On Learning Incomplete Automata

Raffaella Gentilini[1]

[1]University of Perugia

Abstract

Recently, an active learning algorithm for regular languages yielding a possibly incomplete automaton (reading only the prefixes of the words belonging to the target language) has been introduced in [1]. In particular, the classic L^* algorithm proposed by Angluin [2] has been modified toward the so-called PL^* learner. The latter leverages on a modified teacher that answers to equivalence and *prefix* queries, instead of operating with the two classic oracles for testing language equivalence and (word) membership. We provide an alternative learner for incomplete DFAs that does not employ such a new prefix oracle, but interacts with a standard teacher using only membership and equivalence queries.

Keywords

Regular Languages, Active Learning, Incomplete Transition Function

1. Introduction

Automata learning refers to the process of inferring a finite representation for an unknown target formal language. In this context, the term learner is used to indicate an algorithm that builds (or learns) an automaton accepting the target unknown language.

When the learned automaton models an unknown system, the inference process is also called *model learning* or *black-box learning* [3] and finds applications in model checking, verification and reactive synthesis [4, 3, 5, 6]. In particular, being automata a well established formalism to model e.g. (security or communication) protocols and hardware/software controllers, model learning is naturally applied in all those areas to infer or learn a model from a given (black box) system, just by observing its behaviour or its response to certain queries. The learned model can then be used for rigorous analysis to detect bugs or vulnerabilities and design possible fixes, ensuring compliance with specifications [3, 7].

The problem of automata learning has been studied for decades, being initially explored as a fundamental theoretical problem in automata theory and later developed toward model learning for inferring behavioral models of complex black-box systems such as communication protocols and software controllers [8, 9, 7, 10, 11, 5]. The L^* algorithm, introduced by Dana Angluin in 1987, is the foundational learner for inferring a Deterministic Finite Automaton (DFA) from a black-box system. L^* is often referred to as an *active* learning algorithm, because it is based on a direct interaction of the learner with an oracle (or teacher) that can answer to two types of queries: membership queries (does the word w belong to the target language L?) and equivalence queries (is $L(\mathcal{A}) = L$? for a given hypothesis DFA $\mathcal{A}$). In general, model learning in the active framework relies on actively doing experiments (tests simulating the above membership and equivalence queries) on the black box system under learning [12]. This is in contrast with passive learning, where a fixed set of positive and negative examples is given and no interaction with the system is possible.

The class of regular languages cannot be learned efficiently in polynomial time using either membership queries alone or equivalence queries alone [13, 14] and a polynomial DFA learner necessitates indeed the active learning framework introduced by L^*, often referred to as Minimally Adequate Teacher (MAT) framework. Recently, in [1] L^* has been modified toward the so-called PL^* learner. The latter operates within an active framework involving an oracle for answering *prefix* queries (is the word w prefix of some word in the target language L?) beside classic equivalence ones. In particular, PL^*

ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy

*Corresponding author.

raffaella.gentilini@unipg.it (R. Gentilini)

learns a possibly incomplete DFA accepting the target regular language L and is proven to learn more efficiently than L^* both empirically and theoretically (i.e. with globally less queries to the teacher). In [1], the motivation underlying the development of PL^* relies on applying active automata learning in the context of black box learning of program's input parsers. This is because many input parsers usually indicate not only if an input is valid or not, but also if it is the prefix of some valid input. Beside this motivation, it turns out that PL^* is the first DFA learner in the literature that does not rely on having a complete transition function, rather it leverages on incomplete ones. While regular languages always admit a minimum complete canonical model, there are classes of formal languages whose automata recognizers can not be always embedded into a complete one. This is the case for the class of Wheeler languages and Wheeler automata [15] that play an important role in the field of compressed data structures. PL^* is therefore also relevant for the extension of model learning toward such kind of formal languages. In this paper, we provide a polynomial learner for incomplete DFAs alternative to PL^* that interacts with a standard teacher using only membership and equivalence queries. Such a learner does not rely on a complete transition function for the target DFA, and learns a possibly incomplete automaton that reads only the prefixes of the words belonging to the target language. Moreover, within the proposed framework the alphabet does not need to be known in advance, rather it is also learned. Beside giving new insights on regular languages active learning, this result pushes further the path toward learning (in the classic MAT framework) classes of automata that can not be generally embedded into complete automata, as the recalled class of Wheeler automata.

The paper is organized as follows. We introduce some useful notation in the preliminary section. Section 3 defines the active learner in the MAT framework for incomplete automata and illustrates its execution on a concrete example. Section 4 analizes the correctness and the complexity of the proposed algorithm. Section 5 analizes the number of queries to the teacher of the learner and possible optimizations. Finally, Section 6 reports conclusions and open problems.

2. Preliminaries

We assume that the reader is familiar with the basics facts in the theory of automata and formal languages [16]. Let Σ be a finite alphabet: we denote by Σ^* the set of finite words on Σ, while ϵ denotes the empty word. Given $U, V \subseteq \Sigma^*$, let $U \oplus V = \{v \mid v \in U \setminus V \vee v \in V \setminus U\}$ be the symmetric difference of U and V. The word $u \in \Sigma^*$ is said a prefix of $v \in \Sigma^*$, written $u \sqsubseteq v$ if $v = uu'$ for some $u' \in \Sigma^*$. Given $X \subseteq \Sigma^*$, let $\mathsf{Prefs}(X) = \{u \mid \exists v \in X : u \sqsubseteq v\}$ be the set of prefixes of (a word in) X.

A *deterministic finite automaton* (DFA) over the finite input alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, E, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and E is a (partial) transition function mapping $Q \times \Sigma$ to Q. If E is complete, then $\mathcal{A}$ is said complete. The transition from the state $q \in Q$ on the letter $a \in \Sigma$ to the state $q' \in Q$ is denoted by the triple (q, a, q') or by $q \xrightarrow{a} q'$. Given a word $w = w_1 \dots w_n \in \Sigma^*$, we say that $\mathcal{A}$ admits a run on w if there exists a sequence of states $q_0 \dots q_n$ s.t. q_0 is the initial state and for each $0 < i \leq n$, $(q_{i-1}, w_i, q_i) \in E$. In this case, we write $q_0 \xrightarrow{w}_{\mathcal{A}} q_n$ and we say that $\mathcal{A}$ reads w. The run $q_0 \xrightarrow{w}_{\mathcal{A}} q_n$ is said accepting if $q_n \in F$. If $\mathcal{A}$ does not admit a run on $w \in \Sigma^*$, we say that $\mathcal{A}$ diverges on w and we write $\mathcal{A}(w) \uparrow$. The language accepted by $\mathcal{A}$ is the set of words $L(\mathcal{A}) \subseteq \Sigma^*$ such that $\mathcal{A}$ admits an accepting run on w. The state q is said *live* if there exists $w \in \mathsf{Prefs}(L(\mathcal{A}))$ s.t. $q_0 \xrightarrow{w}_{\mathcal{A}} q$. Given the language $L \subseteq \Sigma^*$ and the words $u, v \in \Sigma^*$ we use the notation $u =_L v$ to indicate that $u \in L \leftrightarrow v \in L$.

Given $L \subseteq \Sigma^*$, the Myhill-Nerode equivalence relation $\equiv_L$ on Σ^* is defined as: $u \equiv_L v$ if and only if for each $z \in \Sigma^*$, $uz \in L \leftrightarrow vz \in L$. Given $v \in \Sigma^*$, we denote by $[v]_{\equiv_L}$ the equivalence class of v with respect to the equivalence relation $\equiv_L$. A language is regular if and only if it is accepted by a DFA if and only if $\equiv_L$ has finite index. In particular, $\equiv_L$ can be used to build the DFA with the minimum number of states and transitions accepting L. Such a minimum DFA acceptor for L is unique up to isomorphism.

An *active learner* for regular languages in the *minimally adequate teacher* (MAT) framework is an algorithm that builds the minimum (complete) DFA accepting a certain (unknown) regular language L on an input finite alphabet Σ accessing to a teacher which answers two types of queries:

1. *membership* queries, consisting of a single word $w \in \Sigma^*$, to which the teacher will reply with a boolean answer, indicating if $w \in L$ or $w \notin L$, where $L \subseteq \Sigma^*$ is the target language;

2. *equivalence* queries, consisting of an hypothesis DFA $\mathcal{A}$, to which the teacher will reply *yes*, if $L(\mathcal{A}) = L$ and *no* otherwise providing a counterexample $w \in L(\mathcal{A}) \oplus L$.

3. Active Learning Incomplete Deterministic Finite State Automata

In this section we develop a DFA learner that actively interacts with a teacher by asking membership and equivalence queries to build the minimum DFA that reads $\mathsf{Prefs}(L)$ and accepts L, where L is a regular target language.

The learner is illustrated in Algorithm 1 and consists of an initialization phase followed by a main while loop. The initialization phase at Lines 1-2 initializes the hypothesis DFA $\mathcal{A}$ to a DFA containing just the initial state $q_0 = \epsilon$ (being final only if $\epsilon \in L$) and no transition. In particular, states will be named after their access strings (or representative prefix strings) and the following invariants will be mantained throughout the whole procedure:

1. $Q \subseteq \mathsf{Prefs}(L)$ is prefix-closed, $Q \cap L = F$ and for each $u, v \in Q : u \not\equiv_L v$
2. $\forall u \xrightarrow{a} v$ in $E : ua \in \mathsf{Prefs}(L)$

Moreover for each $ua \in Q$ the set E will contain the edge $u \xrightarrow{a} ua$. Note that for each $ua \in Q$, the minimum DFA for L needs to contain the edge $[u]_{\equiv_L} \xrightarrow{a} [ua]_{\equiv_L}$, being $u \not\equiv_L ua \in \mathsf{Prefs}(L)$. Therefore, the set of edges $\{u \xrightarrow{a} ua \mid ua \in Q\}$ is called the set of *must*-edges and denoted E_{must}. In contrast, the set of edges $E \setminus E_{must} = \{u \xrightarrow{a} v \mid u, v \in Q \wedge v \neq ua\}$ is called the set of *may* edges and denoted E_{may}. As far as may-edges is concerned, note that if $u \xrightarrow{a} v$ is a may edge, then the minimum DFA for L still needs to admit an edge out of $[u]_{\equiv_L}$ reading a (being $ua \in \mathsf{Prefs}(L)$) but the target of such a transition could be different from $[v]_{\equiv_L}$. Intuitively, the learner will eventually learn the "right" target for each may-edge. In particular, as soon as it recognizes that a temptative target v for the may-edge $u \xrightarrow{a} v$ is wrong (being $ua \not\equiv_L v$ entailing that $[u]_{\equiv_L} \xrightarrow{a} [v]_{\equiv_L}$ is not an edge of the minimum DFA for L) , it puts $u \xrightarrow{a} v$ in the so-called set of *forbidden edges* $E^{\oslash}$, in order not to repeat the mistake again. The set $E^{\oslash}$ of forbidden edges is initialized to the empty set in Line 2 .

Algorithm 1: DFA_Learn

Input: Equivalence oracle EQ_L and membership oracle MQ_L answering equivalence and membership queries for a target regular language L

Output: Minimum (possibly incomplete) DFA $\mathcal{A}$ for L

```
// Initialize A = (Q, q0, E, F) and the set of forbidden edges E^⊘
1 if MQ_L(ε) then F ← {ε} else F ← ∅;
2 q0 ← ε; Q ← {q0}; E ← ∅; E^⊘ ← ∅; A ← (Q, q0, E, F)
3 while true do
      // Invariants : (1) Q ⊆ Prefs(L), L ∩ Q = F, ∀u -a-> v in E : ua ∈ Prefs(L)
      (2) ∀ u, v ∈ Q : u ≢_L v and ∀ u -a-> v in E^⊘ : ua ≢_L v.
4     (b, w) ← EQ_L(A);                                   // Step 1: Test Equivalence
5     if b = true then
6         return A;
7     else
8         (A, E^⊘) ← ProcessCEX(w, MQ_L, A, E^⊘);        // Step 2: Cex Processing. Adds
          u -a-> x, either to read ua or to redirect u -a-> v (being ua ≢_L v)
```

Each iteration of the main while-loop at Line 3 consists of two steps. The first step at Line 4 relies on the equivalence oracle EQ_L to check the equivalence between L and $L(\mathcal{A})$, where $\mathcal{A}$ is the hypothesis DFA

built so far. If the answer of the equivalence oracle is positive, then the algorithm terminates returning $\mathcal{A}$. Otherwise, besides its negative answer EQ_L provides a counterexample $w \in L \oplus L(\mathcal{A})$. Such a counterexample is given in input to the procedure ProcessCEX (where CEX is used as a shorthand abbreviation for counterexample) called at Line 8. The second step consists exactly in executing the procedure ProcessCEX$(w, \mathcal{A}, E^{\oslash})$, that adds to $\mathcal{A}$ an edge $u \xrightarrow{a} x$ and possibly modifies also $E^{\oslash}$. In particular the added edge $u \xrightarrow{a} x$ is:

- Either a may-edge to process a previously unreadable prefix ua of some word in L,
- or a may-edge resulting from the redirection of a wrong may-edge e' (that is removed from the hypothesis and put in $E^{\oslash}$),
- or else a must-edge $u \xrightarrow{a} ua$ to the newly created node ua. The must-edge $u \xrightarrow{a} ua$ is created only if $E^{\oslash}$ prevents to redirect the edge out from u on a to any other existing node $v \in Q$ (witnessing that $ua \not\equiv_L v$ for each existing node v in Q).

In other words, each call to the procedure ProcessCEX (within an iteration of the main loop) either learns that a node $u \in Q$ admits an edge on $a \in \Sigma$, or learns that the target v of a transition (out from $u \in Q$ on $a \in \Sigma$) is wrong, or else learns a new node $ua \in Q$. The next subsection illustrates in detail the crucial procedure ProcessCEX outlined above.

3.1. Counterexample Processing

The ProcessCEX procedure is illustrated in Algorithm 2, below. Let $w \in L \oplus L(\mathcal{A})$ be the counterexample returned by the equivalence oracle as a result of testing if L is equal to the language of the hypothesis DFA $\mathcal{A}$.

Algorithm 2: ProcessCEX Procedure

Input: $w \in L \oplus L(\mathcal{A})$, Oracle MQ_L, Hypothesis DFA $\mathcal{A}$, set of forbidden edges $E^{\oslash} \subseteq 2^{Q \times \Sigma \times Q}$
1 , **Output:** $E^{\oslash}$ (possibly augmented) and modified hypothesis $\mathcal{A}$ (with one more/different edge)
2 Let $w = uav$ where u is the longest prefix of w in Q
3 **if** $\mathcal{A}(ua) \uparrow$ // **Case 1:** Missing transition on a out from $u \in Q$ ($\mathcal{A}(ua) \uparrow$)
4 **then**
5 Pick x in Q; $E \leftarrow E \cup u \xrightarrow{a} x$; **return** $\langle (Q, q_0, F, E), E^{\oslash} \rangle$
6 Let $\epsilon \xrightarrow{ua}_{\mathcal{A}} z$ be the run of $\mathcal{A}$ on ua
7 **if** $\mathsf{MQ}_L(uav) \neq \mathsf{MQ}_L(zv)$ // **Case 2.** $uav \not\equiv_L zv \Rightarrow ua \not\equiv_L z$: Forbid z as a target for the edge out from u on a and redirect the latter edge
8 **then**
9 $E^{\oslash} \leftarrow E^{\oslash} \cup \{u \xrightarrow{a} z\}$; $E \leftarrow E \setminus \{u \xrightarrow{a} z\}$ // Forbid the target z for $E(u, a)$
10 **if** $\exists x \in Q : u \xrightarrow{a} x \notin E^{\oslash} \wedge x =_L ua$ // Look for a new target
11 **then**
12 $E \leftarrow E \cup \{u \xrightarrow{a} x\}$
13 **else**
14 $Q \leftarrow Q \cup \{ua\}$; $E \leftarrow E \cup \{u \xrightarrow{a} ua\}$; **if** $\mathsf{MQ}_L(ua)$ **then** $F \leftarrow F \cup \{ua\}$;
15 **return** $(Q, q_0, F, E), E^{\oslash})$;
16 **else**
17 ProcessCEX$(zv, \mathcal{A}, E^{\oslash})$ // **Case 3.** Recursive call on cex zv (less may tr.)

Line 2 in ProcessCEX determines the longest prefix u of w s.t. the run of $\mathcal{A}$ on u contains no may edges[1]. Being $E_{must} = \{x \xrightarrow{a} xa \mid xa \in Q\}$, this amounts at determining the longest prefix u of

[1] i.e. it contains only must transitions or it contains no transitions at all

$w = uu'$ s.t. $u \in Q$. Note that $L \cap Q = F$ entails that u is a strict prefix of $w = uu' = uav$, since $u \in Q$ can not be a counterexample.

There are two cases to consider depending on whether $\mathcal{A}$ can read ua or not. If $\mathcal{A}$ can not read ua, then it can neither read $w = uav$ and $w \in L(\mathcal{A}) \oplus L$ entails $w \in L$. In this case, Line 5 adds to the hypothesis an edge to read a out from u and therefore to process $ua \in \mathsf{Prefs}(L)$. Line 5 returns the modified hypothesis DFA, where the target for the new transition is arbitrarily chosen in Q.

Otherwise, $\mathcal{A}$ reads $ua \sqsubset uav = w$. In this case the last transition in the run of $\mathcal{A}$ on ua is necessarily a may-transition[2]. Let $e = u \xrightarrow{a} z$ such a first may transition in the run of $\mathcal{A}$ on $w = uav$.

If $uav \neq_L zv$, then $ua \not\equiv_L z$ and e is wrong. Therefore, Line 8 inserts e in $E^{\oslash}$. The following if-statement at Line 10 determines whether redirecting e toward another existing node $x \in Q$ or creating the new node ua as a target for the transition out from u on a. In particular, a new node ua is created iff for each $x \in Q$, either $ua \neq_L x$ or the set of forbidden edges $E^{\oslash}$ contains $u \xrightarrow{a} x$, witnessing $ua \not\equiv_L x$. Therefore, the addition of ua to Q preserves the invariant stating that the nodes in Q are pairwise distinct representatives of the classes of $\Sigma^*/_{\equiv_L}$.

If ProcessCEX does not return neither at Line 5 nor at Line 15, then $uav =_L zv$ entails that zv is also a counterexample and ProcessCEX is recursively called on zv.

The termination of the recursive chain of calls to ProcessCEX is ensured by the fact that the number of may edges in the runs processing the corresponding input counterexamples is strictly decreasing[3]. Such an argument is formalized in the following Lemma 1.

Lemma 1 (CEX Processing). *Let $w \in L(\mathcal{A}) \oplus L$, where L is a regular language and $\mathcal{A} = (Q, q_0, F, E)$ is a DFA s.t.: Q is a prefix-closed subset of $\mathsf{Prefs}(L)$, $L \cap Q = F$, $E \supseteq \{u \xrightarrow{a} ua \mid ua \in Q\}$. Let u be the longest word in $\mathsf{Prefs}(w) \cap Q$ and let m be the number of may edges (i.e. edges $z \xrightarrow{b} z'$ with $z' \neq zb$) appearing on the run of $\mathcal{A}$ on w. Then, $w = uaw'$ and one of the following three cases apply:*

1. *$m = 0$ and $\mathcal{A}$ does not read $ua \in \mathsf{Prefs}(L)$.*
2. *$m > 0$ and $uaw' \not\equiv_L vw'$, where v is the node reached by the run of $\mathcal{A}$ on ua.*
3. *$vw' \in L(\mathcal{A}) \oplus L$ and the run of $\mathcal{A}$ on vw' contains $m - 1$ may edges.*

Proof. Let u be the longest prefix of $w \in L(\mathcal{A}) \oplus L$ s.t. $u \in Q$. Since $Q \cap L = F$, u is not a counterexample. Therefore $w = uaw'$, where $a \in \Sigma$ and $w' \in \Sigma^*$. Let m be the number of may edges $z \xrightarrow{b} z' \neq zb$ appearing on the run of $\mathcal{A}$ on w. There are two cases to consider depending on whether $\mathcal{A}$ reads ua or not. In the first case ($\mathcal{A}$ does not read ua), $w = uaw' \notin L(\mathcal{A})$, yielding $w \in L$ and $ua \in \mathsf{Prefs}(L)$. Moreover, since Q is prefix-closed and $E \supseteq \{u \xrightarrow{a} ua \mid ua \in Q\}$, $m = 0$. Hence, the first item in the lemma applies. In the second case ($\mathcal{A}$ reads ua) let $v \in Q$ be the node reached by the run of $\mathcal{A}$ on ua. Since u is the longest prefix of $w = uaw'$ in Q, we get $v \neq ua$. Therefore $u \xrightarrow{a} v$ is a may edge and $m > 0$. If $uaw' \not\equiv_L vw'$, the second item in the lemma applies. Otherwise, $uaw' \equiv_L vw'$. Being uaw' accepted iff vw' is accepted, we get that vw' is a counterexample. Moreover the run of $\mathcal{A}$ on vw' contains $m - 1$ may edges, since the run of $\mathcal{A}$ on $v \in Q$ contains no may edges. □

On the ground of Lemma 1, Theorem 2 below formally proves the termination of the recursive procedure ProcessCEX.

Theorem 2. *Let $w \in L(\mathcal{A}) \oplus L$, where L is a regular language and $\mathcal{A} = (Q, q_0, F, E)$ is a DFA s.t.: Q is a prefix-closed subset of $\mathsf{Prefs}(L)$, $L \cap Q = F$, $E \supseteq \{u \xrightarrow{a} ua \mid ua \in Q\}$. Let m be the number of may transitions contained in the run of $\mathcal{A}$ processing the longest prefix of w readable by $\mathcal{A}$. Then,* ProcessCEX*$(w, \mathcal{A}, E^{\oslash})$ terminates after $\mathcal{O}(m)$ recursive calls returning $\mathcal{A}_{out}, E^{\oslash}_{out}$ where:*

[2] otherwise, we would have $ua \in Q$

[3] Note that if the run of $\mathcal{A}$ on w contains only must transition, then $w \in Q$ entailing that w is not a counterexample since $L \cap Q = F$

- *Either $E^{\oslash} = E^{\oslash}_{out}$, $\mathcal{A}$ does not read $ua \in \mathsf{Prefs}(L)$ and $\mathcal{A}_{out}$ is obtained from $\mathcal{A}$ by adding a transition from u on $a \in \Sigma$ (to any $v \in Q$).*
- *Or $E^{\oslash}_{out} = E^{\oslash} \cup \{e\}$, where $e = u \xrightarrow{a} v$, $ua \not\equiv_L v$ and $\mathcal{A}_{out}$ is obtained from $\mathcal{A}$ by:*
  - *Either redirecting the edge out from u on a toward another node $z \in Q$, such that $u \xrightarrow{a} z \notin E^{\oslash}$*
  - *Or else by adding the node ua and the edge $u \xrightarrow{a} ua$*

Proof. Let m be the number of may edges (i.e. edges $z \xrightarrow{b} z'$ with $z' \neq zb$) appearing on the run of $\mathcal{A}$ on w. By Lemma 1, ProcessCEX($w = uaw', \mathcal{A}, E^{\oslash}$) terminates since entering Line 1 would induce a new recursive call ProcessCEX($vw', \mathcal{A}, E^{\oslash}$) on vw', where the run of $\mathcal{A}$ on vw' contains $m - 1$ may transition. In particular, if any induced recursive call to ProcessCEX($w = uaw', \mathcal{A}, E^{\oslash}$) enters Line 8, then $uaw' \neq_L vw'$ yielding $ua \not\equiv_L v$, where $e = u \xrightarrow{a} v \in E$ is the first may transition in the run of the hypothesis DFA on the counterexample $w = uaw'$. Line 7 adds e to $E^{\oslash}$ obtaining $E^{\oslash}_{out}$. If there exists $z \in Q$ such that $e' = u \xrightarrow{a} z \notin E^{\oslash}_{out}$, then E is updated by removing e and adding e'. Otherwise, ua is added to Q and E is updated by removing e and adding the transition from u to ua reading a. Finally, the procedure terminates returning $E^{\oslash}_{out} = E^{\oslash} \cup \{e\}$ and the updated hypothesis.

If Line 8 is never enterd (by any induced recursive call to ProcessCEX), then $m = 0$ when the last recursive call to ProcessCEX is performed on the counterexample $w = uaw'$, where$u$ is the maximum prefix of w in Q. By Lemma 1, $\mathcal{A}$ does not read $ua \in \mathsf{Prefs}(L)$. Line 5 adds to the hypothesis DFA a transition from u on $a \in \Sigma$ (to any $v \in Q$) to read $ua \in \mathsf{Prefs}(L)$. Finally, the procedure terminates returning $E^{\oslash}_{out} = E^{\oslash}$ and the updated hypothesis.

□

We conclude this section illustrating the learning process of Algorithm 1 on a simple example.

Example 3. Consider the regular language $L = (ab)^*c^*$ on the alphabeth $\Sigma = \{a, b, c\}$ recognized by the minimum (non complete) DFA depicted in Figure 1.

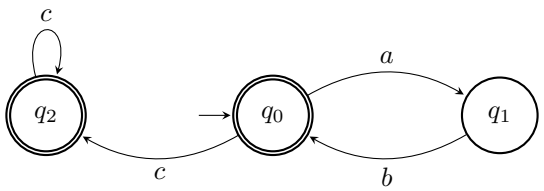

Figure 1: Minimum DFA $\mathcal{A}$ recognizing the language $L = (ab)^*c^*$

The algorithm initializes $\mathcal{A}$ to a DFA containing just the initial state ϵ and no transition. Moreover, $\epsilon \in F$ being $\epsilon \in L$. Figure 2 below illustrates the DFA resulting from such an initialization process.

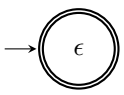

Figure 2: Hypothesis DFA $\mathcal{A}$ after the initialization phase. $E^{\oslash} = \emptyset$

Asked whether $L(\mathcal{A}) = L$ the teacher answers no and returns a counterexample. Suppose that such a returned counterexample is ab. Since $\mathcal{A}$ can read only the prefix ϵ of ab, a transition out from ϵ reading a is created at Line 5 of the ProcessCEX procedure. The only target $x \in Q$ available for a may edge out from ϵ on a is $x = \epsilon$ and the resulting hypothesis is provided below. In particular, throughout this example we will use a dotted notation for may edges, yielding their immediate visualization and distinction from must edges.

Let aab be the following counterexample provided by the teacher. The the first may edge in the run of $\mathcal{A}$ reading aab is $\epsilon \xrightarrow{a} \epsilon$. Being $\mathsf{MQ}_L(aab) \neq \mathsf{MQ}_L(\epsilon \cdot ab)$, the then-branch of the if statement at Line

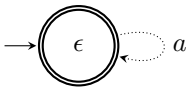

Figure 3: Hypothesis DFA $\mathcal{A}$ after the first iteration of Algorithm 1. $E^{\oslash} = \emptyset$.

7 of ProcessCEX is entered and the may edge $\epsilon \xrightarrow{a} \epsilon$ is inserted in the set of forbidden edges. Hence, ϵ can not be anymore the target of a transition out from ϵ on a and a new node a is created as the target for the must-edge $\epsilon \xrightarrow{a} a$. Figure 4 below illustrates the hypothesis DFA after the second iteration of the learning algorithm.

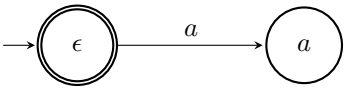

Figure 4: Hypothesis DFA $\mathcal{A}$ after the second iteration of Algorithm 1. $E^{\oslash} = \{\epsilon \xrightarrow{a} \epsilon\}$.

Suppose that the teacher returns again the counterexample ab when asked if $L(\mathcal{A}) = L$. Then, a is the longest prefix of ab readable by $\mathcal{A}$ using no may edges and a transition out from a on b is generated at Line 5 of the ProcessCEX procedure (picking any node $x \in Q$ as a target). Suppose that ϵ is chosen as a target of such a transition and the may-edge $a \xrightarrow{b} \epsilon$ is added to $\mathcal{A}$.

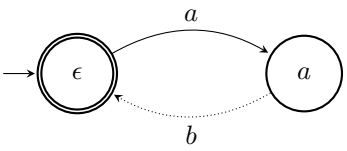

Figure 5: Hypothesis DFA $\mathcal{A}$ after the third iteration of Algorithm 1. May-edges are dotted and $E^{\oslash} = \{\epsilon \xrightarrow{a} \epsilon\}$.

Figure 5 above illustrates the hypothesis DFA after the third iteration of Algorithm 1. Note that, in general, the set of must-edges $\{u \xrightarrow{a} ua \mid ua \in Q\}$ on the prefix-closed set of nodes Q yields a tree. At this stage, such a tree is composed by the only must-edge $\epsilon \xrightarrow{a} a$

Let $\mathsf{EQ}_L(\mathcal{A}) = (0, abc)$. The longest prefix of abc in Q is a and the first may edge encountered reading abc is $a \xrightarrow{b} \epsilon$. Being $\mathsf{QM}_L(abc) = \mathsf{QM}_L(\epsilon \cdot c)$, the else-branch of the if statement at Line 7 of ProcessCEX is entered and the procedure is recursively called on the counterexample c (with less may transitions). The longest prefix of c readable by $\mathcal{A}$ using no may edge is ϵ and $\mathcal{A}(\epsilon \cdot c) \uparrow$. Therefore, the may-edge $\epsilon \xrightarrow{c} \epsilon$ is added to $\mathcal{A}$, where the target ϵ is randomly selected among the nodes in Q. The resulting hypothesis DFA is illustrated in Figure 6.

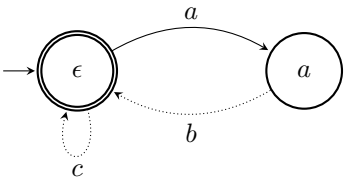

Figure 6: Hypothesis DFA after the fourth iteration of Algorithm 1. May-edges are dotted and $E^{\oslash} = \{\epsilon \xrightarrow{a} \epsilon\}$.

Asked whether $L(\mathcal{A}) = L$ the teacher answers again negatively. Let $abcabc$ the counterexample provided. The first may-edge in the run of $\mathcal{A}$ processing $abcabc$ is $a \xrightarrow{b} \epsilon$. Since $\mathsf{MQ}_L(abcabc) = \mathsf{MQ}_L(cabc)$, the procedure ProcessCEX is recursively called on the counterexample $cabc$ (processed by a run with less may transitions). The first may-edge in the run of $\mathcal{A}$ on $cabc$ is $\epsilon \xrightarrow{c} \epsilon$. This time $\mathsf{MQ}_L(cabc) \neq \mathsf{MQ}_L(abc)$, witnessing that such a may transition is wrong.

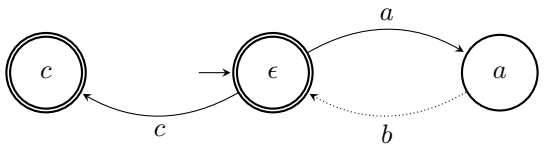

Figure 7: Hypothesis DFA after the fifth iteration of Algorithm 1. May-edges are dotted and $E^{\oslash} = \{\epsilon \xrightarrow{a} \epsilon, \epsilon \xrightarrow{c} \epsilon\}$.

Therefore, $\epsilon \xrightarrow{c} \epsilon$ is added to $E^{\oslash}$. Since there are no more possible targets in Q for the transition on c out of ϵ (being $\mathsf{MQ}_L(c) \neq \mathsf{MQ}_L(a)$) , $\mathcal{A}$ is added of a new final state c and a new must-edge $\epsilon \xrightarrow{c} c$, as illustrated in Figure 7.

Let $cccc$ be the counterexample provided to the hypothesis DFA in Figure 7. The longest prefix of $cccc$ in Q is c and $\mathcal{A}$ can not read cc. Therefore, a new transition $c \xrightarrow{c} a$ is created to read c out from c, picking the target node a randomly in Q, as depicted in Figure 8.

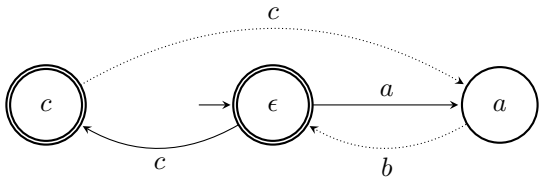

Figure 8: Hypothesis DFA after the sixth iteration of Algorithm 1. May-edges are dotted and $E^{\oslash} = \{\epsilon \xrightarrow{a} \epsilon, \epsilon \xrightarrow{c} \epsilon\}$.

Let $w = cc$ be next counterexample provided by the teacher. the longest prefix of cc belonging to Q is c and $c \xrightarrow{c} a$ is the first may transition encountered reading cc. Being $\mathsf{QM}_L(cc \cdot \epsilon) \neq \mathsf{QM}_L(a \cdot \epsilon)$ we enter the then-branch of the if statement at Line 7 of ProcessCEX. Then, $c \xrightarrow{c} a$ is inserted into $E^{\oslash}$. Now, both ϵ and c can be selected as the target for the transition out from c on c, being $cc =_L \epsilon \wedge c \xrightarrow{c} \epsilon \notin E^{\oslash}$ and $cc =_L c \wedge c \xrightarrow{c} c \notin E^{\oslash}$. If c is selected as a target and $\mathcal{A}$ is added of the may transition $c \xrightarrow{c} c$, EQ_L finally answers positively and the algorithm terminates. Otherwise, the termination is deleted of one iteration.

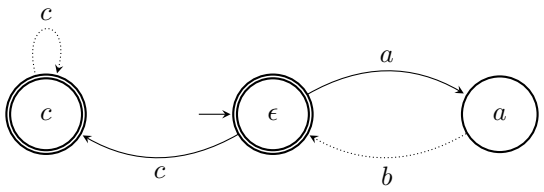

Figure 9: Final hypothesis DFA $\mathcal{A}$ provided and approved by the equivalence oracle.

4. Correctness and Complexity

In this Section we prove the correctness and the complexity of the introduced algorithm for learning (possibly incomplete) automata. The correctness is grounded in the three crucial invariants provided in the following Lemma 4. In particular, Invariant (I) states for each pair of states $u, v \in Q$: $u \not\equiv_L v$. Therefore, if $\mathcal{A}_L = (Q_L, \epsilon, E_L, F_L)$ is the target DFA, it globally holds that $Q \subseteq Q_L$, and eventually $Q = Q_L$. Invariant (II) states that for each forbidden edge $u \xrightarrow{a} v$ in $E^{\oslash}$: $ua \not\equiv_L v$. Hence, it globally holds that $E^{\oslash} \subseteq (Q \times \Sigma \times Q) \setminus E_L$. Finally, the third invariant ensures that each state $u \in Q$ admits a transition on $a \in \Sigma$ only if $ua \in \mathsf{Prefs}(L)$.

Given these invariants, each iteration of the main loop improves the hypothesis DFA by either learning a new node $u \in Q$, or by learning that a node $u \in Q$ admits an edge on some new letter $a \in \Sigma$, or finally learning that the target of a transition is wrong, yielding a new forbidden edge.

Lemma 4 (Invariants)**.** *The following invariants hold upon each iteration of the main while-loop in* DFA_Learn*:*

(I) For each pair of states $u, v \in Q$: $u \not\equiv_L v$.

(II) For each forbidden edge $u \xrightarrow{a} v$ in $E^{\emptyset}$: $ua \not\equiv_L v$.

(III) $Q \subseteq \mathsf{Prefs}(L)$ is prefix-closed, $L \cap Q = F$, $E \supseteq \{u \xrightarrow{a} ua \,|\, ua \in Q\}$ and for each edge $u \xrightarrow{a} v \in E$: $ua \in \mathsf{Prefs}(L)$.

Proof. Base Case Before entering the main loop at Line 3, $Q = \{\epsilon\}$, $E = \emptyset$, $E^{\emptyset} = \emptyset$ and $\epsilon \in F$ only if $\epsilon \in L$. Therefore, the three invariants hold.

Inductive Step Let $\mathcal{A}_k = (Q_k, \epsilon, E_k, F_k)$ be the hypothesis DFA on entering the $k+1$-th iteration of the main loop, for $k \geq 0$. Let $E^{\emptyset}_k$ be the set of forbidden transitions on entering the $k+1$-th iteration of the main loop and suppose that the equivalence oracle provides the counterexample $w \in L \oplus L(\mathcal{A}_k)$. By Theorem 2, ProcessCEX($w, \mathcal{A}_k, E^{\emptyset}_k$) terminates returning $\mathcal{A}_{k+1}, E^{\emptyset}_{k+1}$ where:

- Either $E^{\emptyset}_k = E^{\emptyset}_{k+1}$, $\mathcal{A}_k$ does not read $ua \in \mathsf{Prefs}(L)$ and $\mathcal{A}_{k+1}$ is obtained from $\mathcal{A}_k$ by adding a transition from u on $a \in \Sigma$ (to any $v \in Q_k$). Therefore, $Q_{k+1} = Q_k, F_{k+1} = F_k$, $E^{\emptyset}_k = E^{\emptyset}_{k+1}$ and $E_{k+1} = E_k \cup \{u \xrightarrow{a} v\}$, where $u, v \in Q_k$ and $ua \in \mathsf{Prefs}(L)$. Hence, the three invariants are still valid upon the termination of the $k+1$-th iteration of the main loop.

- Or $E^{\emptyset}_{k+1} = E^{\emptyset}_k \cup \{e\}$, where $e = u \xrightarrow{a} v$, $ua \not\equiv_L v$ and $\mathcal{A}_{k+1}$ is obtained from $\mathcal{A}_k$ by (1) either redirecting the edge out from u on a toward another node $z \in Q_k$, such that $u \xrightarrow{a} z \notin E^{\emptyset}_{k+1}$ and $z =_L ua$, or else (2) by adding the node ua and the edge $u \xrightarrow{a} ua$.

  Being $E^{\emptyset}_k = E^{\emptyset}_{k+1}$, where $\mathcal{A}_k$ does not read $ua \in \mathsf{Prefs}(L)$, the second invariant holds at the end of the exection of the $k+1$-th iteration of the main loop. As far as $\mathcal{A}_{k+1}$ is concerned, the first and the third invariant hold by inductive hypothesis if $\mathcal{A}_{k+1}$ is obtained from $\mathcal{A}_k$ by redirecting an existing edge. Otherwise, $Q_{k+1} = Q_k \cup \{ua\}$ remains prefix-closed by inductive hypothesis and since $u \in Q_k$. $ua \in \mathsf{Prefs}(L)$, since the inductive hypothesis was ensuring $ua \in \mathsf{Prefs}(L)$ for each edge out on u on a in E_k. ua is added to F_{k+1} only if $ua \in L$. Therefore, $Q_{k+1} \subseteq \mathsf{Prefs}(L)$, $L \cap Q_{k+1} = F_{k+1}$, $E_{k+1} \supseteq \{u \xrightarrow{a} ua \,|\, ua \in Q_{k+1}\}$ and for each $u \xrightarrow{a} v \in E_{k+1}$: $ua \in \mathsf{Prefs}(L)$. Since $ua \not\equiv_L v$ and for each other node $z \in Q_k$, $u \xrightarrow{a} z \in E^{\emptyset}_k$ or $z \neq_L ua$, by inductive hypothesis we get that for each pair of different state $x, y \in Q_{k+1}$, $x \not\equiv_L y$. We conclude that the three invariants hold on exiting the $k+1$-th iteration of the main loop.

□

As far as complexity is concerned, the number of transitions of the main loop is polynomial in the size of $\mathcal{A}_L$ since there are at most $|Q_L|$ nodes to discover and at most $|\Sigma|$ letter to read from each node. Finally, for each edge $u \xrightarrow{a} v \in E_L$ there are at most $|Q_L|$ wrong targets $z \neq v$ to detect. By Theorem 2 the cost of ProcessCEX is polynomial in the size of the input. Therefore, we get:

Theorem 5. *Let L be a regular language, let EQ_L be an equivalence oracle for L and let MQ_L be a membership oracle for L. The algorithm $\mathsf{DFA\_Learn}(\mathsf{EQ}_L, \mathsf{MQ}_L)$ learns the minimum (possibly incomplete) DFA $\mathcal{A}_L$ accepting L and reading $\mathsf{Prefs}(L)$ with polynomially many queries to the teacher, with respect to the size of $\mathcal{A}_L$ and the length of the maximum counterexample w provided by the teacher.*

Proof. Let n be the number of states of the target DFA, let m be the number of transitions. By Lemma 4, and Theorem 2 each iteration of the main loop:

- either adds a representative ua of a newly discovered equivalence class $[ua]_{\cong_L}$, meaning that $[ua]_{\equiv_L} \neq [x]_{\equiv_L}$ for any other access string in Q.
- or discovers that $ua \subseteq \mathsf{Prefs}(L)$, while the hypothesis built so far was not admitting any edge for reading a out from $u \in Q$.
- or else adds an edge $u \xrightarrow{a} v$ to $E^{\oslash}$, discovering that the target of the transition out from $[u]_{\equiv_L}$ on a (in the canonical DFA for the target language) is not $[v]_{\equiv_L}$, being $ua \not\equiv_L v$.

Since there are at most n nodes and $|\Sigma|n$ transitions to discover, and at most $|\Sigma|n^2$ wrong targets (of existing transitions) to eliminate, the algorithm terminates after $\mathcal{O}(n^2)$ iterations. Finally, each iteration rely on an equivalence query and a number of membership queries bounded by the may transitions of the run of the current hypothesis on the provided counterexample w. □

5. On the Number of Queries to the Oracles

We finally discuss on the number of queries to the oracles required by the proposed learner and on possible optimizations.

Equivalence Queries The number of queries to the equivalence oracle corresponds to the number of iterations of the main loop and is bounded by:

- the size of the target DFA (number of states and transitions to discover)
- the number of redirections of each discovered transition

It is possible to optimize the number of queries to the equivalence oracle yielding each iteration of the main algorithm to learn either a new state or a new letter to be read out from some state or else to disambiguate among multiple targets for a may transition (eliminating all of them but one). To avoid multiple redirections of may transitions, we proceed as follows. As soon as a suffix z witnesses that an edge $e = u \xrightarrow{a} v$ is wrong (being $uaz \neq_L vz$), the pair $\langle e, z\rangle$—rather than the only edge e—is inserted into the set of forbidden edges $E^{\oslash}$. Given such an enriched $E^{\oslash}$, it is possible to disambiguate between two target candidates, say x, x', for a may-edge reading the letter b out of y (where $x =_L yb$ and $x' =_L yb$) as follows. Being x, x' two distinct nodes s.t. $x =_L x'$, $E^{\oslash}$ needs to contain a forbidden edge preventing to merge x with x'. In particular, $E^{\oslash}$ needs to contain a pair $\langle p \xrightarrow{d} q, s\rangle$, where either $pd = x, q = x'$ or $pd = x', q = x$. Therefore $xs \neq_L x's$ and checking whether ybs is accepted or not eliminates either x or x' as a target[4] for the may-edge out from y on b. Iterating such a reasoning yields a unique target for a given may edge out from an intial set of multiple possible candidates. The outlined idea is further illustrated and formalized in the pseudocode of the procedure OptimizedProcessCEX, given in Figure 3.

Example 6. As an example, for the hypothesis DFA of Figure 7, we would have inserted into the set $E^{\oslash}$ the pair $\langle \epsilon \xrightarrow{c} \epsilon, abc\rangle$. This way, when later (in the last iteration) the may transition out of c reading c is redirected for the first time, it is possible to use $E^{\oslash}$ to disambiguate between the two possible targets c and ϵ by using the suffix abc. In this case, $\mathsf{MQ}_L(ccabc) = 0$ and only one between $\epsilon \cdot abc$ and $c \cdot abc$ is also rejected inducing the only target c for the may transition $c \xrightarrow{c} c$.

Membership Queries The number of recursive calls to the procedure for counterexample processing is linear in the length of the input counterexample. It is possible to use a binary-search strategy similar to the one introduced by Rivest and Shapiro in [18] to have a number of recursive calls (and queries to the membership oracles) being logarithmic in the length of the input counterexample.

[4] This resembles so called pseudotransitivity in [17]

Algorithm 3: OptimizedProcessCEX Procedure

Input: counterexample w, hypothesis DFA $\mathcal{A}$, set of forbidden edges $E^{\oslash} \subseteq 2^{Q \times \Sigma \times Q}$
Output: $E^{\oslash}$ (possibly augmented) and modified hypothesis $\mathcal{A}$ (with one more/different edge)

Let $w = uav$ where u is the longest prefix of w in Q
if $\mathcal{A}(ua) \uparrow$ **then**
  Pick x in Q; $E \leftarrow E \cup u \xrightarrow{a} x$; **return** $(Q, q_0, F, E), E^{\oslash}$
Let $\epsilon \overset{ua}{\rightsquigarrow}_{\mathcal{A}} z$ be the run of $\mathcal{A}$ on ua
if $\mathsf{MQ}_L(uav) \neq \mathsf{MQ}_L(zv)$ // Redirection of $u \xrightarrow{a} z$ yields a new must or may edge
then
  $E^{\oslash} \leftarrow E^{\oslash} \cup \{\langle u \xrightarrow{a} z, v\rangle\}$; $E \leftarrow E \setminus \{u \xrightarrow{a} z\}$
  if $\nexists x \in Q : \langle u \xrightarrow{a} x, \_\rangle \notin E^{\oslash} \wedge \mathsf{MQ}_L(x) = \mathsf{MQ}_L(ua)$ // New must-edge is necessary
  then
    $Q \leftarrow Q \cup \{ua\}$; $E \leftarrow E \cup \{u \xrightarrow{a} ua\}$; **if** $\mathsf{MQ}_L(ua)$ **then** $F \leftarrow F \cup \{ua\}$;
  else
    // Disambiguate the target for the may transition out of u on a
    while $\exists x, x' \in Q : (u \xrightarrow{a} x \notin E^{\oslash}, ua =_L x) \wedge (u \xrightarrow{a} x' \notin E^{\oslash}, ua =_L x')$ **do**
      // $x, x' \in Q \Rightarrow E^{\oslash}$ contains an edge yielding their separation
      Pick $\langle p \xrightarrow{b} q, s\rangle \in E^{\oslash} : (pb = x \wedge q = x') \vee (pb = x' \wedge q = x)$
      if $\mathsf{MQ}_L(uas) = \mathsf{MQ}_L(xs)$ **then**
        $E^{\oslash} \leftarrow E^{\oslash} \cup \{\langle u \xrightarrow{a} x', s\rangle\}$
      else
        $E^{\oslash} \leftarrow E^{\oslash} \cup \{\langle u \xrightarrow{a} x, s\rangle\}$
    Pick the unique $x \in Q : \langle u \xrightarrow{a} x, \_\rangle \notin E^{\oslash} \wedge \mathsf{MQ}_L(x) = \mathsf{MQ}_L(ua)$
    $E \leftarrow E \cup \{u \xrightarrow{a} x\}$ // New may edge having x as unique possible target
  return $(Q, q_0, F, E), E^{\oslash})$;
else
  ProcessCEX$(zv, \mathcal{A}, E^{\oslash})$

6. Conclusions

We have introduced an active learner in the MAT framework for learning possibly incomplete DFAs. Such a learner does not rely on having a complete transition function for the target unknown automaton and learns both the alphabet and the transitions to live states just relying on standard membership and equivalence queries. We aim at generalizing the proposed procedure for learning classes of formal languages whose canonical models are possibly not complete automata, as Wheeler languages.

Declaration on Generative AI

No generative AI has been employed during the preparation of this work.

References

[1] E. Fernando, S. Rubin, R. Gopinath, Example-free learning of regular languages with prefix queries, 2025. URL: https://arxiv.org/abs/2504.02170. arXiv:2504.02170.

[2] D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation 75 (1987) 87–106. URL: https://www.sciencedirect.com/science/article/pii/0890540187900526. doi:https://doi.org/10.1016/0890-5401(87)90052-6.
[3] F. Vaandrager, Model learning, Commun. ACM 60 (2017) 86–95. URL: https://doi.org/10.1145/2967606. doi:10.1145/2967606.
[4] M. Leucker, Learning meets verification, in: Lecture Notes in Computer Science, volume 4709, Springer, Springer, 2007, pp. 127–151.
[5] O. Sankur, Timed automata verification and synthesis via finite automata learning, in: Tools and Algorithms for the Construction and Analysis of Systems: 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings, Part II, Springer-Verlag, Berlin, Heidelberg, 2023, p. 329–349. URL: https://doi.org/10.1007/978-3-031-30820-8_21. doi:10.1007/978-3-031-30820-8_21.
[6] M. Balachander, E. Filiot, J.-F. Raskin, LTL reactive synthesis with a few hints, in: Tools and Algorithms for the Construction and Analysis of Systems: 29th International Conference, TACAS 2023, Springer-Verlag, Berlin, Heidelberg, 2023, p. 309–328. URL: https://doi.org/10.1007/978-3-031-30820-8_20. doi:10.1007/978-3-031-30820-8_20.
[7] P. Fiterău-Broştean, R. Janssen, F. Vaandrager, Combining model learning and model checking to analyze tcp implementations, in: S. Chaudhuri, A. Farzan (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2016, pp. 454–471.
[8] H. Raffelt, B. Steffen, T. Berg, Learnlib: a library for automata learning and experimentation, FMICS '05, Association for Computing Machinery, New York, NY, USA, 2005. URL: https://doi.org/10.1145/1081180.1081189. doi:10.1145/1081180.1081189.
[9] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, D. R. Piegdon, libalf: the automata learning framework, in: Proceedings of the 22nd International Conference on Computer Aided Verification, CAV'10, Springer-Verlag, Berlin, Heidelberg, 2010, p. 360–364. URL: https://doi.org/10.1007/978-3-642-14295-6_32. doi:10.1007/978-3-642-14295-6_32.
[10] F. Aarts, B. Jonsson, J. Uijen, Generating models of infinite-state communication protocols using regular inference with abstraction, in: Proceedings of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems, ICTSS'10, Springer-Verlag, Berlin, Heidelberg, 2010, p. 188–204.
[11] S. Marksteiner, D. Schögler, M. Sirjani, M. Sjödin, Learning single and compound-protocol automata and checking behavioral equivalences, International Journal on Software Tools for Technology Transfer 27 (2025) 35–52. doi:10.1007/s10009-025-00797-y.
[12] D. Peled, M. Vardi, M. Yannakakis, Black box checking, Journal of Automata, Languages and Combinatorics 7 (2002) 225–246. doi:10.1007/978-0-387-35578-8_13.
[13] D. Angluin, A note on the number of queries needed to identify regular languages, Information and Control 51 (1981) 76–87. URL: https://www.sciencedirect.com/science/article/pii/S0019995881900905. doi:https://doi.org/10.1016/S0019-9958(81)90090-5.
[14] O. H. Ibarra, T. Jiang, Learning regular languages from counterexamples, J. Comput. Syst. Sci. 43 (1991) 299–316. URL: https://api.semanticscholar.org/CorpusID:275689753.
[15] J. Alanko, G. D'Agostino, A. Policriti, N. Prezza, Wheeler languages, Information and Computation 281 (2021) 104820. URL: https://www.sciencedirect.com/science/article/pii/S0890540121001504. doi:https://doi.org/10.1016/j.ic.2021.104820.
[16] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation (2nd Edition), volume 32, 2001. doi:10.1145/568438.568455.
[17] F. Vaandrager, B. Garhewal, J. Rot, T. Wißmann, A new approach for active automata learning based on apartness, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer International Publishing, Cham, 2022, pp. 223–243.
[18] R. L. Rivest, R. E. Schapire, Inference of finite automata using homing sequences, in: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, STOC '89, New York, NY, USA, 1989, p. 411–420. URL: https://doi.org/10.1145/73007.73047. doi:10.1145/73007.73047.