q \setminus (S \cup F)$.

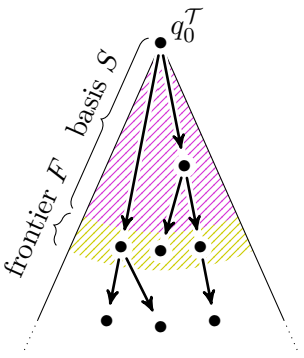

Figure 2: *$\mathcal{T}$, S, and F*

In our example, a possible basis is $S = \{t_0, t_2, t_3\}$, leading to a frontier $F = \{t_1, t_5, t_4\}$, and an empty set of remaining states.

For each state r in the frontier, the *candidate set* $C(r)$ is the set of basis states that are not apart from r, that is, $C(r) = \{q \in S \mid \neg(r \# q)\}$. In our example, we have no information about the behavior of the frontier states, and thus for all frontier states the candidate set is equal to the basis. The $L^\#$ algorithm starts from a trivial observation tree with just a single root node t_0, which constitutes the basis, and repeatedly applies the following rules (slightly simplified for explanatory purposes), in arbitrary order:

(R1) **Promotion:** If frontier F contains a state r with $C(r) = \emptyset$, then r is apart from all states in basis S, and therefore we may move it from F to S. If multiple states may be moved from the frontier to the basis then we may nondeterministically choose any of these states.

(R2) **Extension:** If some state $q \in S$ does not have an outgoing i-transition, for some input i, then the frontier is extended by adding a new state and an i-transition from q to this new state. The output of the new transition is determined via an output query $\mathsf{access}(q)\ i$, where $\mathsf{access}(q)$ is the unique sequence of inputs leading to state q.

(R3) **Identification:** If frontier F contains a state r with a candidate set $C(r)$ that contains at least two elements, say q and q', then we pick a witness σ with $\sigma \vdash q \# q'$ and pose output query $\mathsf{access}(r)\ \sigma$. In the resulting observation tree $q \notin C(r)$ or $q' \notin C(r)$.

(R4) **Equivalence:** If rules (R1), (R2) and (R3) cannot be applied, we construct a hypothesis from $\mathcal{T}$ and pose an equivalence query to the teacher. If the answer is YES then we are done, otherwise we process the received counterexample.

We now show how $L^\#$ learns the FSM of Figure 1 and explain rule (R4) in more detail:

1. Initially, the observation tree has a single node t_0 and basis $\{t_0\}$. The only rule that can be applied is the extension rule (R2). This rule is applied twice (for both inputs) giving us new frontier states t_1 and t_2 (see Figure 3 (left)).

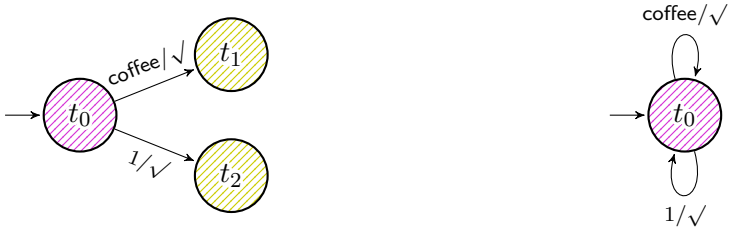

Figure 3: *An observation tree (left) and initial hypothesis $\mathcal{H}_1$ (right)*

2. At this point we cannot apply rules (R1), (R2) and (R3), so we apply rule (R4) and construct a hypothesis $\mathcal{H}$. This is done as follows: the set of states of $\mathcal{H}$ is equal to the basis S, with the root as the initial state; transitions between basis states in $\mathcal{T}$ are included in $\mathcal{H}$; and transitions from a basis state q to a frontier state r are redirected to the unique state in $C(r)$. Phrased differently: each frontier state r is merged with the unique basis state contained in $C(r)$. In the case of Figure 3 (left), $C(t_1) = C(t_2) = \{t_0\}$ and so we obtain the hypothesis $\mathcal{H}_1$ shown in Figure 3 (right). Assume the teacher returns counterexample 1 1 coffee. We add this sequence to the observation tree. Since this does not create any new apartness pair between the frontier and the basis, counterexample processing (which we explain in more detail below) performs an output query for suffix 1 coffee of the counterexample. This leads to the observation tree of Figure 1 (left). Now there is a new apartness pair $t_0 \# t_2$.

3. We promote state t_2 and add it to the basis with rule (R1). Next, since state t_3 is apart from both t_0 and t_2, we apply rule (R1) again and add t_3 to the basis.

4. We complete the frontier and add a 1-transition to a new state t_6- with rule (R2).

5. We repeatedly apply rule (R3) and explore input sequence 1 coffee (that witnesses the apartness of all pairs of basis states) in each of the frontier states. We obtain the extended observation tree of Figure 4 (left), where $C(t_1) = C(t_5) = \{t_0\}$, $C(t_4) = \{t_2\}$, and $C(t_6) = \{t_3\}$.

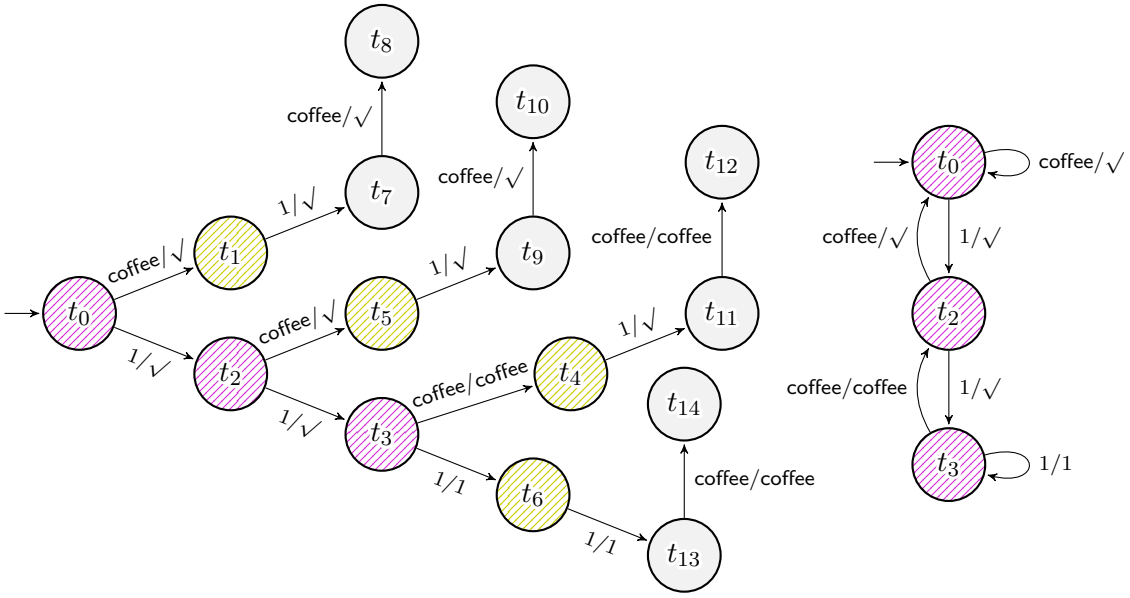

Figure 4: *Final observation tree (left) and hypothesis $\mathcal{H}_2$ (right)*

6. We apply rule (R4) and construct hypothesis $\mathcal{H}_2$ of Figure 4 (right). This hypothesis is clearly equivalent to $\mathcal{M}$ and so learning terminates.

The general procedure for counterexample analysis of $L^{\#}$, illustrated in Figure 5, works as follows: For a hypothesis $\mathcal{H}$ and observation tree $\mathcal{T}$, an input sequence $\sigma \in I^*$ *leads to conflict* if the state r reached by σ in $\mathcal{T}$ is apart from the state q reached by σ in $\mathcal{H}$ (note that by construction any state of $\mathcal{H}$ is also a state of $\mathcal{T}$). A conflict indicates that one of the frontier states was merged with a basis state in $\mathcal{H}$ while they are distinct states in the target automaton, causing a wrong transition in $\mathcal{H}$. Any counterexample decomposes into a concatenation of two words σ and η, where σ leads to a conflict and η witnesses it. Our goal is to pull the conflict back to the frontier, as this will allow us to apply the promotion rule and extend the basis. We define a recursive procedure to analyze σ. Since σ can be very long, we reduce the length of σ using binary search (a trick due to Rivest and Schapire (1993)). If r is in the basis or frontier, then r must be apart from all other states in the basis. This means that $\mathcal{H}$ is not a hypothesis anymore and counterexample analysis is finished. Otherwise, let $\sigma_1\ \sigma_2 := \sigma$ such that running σ_1 in $\mathcal{T}$ ends halfway between the

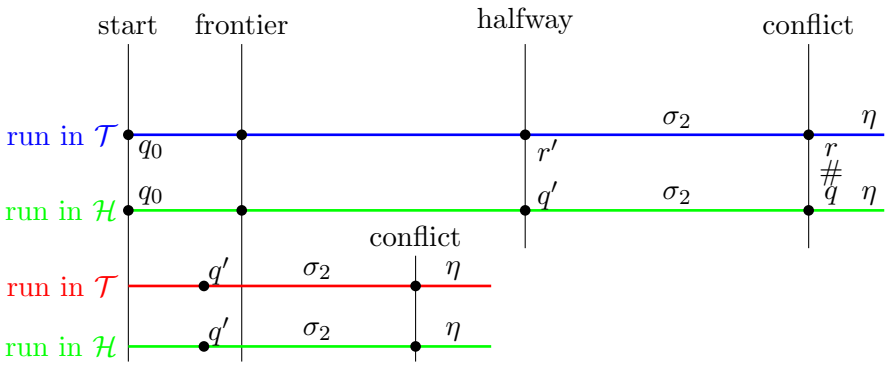

Figure 5: *Counterexample analysis*

frontier and r. Let q' be the state reached by σ_1 in $\mathcal{H}$, and r' the state reached by σ_1 in $\mathcal{T}$. We pose an extra output query $\mathsf{access}(q')\ \sigma_2\ \eta$ and add the result to $\mathcal{T}$. Here $\mathsf{access}(q')$ is the unique access sequence of state q' in $\mathcal{T}$. Now there are two cases. Either $q' \# r'$, which means that σ_1 already leads to conflict. In this case we have reduced the distance from the frontier to a conflict by half, and we recurse. Otherwise, $\mathsf{access}(q')\ \sigma_2$ leads to a conflict, we have again reduced the distance from the frontier to a conflict by half, and we recurse.

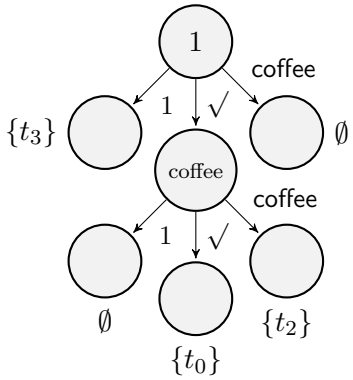

Figure 6: *An adaptive output query*

The implementation of $L^\#$ uses adaptive output queries, in which the choice of an input symbol may depend on the outputs received in response to previous inputs. The main reason why $L^\#$ outperforms other learning algorithms is that it uses dynamic programming to compute an adaptive output query that maximizes the expected number of new apartness pairs. Figure 6 shows the adaptive query that $L^\#$ would use to identify a frontier state fs in our coffee machine example. Essentially, an adaptive output query is a decision tree where the internal nodes are labeled with an input and have outgoing transitions for each possible output. If the first input 1 triggers output 1 we stop the experiment (the candidate set $\mathsf{C}(\mathsf{fs})$ of fs will be $\{t_3\}$), if it triggers output $\mathsf{cofffee}$ we also stop the experiment (the candidate set will be $\emptyset$), if it triggers output $\surd$ we continue the experiment with an input coffee, etc.

Adaptivity only helps a bit in the case of our coffee machine example (the transition from t_{13} to t_{14} is no longer needed), but for larger FSMs the savings can be significant.

3. Lower Bound Results

Balcázar et al. (1997) present a lower bound of $\frac{1}{16} \cdot (k-1) \cdot n^2$ for a weighted sum of the numbers of membership and equivalence queries required for learning a DFA. Below we present a strengthening of this bound, phrased in terms of FSMs; it is straightforward to adapt this result to DFAs. For learning algorithm L, $\sharp outp(L, n, k, m)$ denotes the number of output queries made by L in the worst case over any teacher and any FSM with n states, k inputs and length of the longest counterexample m. The definition of $\sharp equiv(L, n, k, m)$ is analogous; we only count the nonfinal equivalence queries for which the reply is NO. For clarity, we sometimes omit parameters in $\sharp outp(L, n, k, m)$ or $\sharp equiv(L, n, k, m)$ that are irrelevant for the discussion or obvious from the context.

In order to prove the inequality of Theorem 1, we define a collection of acyclic FSMs with n states and k inputs. Assume the teacher chooses one specific FSM from this collection. In order to identify the FSM selected by the teacher, the learner has to determine the target state of $(k-1)\lfloor \frac{n}{2} \rfloor$ transitions. Each of these transition has $\lceil \frac{n}{2} \rceil$ potential target states. In the worst case, for any given transition, the learner either needs $\lceil \frac{n}{2} \rceil - 1$ output queries to determine the target state, or a single equivalence query. For the details of the proofs in this section we refer to Appendix A.

Theorem 1 *For every learning algorithm L, every $n \geq 3$, and every $k \geq 2$,*

$$\sharp outp(L, n, k) + (\lceil \frac{n}{2} \rceil - 1) \cdot \sharp equiv(L, n, k) \quad \geq \quad (k-1) \cdot \lfloor \frac{n}{2} \rfloor \cdot (\lceil \frac{n}{2} \rceil - 1)).$$

In Theorem 1, we assume that $n \geq 3$ and $k \geq 2$. The next theorem presents lower and upper bounds for the basis cases where $n \leq 2$ or $k = 1$.[1]

Theorem 2

1. *For every learning algorithm L, every $n \geq 1$ and every $k \geq 1$, $\sharp outp(L, n, k) + \sharp equiv(L, n, k) > 0$.*
2. *There exists an algorithm L_0 with $\sharp outp(L_0, 1, k) = 1$ and $\sharp equiv(L_0, 1, k) = 0$.*
3. *Assume $k = 1$ and assume the learner knows an upper bound on n. Then there exists an algorithm L_1 that can learn an FSM with a single output query.*
4. *For every learning algorithm L and every $k \geq 2$, $\sharp outp(L, 2, k) + \sharp equiv(L, 2, k) \geq k$.*
5. *Assume $k \geq 2$. There exists an algorithm L_2 with $\sharp outp(L_2, 2, k) = k + 1$ and $\sharp equiv(L_2, 2, k) = 1$.*

1. Balcázar et al. (1997) present no constraints on n in their lower bound result. Since a 1-state DFA can be learned with a single query (e.g. the membership query ϵ), their lower bound is incorrect for $n = 1$.

For completeness, Theorem 2 covers trivial values of n and k as well. For the case $n = 1$, the proofs that the upper and lower bound on the number of queries coincide are trivial. In the case of $k = 1$ lower bound and upper bound also coincide if we assume the learner knows a bound on n. Because a separating sequence has a length of at most $n - 1$, we can perform one long output query of length $2N$, with N the given upper bound, to prove the case of $k = 1$. In case the learner does not know such a bound it needs to perform an equivalence query to check whether its current estimate of the bound on n is correct, and e.g. repeatedly double the estimate in case the case of a NO answer. If $n = 2$ we leave a gap of 2 between the upper and lower bound. For the lower bound, we assume that the initial state has no self-loops and the second state is a sink state, this forces the learner to ask one output query for every input symbol to learn the transition from the initial state to the second state. For the upper bound, we find a separating sequence with the equivalence query and use this separating sequence in an adaptive output query. If the second state is a sink state we need at most $k - 1$ additional preset output queries to learn the FSM. We do not know if the same bounds can be shown with preset output queries only.

It is well-known that for learning algorithms such as L^* and $L^\#$ the maximal number of equivalence queries is at most $n - 1$. Below we show that in the worst case $L^\#$ requires exactly $n - 1$ equivalence queries. In order to prove this result, we define a collection of FSMs with n states and k inputs with $k \geq n$. Assume the teacher chooses one specific FSM from this collection and always returns the minimal counterexample. $L^\#$ identifies frontier states with inputs that are able to separate the previously found states. We exploit this characteristic of $L^\#$ by giving each state a unique input symbol to identify the state. Therefore, each state except the initial state requires an equivalence query because the unique input symbol to identify the state has not been used to separate previously found states.

Theorem 3 *For every $n \geq 1$ and every $k \geq n$, $\sharp equiv(L^\#, n, k) \geq n - 1$.*

The construction in the proof of Theorem 1 involves FSMs that are acyclic except for the sink state. In these FSMs, an adversarial teacher can only return counterexamples of which the length of the relevant part is bounded by the number of states in the FSM. In practice, FSMs often contain loops that allow for counterexamples of unbounded length m. As a result, state-of-the-art active learning algorithms have an asymptotic complexity $\mathcal{O}(kn^2 + n \log m)$, where subterm $n\ log\ m$ corresponds to the output queries needed to process counterexamples. The problem whether or not this subterm is necessary has been open for a long time (it was already mentioned by Balcázar et al. (1997)). As a first step towards solving this open problem, we present a lower bound proof that shows that at least in the asymptotic query complexity of $L^\#$ the subterm $n\ log\ m$ is required. In order to prove the inequality of Theorem 4, we define a collection of acyclic FSMs with n states and k inputs with $k \geq n+1$. Assume the teacher chooses one specific FSM from this collection. Similar to Theorem 3, each state except the initial state has a uniquely identifying input symbol. Contrary to Theorem 3, the teacher does not return the minimal counterexample but a counterexample with many redundant input symbols. The redundant input symbols leads to at least $log(\frac{m}{n} - 1)$ output queries during counterexample processing. Moreover, we ensure that at least $k - 3$ transitions per state lead to the initial state which can only be identified by ruling out all other states, leading to $(k - 3) \cdot n \cdot (n - 1)$ output queries.

Theorem 4 *For every $n \geq 2$ and every $k \geq n+1$,*

$$\sharp outp(L^{\#}, n, k, m) \geq (k-3) \cdot n \cdot (n-1) + log\left(\frac{m}{n} - 1\right) \cdot (n-1)$$

4. Adding Lookahead

The asymptotic symbol complexity of the fastest active learning algorithms is $\mathcal{O}(kmn^2 + mn \log m)$, see Isberner (2015); Frohme (2019); Vaandrager et al. (2022). During our efforts to come up with lower bounds for the symbol complexity of these algorithms, we experimented with the benchmarks from `https://automata.cs.ru.nl` (see also Neider et al. (2018)) that have been used for evaluation of the $L^{\#}$ algorithm Vaandrager et al. (2022). This collection of 46 models has been obtained through case studies involving real implementations of the SSH, TCP, and TLS protocols, alongside bank card applications. The largest model in the collection has 66 states and 13 input symbols. We observed that, for 23 of these benchmarks, even when we generated really long counterexamples, there were never applications of the identification rule with long witnesses. As it turned out, a simple explanation for this observed behavior is that for all these benchmarks the *Maximum length of the Shortest Separating Sequences for all pairs of inequivalent states (MS3)* equals 1. Consider an FSM with a set of states Q over a set of inputs I. A *separating sequence* for two inequivalent states $q, q' \in Q$ is an input sequence $\sigma \in I^*$ s.t. both states have different output responses to the input sequence, and *MS3* is the length of the longest sequence in the set of *all* shortest separating sequences for each pair of states:

$$\max_{(q,q') \in Q \times Q \text{ with } q,q' \text{ inequivalent}} \quad \min_{\sigma \in I^* \text{ a separating sequence for } q,q'} |\sigma|.$$

Via the extension rule, $L^{\#}$ explores all outgoing transitions of each basis state, and this allows the implementation to always find a witness of length 1. Figure 7 shows a histogram of the MS3 values for all 46 benchmarks. As one can see more than 2/3 of the benchmarks have an MS3 of at most 2.

Inspired by Figure 7, we consider, for each natural number h, the class h-FSM of finite state machines with an MS3 value of at most h. Thus, for example, the FSM of Figure 2 (right) belongs to the class 2-FSM, and the FSM of Figure 11 belongs to the class 1-FSM. Assuming that Figure 7 is indicative for a general pattern, it makes sense to look for dedicated learning algorithms that are good at learning models in h-FSM, for small h. In this section, we present $L^{\#}_h$, a variation of $L^{\#}$ that can learn any model in h-FSM. We first discuss $L^{\#}_h$, and then present some experimental results. $L^{\#}_h$ is designed to learn h-FSMs without posing any equivalence queries. It does so by exhaustively exploring all input sequences of length $h+1$ from each state in the basis (or equivalently, all input sequences of length h from each state on the frontier). The idea to explore all words of length $h+1$ from each discovered state is certainly not new and, for instance, also shows up in the h-closed observation packs of Balcázar et al. (1997) and in the approach of Soucha and Bogdanov (2020), which integrates learning and testing. Thus, besides observing the practical relevance of h-FSM, for small h, our contribution is that we provide a correctness proof, a lower bound, and an efficient implementation building on the $L^{\#}$ algorithm.

The $L^{\#}_h$ algorithm starts exactly as $L^{\#}$: from a trivial observation tree with just a single root node in the basis. $L^{\#}_h$ has the following rules:

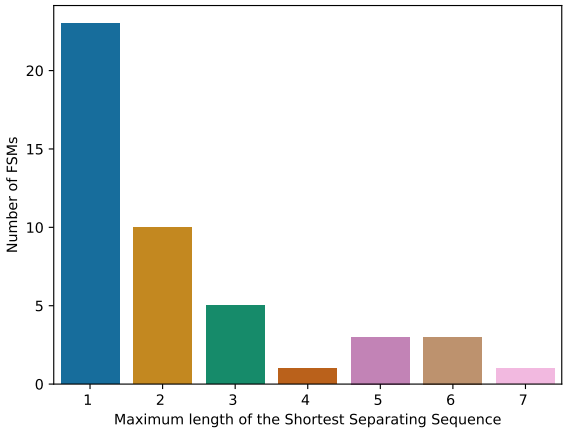

Figure 7: *Histogram of MS3 values for a collection of 46 benchmarks.*

(R1) **Promotion:** This rule is the same as the promotion rule in $L^{\#}$. If frontier F contains a state r with $C(r) = \emptyset$, then we move r from F to S.

(R2') **Extension:** If a state $q \in S$ does not have an outgoing word $\sigma \in I^{h+1}$, then we make an output query $\mathsf{access}(q)\ \sigma$. Here, I^k is the set of all words in I of length k.

(R4') **Equivalence:** If neither rule (R1) nor rule (R2') can be applied, then (see the proof of Theorem 5), the candidate set of all the frontier states is a singleton. Therefore, we may construct a hypothesis $\mathcal{H}$ from $\mathcal{T}$. We return $\mathcal{H}$ as the learned model.

We show how $L_h^{\#}$ learns the coffee machine FSM of Figure 1. Note that the set of input sequences $\{1\ \mathsf{coffee}, 1, \mathsf{coffee}\}$ contains a separating sequence for each (distinct) pair of states. Thus, the model is in 2-FSM and can be learned using $L_2^{\#}$ as follows:

1. Initially, the observation tree has a single node t_0 and basis $\{t_0\}$. The only rule that can be applied (repeatedly) is extension rule (R2'). States t_1 and t_2 now become frontier states.

2. We observe that $1\ \mathsf{coffee}$ is a witness for $t_1 \ \# \ t_0$. This means we may apply promotion rule (R1) and add state t_1 to the basis. States t_3 and t_4 now become frontier states.

3. We observe that coffee is a witness for $t_3 \ \# \ t_0$ and for $t_3 \ \# \ t_1$. This means we may apply promotion rule (R1) again and add state t_3 to the basis. States t_7 and t_8 now become frontier states.

4. We repeatedly apply extension rule (R2') for states t_3 and t_2 (not visualized here). This does not add any new states to the basis or frontier, but it allows us to identify all frontier states (reduce their candidate set to a singleton).

5. We apply equivalence rule (R4') and construct an hypothesis $\mathcal{H}_2$. The algorithm terminates.

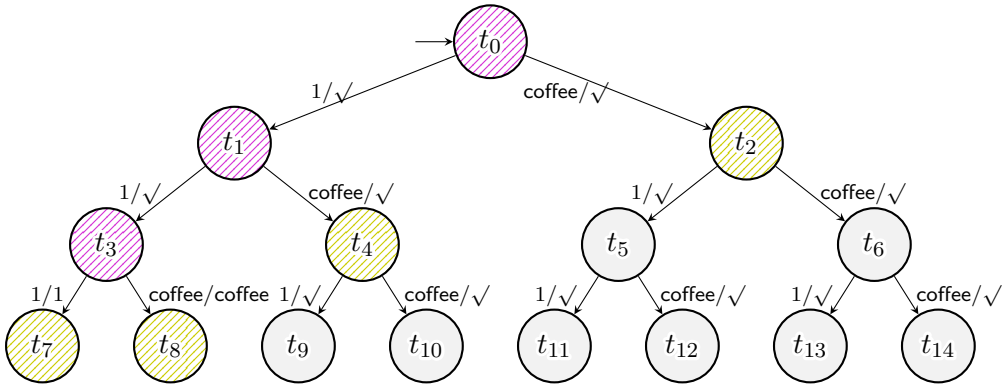

Figure 8: *Observation tree after steps 1-3.*

The theorems below state the correctness of $L_h^\#$ and provide lower and upper bounds on the number of output queries posed by the algorithm. Theorem 5 relies on the fact that rules (R1) and (R2') can only be applied a finite number of times. If rules (R1) and (R2') cannot be applied anymore, the algorithm terminates. Upon termination, each state has outgoing paths for each input sequence of length at most h. Because we assume that all states are distinguishable with a sequence of length at most h, all states must have been found because otherwise, rule (R1) can be applied. In the worst-case scenario of Theorem 6, rule (R2') has to be applied for each basis state and each $h+1$ sequence. In the best-case scenario, some basis states have other basis states as successors, allowing for the reuse of previously asked output queries.

Theorem 5 *For any h, $L_h^\#$ correctly learns any model in h-FSM.*

Theorem 6 *For any h, $L_h^\#$ requires at least $nk^{h+1} - (n-1)k^h$ and at most nk^{h+1} output queries to learn a model in h-FSM.*

We present some experimental results using $L_2^\#$ to learn 1-FSMs and 2-FSMs from the FSMs available at `https://automata.cs.ru.nl`. We have selected 33 FSMs to learn. For $L^\#$, we use our implementation of Hybrid-ADS (Smeenk et al., 2015) to answer equivalence queries, constructing a complete test suite for up to 2 extra states. Additionally, while we typically skip the final equivalence query, where we know the hypothesis generated is correct (i.e., where the teacher answers YES), in this case, we do not. $L_2^\#$ is a learner which essentially combines the learning and testing phase of active automata learning. Given that, it would be unfair to skip the final equivalence query for Hybrid-ADS and not for $L_h^\#$.

Figure 9 shows the results of the number of queries made by $L_2^\#$ and $L^\#$+ Hybrid-ADS while learning our FSMs.[2] We also include the upper and lower bounds for each model. As expected, $L_2^\#$ generally asks fewer queries than standard $L^\#$ for learning FSMs belonging to the classes 1-FSM and 2-FSM. While reaching the exact lower bound in practice would be very difficult, we get quite close to it using some heuristics adopted from the $L^\#$ implementation. The $L^\#$ + Hybrid-ADS version also performs well (with the exception

2. A full table with all the data can be found in Appendix B.

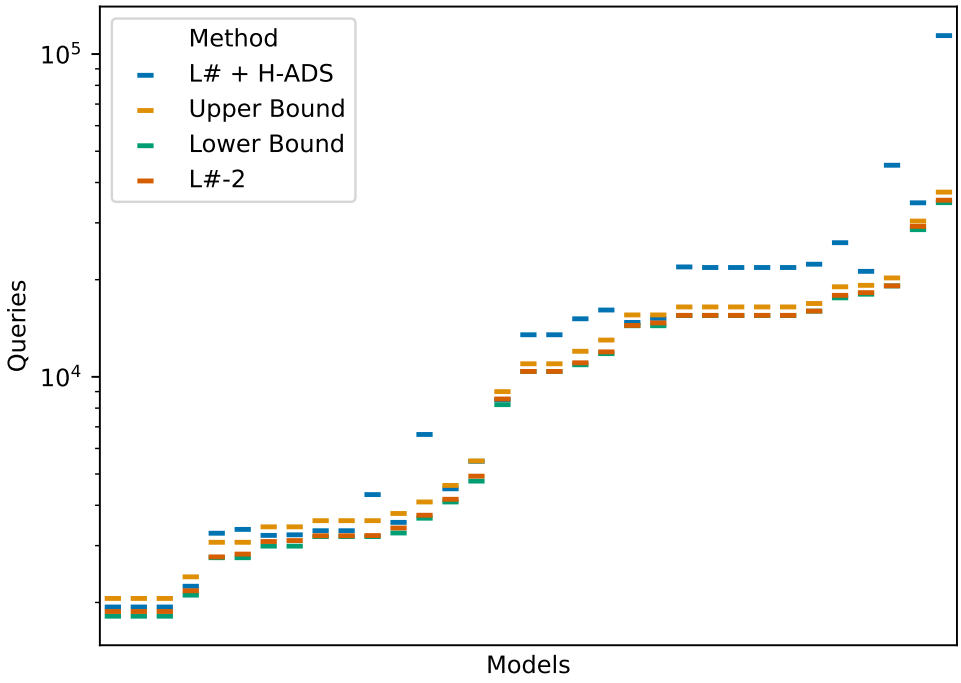

Figure 9: *Comparison of $L_2^\#$ and $L^\#$ + Hybrid-ADS. Models are sorted in ascending order of (n, k). In addition to the experimental results, we also plot the upper and lower bounds derived from Theorem 6.*

of a couple of outliers); however, for most cases it performs slightly worse than $L_h^\#$. The source code for our tool can be found online.[3] We have also provided a Dockerfile for ease of replication.

Discussion Our results by no means imply that $L_2^\#$ is 'better' than $L^\#$ combined with a conformance testing algorithm: both approaches offer different guarantees and advantages.

$L_h^\#$ guarantees that the learned model is correct if we assume that the system under learning is in the class h-FSM. However, in practice this is typically not a realistic assumption. Similarly, a learner who uses a learning algorithm such as $L^\#$ in combination with a conformance testing algorithm to approximate equivalence queries, typically needs to make an unrealistic assumption in order to claim correctness of a model. For instance, standard conformance testing algorithms (such as Hybrid-ADS) construct test suites that are *t-complete*, for some number t. A test suite T is t-complete for an hypothesis $\mathcal{H}$ if, for any FSM $\mathcal{M}$ with at most t extra states with respect to $\mathcal{H}$, $\mathcal{M}$ passes T iff $\mathcal{M}$ is equivalent to $\mathcal{H}$. In practice it is usually not realistic to bound on the number of states of a system under learning with a small t. However, since the size of test suite T grows exponentially in t, running test suits for values $t > 3$ is typically not feasible. Similarly, since by Theorem 6 the number of output queries of $L_h^\#$ grows exponentially in h, running $L_h^\#$ for values $h > 3$ is typically not feasible.

3. Available at https://gitlab.science.ru.nl/sws/lsharp.git under the branch "icgi."

If we approximate an equivalence oracle by running a h-complete test suite for the hypothesis, then a learning algorithm may succeed to learn models from k-FSM, where $k > h$, while $L_h^\#$ will not be able to do so. However, if we somehow know to which class h-FSM some system belongs, then we can learn the correct model using $L_h^\#$ without resorting to expensive equivalence queries, no matter the number of states of this system.

One may view the number of states and the MS3 value of the minimal FSM as complexity measures of systems, similar to other complexity measures that have been considered in the literature. Fisman (2018), for instance, suggests that the column-index can be viewed as a complexity measure for a regular language. At a more speculative level, one may view a low MS3 value as a guideline for design. Similar to the way in which *Design for Testability* (Williams and Parker, 1983) is a well-known approach that aims to make digital circuits easier to test during the manufacturing and debugging process, one could propose *Design for Learnability* as an approach that aims to make it easier to learn models of reactive systems. FSMs with an MS3 value of 1, for instance, can be constructed by introducing a special input *state* that triggers a self-loop transition with the name of the current state as output.

5. Conclusions and Future Work

We improved the lower bound of Balcázar et al. (1997) on the combined number of output and equivalence queries required by any learning algorithm, and provided lower bounds for the query complexity of $L^\#$ that are rather close to the known upper bounds. As argued in the introduction, in practical learning scenarios we are often not so much interested in minimizing the number of queries required for learning, but rather in minimizing the number of symbols required for learning plus testing. In this context, it may be interesting to explore clever combinations of $L^\#$ and algorithms that use exhaustive search like $L_h^\#$. Many open problems remain, for instance defining efficient learning algorithms whose query complexity does not depend on the length of counterexamples, proving nontrivial lower bounds on the symbol complexity of learning algorithms, and proving that adaptive sequences do (or do not) help to improve the asymptotic complexity of learning algorithms.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback.

References

D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2): 87–106, 1987a.

D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1987b. URL https://doi.org/10.1007/BF00116828.

D. Angluin. Negative results for equivalence queries. *Mach. Learn.*, 5:121–150, 1990. URL https://doi.org/10.1007/BF00116034.

Kousar Aslam, Loek Cleophas, Ramon R. H. Schiffelers, and Mark van den Brand. Interface protocol inference to aid understanding legacy software components. *Softw. Syst. Model.*, 19(6):1519–1540, 2020. doi: 10.1007/s10270-020-00809-2.

J.L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe. The query complexity of learning DFA. *New Gener. Comput.*, 12(4):337–358, 1994. URL https://doi.org/10.1007/BF03037351.

J.L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe. Algorithms for learning finite automata from queries: A unified view. In Ding-Zhu Du and Ker-I Ko, editors, *Advances in Algorithms, Languages, and Complexity - In Honor of Ronald V. Book*, pages 53–72. Kluwer, 1997.

Dana Fisman. Inferring regular languages and ω-languages. *J. Log. Algebraic Methods Program.*, 98:27–49, 2018. URL https://doi.org/10.1016/j.jlamp.2018.03.002.

Markus Theo Frohme. Active automata learning with adaptive distinguishing sequences. *CoRR*, abs/1902.01139, 2019. URL http://arxiv.org/abs/1902.01139.

Roland Groz, Nicolas Brémond, Adenilso da Silva Simão, and Catherine Oriat. *hW*-inference: A heuristic approach to retrieve models through black box testing. *J. Syst. Softw.*, 159, 2020. doi: 10.1016/j.jss.2019.110426.

F. Howar. *Active learning of interface programs*. PhD thesis, University of Dortmund, June 2012.

Falk Howar and Bernhard Steffen. Active automata learning in practice. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, pages 123–148. Springer International Publishing, 2018. ISBN 978-3-319-96562-8.

Malte Isberner. *Foundations of active automata learning: an algorithmic perspective*. PhD thesis, Technical University Dortmund, Germany, 2015. URL http://hdl.handle.net/2003/34282.

Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON,*

Canada, September 22-25, 2014. Proceedings, pages 307–322, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11164-3.

L. Kruger. *How to Handle Long Counterexamples: Heuristics, Optimizations and Asymptotic Lower Bounds for $L^{\#}$*. Master thesis, Radboud University Nijmegen, February 2023.

Daniel Neider, Rick Smetsers, Frits W. Vaandrager, and Harco Kuppens. Benchmarks for automata learning and conformance testing. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 390–416. Springer, 2018.

R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences (extended abstract). In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, 15-17 May 1989, Seattle, Washington, USA*, pages 411–420. ACM, 1989.

R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993. doi: 10.1006/inco.1993.1021.

Wouter Smeenk, Joshua Moerman, Frits W. Vaandrager, and David N. Jansen. Applying automata learning to embedded control software. In Michael J. Butler, Sylvain Conchon, and Fatiha Zaïdi, editors, *Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings*, volume 9407 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2015. URL https://doi.org/10.1007/978-3-319-25423-4_5.

Michal Soucha and Kirill Bogdanov. Observation tree approach: Active learning relying on testing. *Comput. J.*, 63(9):1298–1310, 2020. doi: 10.1093/comjnl/bxz056.

Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In Marco Bernardo and Valérie Issarny, editors, *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 2011. URL https://doi.org/10.1007/978-3-642-21455-4_8.

Frits Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, February 2017. ISSN 0001-0782. doi: 10.1145/2967606.

Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022. URL https://doi.org/10.1007/978-3-030-99524-9_12.

T.W. Williams and K.P. Parker. Design for testability—a survey. *Proceedings of the IEEE*, 71(1):98–112, 1983. doi: 10.1109/PROC.1983.12531.

Appendix A. Proofs

Proof of Theorem 1

We construct a family of FSMs $\mathcal{M}_{n,k,f}$, for $n \geq 3$, $k \geq 2$, and any function $f : [0, l-1] \times [1, k-1] \to [l, n-1]$, where $l = \lfloor \frac{n}{2} \rfloor$. Figure 10 shows one element from this family. Machine $\mathcal{M}_{n,k,f}$ is defined as follows:

- The set of states is $\{q_0, q_1, \ldots, q_{n-1}\}$ and the initial state is q_0.
- The set of inputs is $I = \{a_0, a_1, \ldots, a_{k-1}\}$. We write $a = a_0$ and $b = a_1$.
- The set of outputs is $\{0, 1\}$.
- There is a spine of a-transitions; for all $0 \leq j < n-1$:

$$q_j \xrightarrow{a/0} q_{j+1} \tag{1}$$

  All transitions carry output 0, except for a single transition with output 1:

$$q_{n-2} \xrightarrow{b/1} q_{n-1} \tag{2}$$

  Function f specifies transitions, for non-a inputs, from the first half of the spine to the second half of the spine. For $0 \leq j < l$ and $1 \leq p < k$,

$$q_j \xrightarrow{a_p/0} q_{f(j,p)} \tag{3}$$

  State q_{n-1} acts as a sink state, that is, if for some pair q_j and a_p, no outgoing transition is specified in any of the above three rules, this means there is a transition to the sink state with output 0:

$$q_j \xrightarrow{a_p/0} q_{n-1} \tag{4}$$

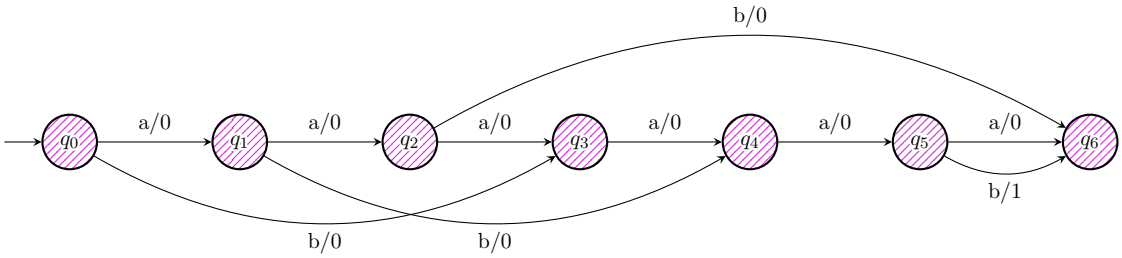

Figure 10: *Example of construction for Theorem 1 with $n = 7$, $k = 2$, $f(0,1) = 3$, $f(1,1) = 4$, and $f(2,1) = 6$. For readability, some edges to sink state q_6 are not drawn.*

Note that all states of FSM $\mathcal{M}_{n,k,f}$ are reachable and that the machine is minimal. The set $C = \{a^j\ b \mid 0 \leq j \leq n-2\}$ constitutes a characterization set with a separating sequence for each pair of states. Also note that amongst the outputs in response to any input sequence there is at most one 1, all other outputs are 0. There are at most $l \cdot (k-1)+1$ input sequences

such that the output in response to the last input equals 1. We call these input sequences *important*, and use S to denote the set of important input sequences. Note that set S uniquely determines function f. This means that if f and g are different functions, their sets of important sequences are different, and hence $\mathcal{M}_{n,k,f}$ and $\mathcal{M}_{n,k,g}$ are inequivalent.

We confront a given learning algorithm L against the family of target FSMs $\mathcal{M}_{n,k,f}$. This means that initially the learner knows n, I and important sequence $a^{n-2}\ b$. The task of the learner is to learn f, i.e., to determine the full set S of important sequences.

By posing an equivalence query, the learner may sometimes force the teacher to reveal an important sequence. For instance, if the learner offers a hypothesis in which $a^{n-2}\ b$ is the only sequence that triggers an output 1, then the teacher will be forced to reveal a new important sequence as a counterexample. However, since any path in any automaton from our class contains at most one transition of type (3), any counterexample will provide the learner with information about at most one function value $f(j,p)$.

The learner may also use output queries to determine a function value $f(j,p)$. For instance, in the case of $n = 7$ and $k = 2$ of Figure 10, the learner may perform output queries $b\ a\ a\ b$, $b\ a\ b$ and $b\ b$ to determine the value of $f(0,1)$: if one of the sequences triggers an output 1 then the learner has discovered an important sequence (revealing the value of $f(0,1)$), and if none of the sequences triggers a 1 then the learner concludes that $f(0,1) = 6$. In general, the learner will in the worst case need $n-l-1$ queries to determine the value of $f(j,p)$.

In the worst case, learner L will need to perform $n-l-1$ output queries or 1 equivalence query in order to discover the value of $f(j,p)$. Therefore, since the domain of f contains $(k-1)\cdot l$ elements, we have

$$\sharp outp(L,n,k) + (n-l-1)\cdot \sharp equiv(L,n,k) \quad \geq \quad (k-1)\cdot l\cdot (n-l-1)$$

or equivalently

$$\sharp outp(L,n,k) + (\lceil \frac{n}{2} \rceil - 1)\cdot \sharp equiv(L,n,k) \quad \geq \quad (k-1)\cdot \lfloor \frac{n}{2} \rfloor \cdot (\lceil \frac{n}{2} \rceil - 1)).$$

Proof of Theorem 2

1. Since $k \geq 1$ the initial state has at least one outgoing transition. In order to determine the output of that transition we need at least one query.

2. Assume that the set of inputs, known to the learner, is $I = \{a_0, \ldots, a_{k-1}\}$. Then L_0 performs a single output query $a_0\ a_1 \cdots a_{k-1}$. The reply contains a unique output for each input, allowing L_0 to infer all outgoing transitions of the unique state.[4]

3. Let the input symbol be a and assume the learner knows a bound N on n. Algorithm L_1 performs a single output query a^{2N} and then behaves as $L^\#$ continuing adding new states to the basis with promotion rule (R1). In $L^\#$ the basis contains at most n states. But since the length of separating sequence in an FSM is at most $n-1$, and the length $2N$ of the single trace in the observation tree is at least $2n$, $L^\#$ will construct

4. In some papers on active automata learning, e.g. Steffen et al. (2011), the answer to an output query only contains the output for the last input symbol. In such a setting, at least k output queries will be required to learn a 1-state model.

a basis with n states, and determine a singleton candidate set for the frontier state after the n-th basis state. Thus, $L^{\#}$ will construct the correct hypothesis without the need to perform additional output queries.

4. This is a variant of the construction in the proof of Theorem 1. We consider a family of FSMs $\mathcal{M}_{k,f}$, for $k \geq 2$, and any function $f : [0, k-1] \to \{0, 1\}$. Machine $\mathcal{M}_{k,f}$ has states q_0 (the initial state) and q_1, inputs $a_0, \ldots a_{k-1}$, and outputs 0 and 1. For each $j \in [0, k-1]$ there are transitions $q_0 \xrightarrow{a_j/f(j)} q_1 \xrightarrow{a_j/0} q_1$. We confront the learning algorithm L against this family of target FSMs. Since each output or equivalence query only provides information about at most one outgoing transition of q_0, the learner needs at least k queries in the worst case to determine the outputs on the outgoing transitions of q_0.

5. We present a learning algorithm L_2 that only needs $k+1$ output queries and a single equivalence query to learn an FSM $\mathcal{M}$ with 2 states:

   (a) First L_2 performs an output query $a_0\, a_1 \cdots a_{k-1}$ containing all the inputs.

   (b) Next, L_2 performs an equivalence query for a hypothesis $\mathcal{H}$ with a single state and outputs as observed in the output query. If $\mathcal{H}$ is incorrect, then the provided counterexample must contain at least one input a with an output that is different from the output triggered by a in the output query. This a is a separating sequence for states q_0 and q_1.

   (c) Now L_2 determines the output and target states of all transitions using one adaptive output query and at most $k-1$ preset output queries. The adaptive query begins with 2 a's. After the first a we know the output of a in q_0 and after the second a we know the target state after input a in q_0. As long as the target state of transitions is q_0 we explore the output and target for other inputs d in q_0, simply by doing d followed by a. But since state q_1 is reachable, at some point there will be an input b that brings us to state q_1 (it may be that $b = a$). Now we first determine the target of input a in state q_1 by doing another a. As long as the target state of transitions is q_1 we explore the output and target for other inputs d in q_1, simply by doing d followed by a. Some input c may bring us back again from q_1 to q_0 (possibly $c = a$). If we discover such a c then we can go wherever we want since b brings us from q_0 to q_1, and c brings us from q_1 back to q_0. In this case we may simply continue our adaptive query and determine the output and target state for each input d in each of the states by first going to that state (using either b or c) if necessary, then doing d (observing the output), followed by a (to observe the target state). If there is no c that brings us from q_1 back to q_0 then, after having determined the outputs for all outgoing transitions in q_1, we need to perform at most $k-1$ additional (preset) output queries da for all the inputs for which we do not know yet the output and target state from q_0.

Proof of Theorem 3

We construct a family of FSMs $\mathcal{M}_{n,k}$, for $n \geq 1$ and $k \geq n$. Figure 11 shows one element from this family. Machine $\mathcal{M}_{n,k}$ has states $\{q_0, \ldots, q_{n-1}\}$, initial state q_0, inputs

$\{a_0, \ldots, a_{k-1}\}$ and outputs is $\{0, 1\}$. There is a spine of a_0-transitions; for all $0 \leq j < n-1$:

$$q_j \xrightarrow{a_0/0} q_{j+1}$$

In state q_j there is a self-loop for input a_j with output 1, for $0 < j < n$; these are the only transitions with output 1:

$$q_j \xrightarrow{a_j/1} q_j$$

All other inputs trigger self-loops with output 0; for all $0 \leq j < n$ and $0 \leq p < n$ such that either $p = 0 \wedge j = n-1$ or $p \neq 0 \wedge p \neq j$:

$$q_j \xrightarrow{a_p/0} q_j$$

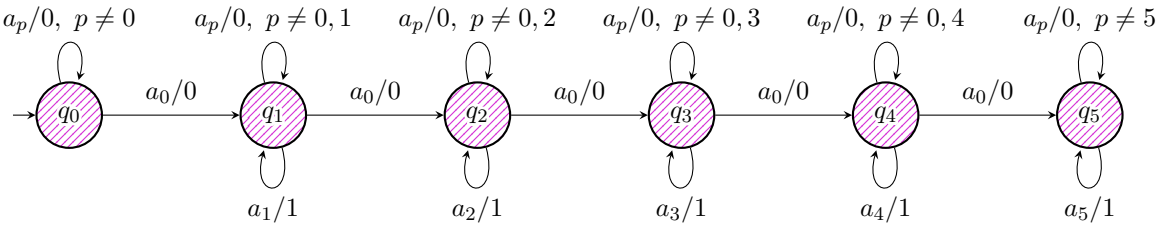

Figure 11: *Example of construction for Theorem 3 with $n = 6$.*

After exploring all outgoing input transitions from the initial state, $L^\#$ constructs a first hypothesis $\mathcal{H}_1$ with a single state and self-loops with output 0 for all k inputs. We assume that the teacher always provides a counterexample of minimal length, in this case $a_0\ a_1$. Since this leads to a conflict at the frontier, counterexample processing finishes immediately, and a state in which input a_1 triggers output 1 is added to the basis. Input a_1 witnesses the apartness of the two basis states, so after completing the frontier witness a_1 is used to identify the frontier states. A second hypothesis is $\mathcal{H}_2$ is constructed, for which the teacher generates counterexample $a_0\ a_0\ a_2$, a state in which input a_2 triggers output 1 is added to the basis, etc. Whenever $L^\#$ identifies frontier states it only tries inputs that played a role in separating the basis states that it has found thus far, not any other input that would reveal the presence of a new state. Each time, $L^\#$ needs the help of the teacher to discover a next state, and thus $n-1$ (non-terminating) equivalence queries will be made.

Proof of Theorem 4

We construct a family of FSMs $\mathcal{M}_{n,k}$, for $n \geq 2$ and $k \geq n+1$. Figure 12 shows one element from this family. Machine $\mathcal{M}_{n,k}$ has states $\{q_0, \ldots, q_{n-1}\}$, initial state q_0, inputs $I = \{a_0, \ldots, a_{k-1}\}$ and outputs is $\{0, 1\}$. There is a spine of a_0-transitions; for all $0 \leq j < n-1$:

$$q_j \xrightarrow{a_0/0} q_{j+1}$$

In state q_j there is a self-loop for input a_j with output 1, for $0 < j < n$; these are the only transitions with output 1:

$$q_j \xrightarrow{a_j/1} q_j$$

Additionally, each state q_j has a self-loop with input a_{k-1} with output 0, for $0 \leq j < n$

$$q_j \xrightarrow{a_{k-1}/0} q_j$$

All other inputs trigger transition to the initial state with output 0; for all $0 \leq j < n$ and $0 \leq p < k$ such that either $p = 0 \wedge j = n-1$ or $p \neq 0 \wedge p \neq j \wedge p \neq k-1$:

$$q_j \xrightarrow{a_p/0} q_0$$

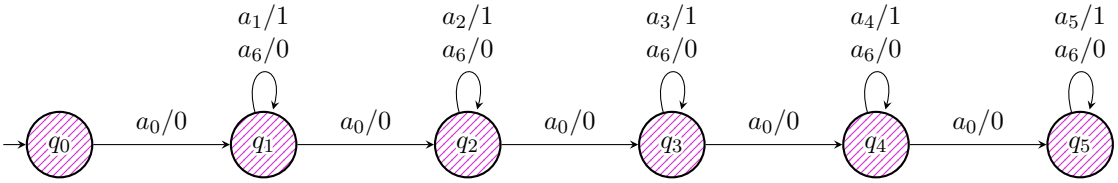

Figure 12: *Example of construction for Theorem 4 with $n = 6$. For readability, the transitions to state q_0 are not drawn.*

After exploring all outgoing input transitions from the initial state, $L^\#$ constructs a first hypothesis $\mathcal{H}_1$ with a single state and self-loops with output 0 for all k inputs. The teacher always finds a minimal counterexample and then modifies it. A minimal counterexample always has the form $a_0^j\, a_j$ where j is equal to the current number of states in the hypothesis. But instead of returning a minimal counterexample, the teacher returns the result of the conversion function $g_i : I^+ - > I^+$ where i is some natural number that is fixed for the complete learning procedure. We specify function g_i as follows:

$$g_i(a_0^j\, a_j) = a_0^j\, a_{k-1}^{n \cdot 2^i}\, a_j$$

Function g_i blows up the length of the counterexample. Because i can be chosen freely, the length of the counterexample after application of g_i is unbounded. While processing a counterexample generated by g_i, we find that σ_2 only contains input symbols a_{k-1} if $|\sigma_1| \geq n$. Moreover, the access sequence cannot contain j symbols a_0 because the last found state q_j has an access sequence of length $j-1$. Together, this means that the output query performed by counterexample processing never contains enough a_0's to get output 1 if $|\sigma_1| \geq n$. Therefore, the output query gives us no useful information which means we have to use σ_1 again in the recursive call and the witness η gets longer. We know that $|\sigma_1| \geq n$ occurs at least i times during the processing of the counterexample.

$$\frac{j + n \cdot 2^i + 1}{2^i} = \frac{j+1}{2^i} + \frac{n \cdot 2^i}{2^i} = \frac{j+1}{2^i} + n \geq n$$

When counterexample processing finishes, a state where input a_j triggers output 1 is promoted to the basis. Input a_j witnesses the apartness of the new basis state with all other basis states, so after completing the frontier witness a_1 is used to identify the frontier states. Whenever $L^\#$ identifies frontier states, it only tries inputs that played a role in separating

the basis states that it has found thus far, not any other input that would reveal the presence of a new state, and thus $n-1$ (non-terminating) equivalence queries will be made each using i at least output queries. The maximal counterexample generated by g_i has length $m = (n-1) + n \cdot 2^i + 1 = n + n \cdot 2^i$. We can rewrite this statement to find

$$i = log\left(\frac{m-n}{n}\right) = log\left(\frac{m}{n} - 1\right)$$

Therefore, we know that $L^{\#}$ needs at least $log(\frac{m}{n} - 1)$ output queries to process each counterexample.

Additionally, each state has at least $k-3$ transitions leading to the initial state. The initial state can never be identified by a witness found during the learning process because we only use witnesses of the form a_j with $0 < j < n$ for identification and these witnesses always lead to output 0 in state q_0. This means that for each of these $k-3$ transitions, we need to rule out all other $n-1$ states during the learning process. In total, this means that $L^{\#}$ needs at least $(k-3) \cdot n \cdot (n-1)$ output queries to identify frontier states.

By combining the output queries needed to process counterexamples and to identify the frontier states, the theorem holds.

Proof of Theorem 5

Fix a number h and fix an FSM $\mathcal{M}$ from the class h-FSM.

At any point when we run the $L_h^{\#}$ algorithm, there exists a functional simulation f from the states of the observation tree to states of $\mathcal{M}$: a function that preserves initial states and transitions. If q and q' are two different states in the basis then, by definition, there exists a witness σ for $q \# q'$. Since f is a functional simulation, this implies that σ is also a separating sequence for $f(q)$ and $f(q')$. This in turn implies that the number of states in the basis S can be at most equal to the number n of equivalence classes of reachable states of FSM $\mathcal{M}$. Therefore, rules (R1) and (R2') can only be applied finitely many times and the $L_h^{\#}$ algorithm will terminate.

Let $\mathcal{T}$ be the observation tree upon termination of the algorithm. Then each state of the basis S and each state of the frontier F will have outgoing paths for each sequence of inputs of length at most h.

Suppose r and r' are two states of $\mathcal{M}$ that are inequivalent. Then, by definition of an h-FSM, there exists a separating sequence σ of length at most h such that the outputs triggered by σ from r and r' are different. Now suppose that q and q' are two distinct states in the basis. Since $f(q)$ and $f(q')$ are inequivalent and therefore have a separating sequence of length at most h, and because q and q' enable all input sequences of length h, there exists a witness of length at most h such that $\sigma \vdash q \# q'$. It also follows that for each state t in the frontier F the candidate set $C(t)$ is a singleton: $C(t)$ cannot be empty because then we could apply rule (R1), but also does not contain two distinct basis states q and q', as t enables the witness σ of length at most h which demonstrates that $q \# q'$. In the terminology of Vaandrager et al. (2022), all states of the frontier have been *identified*. Since rule (R1) cannot be applied anymore, we know that the number of states in basis S is equal to the number n of equivalence classes of reachable states of $\mathcal{M}$. Let $\mathcal{H}$ be the hypothesis constructed from the observation tree and basis S. We may now apply Theorem 3.7 of Vaandrager et al. (2022) to conclude that $\mathcal{H}$ and $\mathcal{M}$ are equivalent.

Proof of Theorem 6

Clearly, in the worst case $L_h^{\#}$ performs rule (R2') for each basis state and for each input sequence of length $h+1$. This explains the upper bound nk^{h+1}. However, the actual number of queries may be lower. For instance, if in Figure 8 we first apply rule (R2') eight times in state t_3, we only need to apply (R2') four times in state t_1. A lower bound can be derived as follows. Frontier states are successors of basis states that are not basis states themselves. Upon termination of the algorithm there are n basis states. This means that there are at least $nk-(n-1)$ frontier states (we know that the root of the observation tree is not a successor of a basis state). For each frontier state we must explore all possible input sequences of length h. For this we will need at least $k^h(nk-(n-1)) = nk^{h+1}-(n-1)k^h$ output queries.

Appendix B. Benchmarking Results for $L_2^\#$

SUL name	n	k	$L^\#$ + H-ADS	Lower Bound	$L_2^\#$	Upper Bound
4_learnresult_SecureCode_Aut_fix	4	14	13501	10388	10394	10976
ASN_learnresult_SecureCode_Aut_fix	4	14	13501	10388	10394	10976
1_learnresult_MasterCard_fix	5	15	22333	15975	15986	16875
OpenSSL_1.0.1l_client_regular	6	7	1936	1813	1875	2058
OpenSSL_1.0.1j_client_regular	6	7	1936	1813	1875	2058
OpenSSL_1.0.2_client_regular	6	7	1936	1813	1875	2058
RSA_BSAFE_Java_6.1.1_server_regular	6	8	3277	2752	2768	3072
miTLS_0.1.3_server_regular	6	8	3368	2752	2824	3072
4_learnresult_PIN_fix	6	14	21813	15484	15506	16464
Rabo_learnresult_MAESTRO_fix	6	14	21813	15484	15506	16464
4_learnresult_MAESTRO_fix	6	14	21813	15484	15506	16464
ASN_learnresult_MAESTRO_fix	6	14	21813	15484	15506	16464
10_learnresult_MasterCard_fix	6	14	21904	15484	15506	16464
Rabo_learnresult_SecureCode_Aut_fix	6	15	45243	19125	19141	20250
OpenSSL_1.0.2_server_regular	7	7	2245	2107	2174	2401
GnuTLS_3.3.12_server_regular	7	8	3336	3200	3218	3584
GnuTLS_3.3.12_client_regular	7	8	3336	3200	3218	3584
NSS_3.17.4_client_regular	7	8	4317	3200	3223	3584
Volksbank_learnresult_MAESTRO_fix	7	14	21224	18032	18234	19208
NSS_3.17.4_server_regular	8	8	6629	3648	3726	4096
RSA_BSAFE_C_4.0.4_server_regular	9	8	4499	4096	4175	4608
OpenSSL_1.0.2_client_full	9	10	8493	8200	8538	9000
GnuTLS_3.3.12_server_full	9	12	14745	14400	14428	15552
GnuTLS_3.3.12_client_full	9	12	15180	14400	14689	15552
learnresult_fix	9	15	34607	28575	29246	30375
OpenSSL_1.0.1g_client_regular	10	7	3227	2989	3089	3430
OpenSSL_1.0.1l_server_regular	10	7	3240	2989	3110	3430
OpenSSL_1.0.1j_server_regular	11	7	3541	3283	3402	3773
NSS_3.17.4_client_full	11	12	26037	17568	17885	19008
TCP_FreeBSD_Client	12	10	15130	10900	11053	12000
TCP_Windows8_Client	13	10	16107	11800	11949	13000
OpenSSL_1.0.1g_server_regular	16	7	5475	4753	4924	5488
DropBear	17	13	114081	34645	35254	37349