ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200367

Learning Deterministic Automata on Infinite Words

Jakub Michaliszyn and **Jan Otop**[1]

Abstract. We study active learning of deterministic infinite-words automata. In our framework, the teacher answers not only membership and equivalence queries, but also provides the loop index of the target automaton on wv^ω, which is the minimal number of letters of wv^ω past which the target automaton reaches the final cycle on wv^ω. We argue that in potential applications if one can answer Boolean part in membership (and equivalence) queries, one can compute the loop index as well.

Our framework is similar to the one of Angluin's L^*-algorithm, but the crucial difference is that the queries about the loop index depend on a particular automaton representing an ω-regular language. This allows us to bypass the NP-hardness coming from the minimisation problem for deterministic Büchi automata and provide a polynomial-time algorithm for learning deterministic Büchi automata. We adapt this algorithm to deterministic infinite-word weighted automata with LIMINF and LIMSUP value functions, which, treated as parity automata, can recognize all ω-regular languages.

1 Introduction

Automata are the fundamental computation model, which have a variety of applications ranging from being the baseline of computational complexity to applications in AI [23] and formal methods [14].

Many areas, including verification [20], benefit from the fact that various types of automata can be constructed automatically from data. Such a construction, called *learning*, typically cannot be performed effectively in a *passive* way, i.e., by only considering a list of positive and negative examples [18, 15]. In particular, such an algorithm for deterministic finite automata (DFA) could be used to break popular cryptographic systems [18].

To overcome this difficulty, the *active learning* framework was developed. In this framework, the learning algorithm can actively ask the *teacher* queries of two forms: does a given word belong to the (hidden) target regular language (*membership query*), and does the constructed automaton recognise the target language (*equivalence query*); it is essential that the answer to the latter query is supported with an appropriate counterexample where applicable. Angluin in her seminal paper [1] showed the L^* algorithm that learns DFA in polynomial time and asks polynomially many queries in the size of the automaton. The L^* algorithm for DFA is versatile; it has been extended to other types of automata such as weighted automata [9], tree automata [17, 21] or nominal automata [24].

The extension of L^* to infinite words has proved to be elusive. There are two main problems over infinite words. First, the right congruence relation, which is the crux of Angluin's algorithm, does not characterise ω-regular languages; there are different ω-regular languages defining the same right congruence relation. Second, in the infinite word case there is no notion of the canonical (syntactic) automaton. For some ω-languages there are several different minimal automata (see Section 3). For these reasons, extending the L^* algorithm to ω-regular languages is difficult.

An L^*-style algorithm for ω-regular languages has been given in [4]. However, the algorithm given there does not learn ω-automata; it returns a Family of DFA (FDFA), which is an alternative formalism to represent ω-regular languages. FDFA can be translated to deterministic ω-automata, but the translation may involve exponential blow-up.

The existence of polynomial-time ω-automata learning algorithm remains open. One of the possible reasons is that for deterministic Büchi automata (DBA), the minimalisation problem is NP-complete [26], whilst L^*-style algorithms typically construct minimal automata. Thus, to overcome this NP-hard lower bound, we must consider algorithms that may construct automata that are not minimal.

Our framework. We study the problem of learning deterministic ω-automata. The main body of the paper is devoted to DBA, but we also briefly discuss how our technique can be extended to other types of deterministic automata.

In contrast to previous approaches, we assume that the *teacher* has some limited knowledge regarding the automaton recognising the ω-regular language rather than the language alone. We introduce a third type of queries, called *loop-index* queries. To understand this type of queries observe that a computation of a deterministic automaton on an ultimately-periodic word is also ultimately-periodic, i.e., after some finite number of steps it reaches an ultimate cycle. The *loop index* of an ultimately periodic word is the number of transitions after which this ultimate cycle is reached. The *loop-index query* asks what is the loop index of a given ultimately-periodic word.

Feasibility and advantages of using loop-index queries. In contrast to the membership and equivalence queries, the loop-index queries are sensitive to the automaton structure; swapping the automaton with an equivalent one can change the answers to loop-index queries. Still, to answer these queries we need only modest information about the structure of $\mathcal{A}$. We discuss two possible scenarios where the teacher is able to answer such questions.

Black-box model. Assume that we have a black-box program in a setting where we can check the state equivalence (i.e., by comparing snapshots of the physical memory) along a run, but it is not possible to check state equivalence among different runs because, for instance, the programs memory is fragmented differently. In this case, if the black-box program is essentially a deterministic finite automaton and we can provide a reasonable estimation on the automaton size, then we can compute the loop index of a word, because the value of the loop index can be bounded in the product of the sizes of the automaton and the ultimately periodic word.

[1] University of Wrocław, Poland

Having the black-box model, we can answer the membership and loopindex queries, but not the equivalence queries. So we keep the black-box model running, but in parallel we run the automaton learned by our technique. If at some point, they produce different results, we use it as a counterexample and learn from experience to obtain a new automaton which is a better approximation of the target one. Our technique (Lemma 11 in particular) guarantees that these approximations converge, i.e., after a finite number of such iterations (at most cubic in size of the target automaton) we obtain the target automaton.

Learning automata given as black-box has a big advantage for automata given implicitly, which are too big to construct explicitly [20]. Many implicit representations (through programs or propositional formulas) allow for checking states for equality, which is sufficient for computing the loop index.

Human teacher model. We assume that the teacher is a human who has a general understanding of how the automaton should look like. Answering the loop index queries requires additional effort, but the possible benefit is that the resulting automaton may be easier to understand by the teacher as it fits better their intuition.

In particular, ω-automata can be used as a specification formalism for temporal properties [25]. In this context, learning of ω-automata facilitates the construction of temporal specifications [20]. Our framework can be used to learn specifications expressible with DBA. Still, the teacher specifying a temporal property needs to understand well what is being specified. In particular, the teacher is likely to evaluate the property over finite prefixes and partition the infinite words into parts with equivalent impact on the property. This, in turn, can be used to find a cycle and the length of the prefix leading to this cycle is the loop index.

On the technical side, adding the loop index queries to the framework makes the NP-completeness of the minimisation problem [26] no longer applicable, since we only require to construct a minimal automaton consistent with the loop index queries, which may not be minimal in general.

Results. Our main contribution is a polynomial-time algorithm for learning DBA. The algorithm asks three types of queries as discussed above and constructs a DBA of size not greater than the target automaton. Then, we adapt the learning algorithm to deterministic weighted automata with LIMINF and LIMSUP value functions. Weighted automata return values, and hence the answer to a membership query wv^ω is the value that the target automaton returns on wv^ω. We argue that this can be seen as learning parity automata. We also briefly discuss that the algorithm can be extended to deterministic Muller automata, but the complexity can suffer as the acceptance conditions constructed in the algorithm may be of exponential size.

Paper organization. We discuss the related work right after preliminaries, which allows us to give more details on how results fit into the current state of the research. Then, we formally define our framework in Section 4, develop the necessary theory in Section 5 and present the learning algorithm in Section 6. The remaining sections are devoted to discussion on possible extensions of the algorithm and discussion of the future work.

2 Preliminaries

Automata and runs. Given a finite alphabet Σ of letters, a *word* w is a finite or infinite sequence of letters. A word is *ultimately periodic* if it is of the form wv^ω, i.e., some prefix followed by an infinite repetition of a word. We denote the set of all finite words over Σ by Σ^*, and the set of all infinite words over Σ by Σ^ω. For a word w, we define $w[i]$ as the i-th letter of w, and we define $w[i,j]$ as the subword $w[i]w[i+1]\ldots w[j]$ of w. We use the same notation for vectors and sequences; we assume that sequences start with 0 index.

An *ω-generator* is a pair $(w,v) \in \Sigma^* \times (\Sigma^* \setminus \{\epsilon\})$ such that there is no v' shorter than v such that $v'^\omega = v^\omega$. An ω-generator (w,v) is a succinct representation of an ultimately-periodic word wv^ω; clearly, every such word has infinitely many possible ω-generators. Notice that we do not require here w to be minimal.

A *deterministic Büchi automaton* (DBA) is a tuple $(\Sigma, Q, q_0, F, \delta)$ consisting of the alphabet Σ, a finite set of states Q, the initial state $q_0 \in Q$, a set of accepting states F, and a transition function $\delta\colon Q \times \Sigma \to Q$.

We extend δ to $\hat{\delta}\colon Q \times \Sigma^* \to Q$ inductively: for each q, we set $\hat{\delta}(q,\epsilon) = q$, and for all $w \in \Sigma^*, a \in \Sigma$, we set $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w),a)$. The *run* π of a DBA $\mathcal{A}$ on a word w is the sequence of states $\hat{\delta}(q_0,\epsilon)\hat{\delta}(q_0,w[1])\hat{\delta}(q_0,w[1,2])\ldots$. A run is accepting if some accepting state occurs infinitely often in that run. By $\mathcal{A}(w)$ we denote the Boolean value true if the run of $\mathcal{A}$ on w is accepting.

The size of a DBA $\mathcal{A}$, denoted by $|\mathcal{A}|$, is its number of states.

The loop index. A run of a deterministic automaton on an ultimately periodic word is an ultimately periodic sequence. We define the *loop index* of a DBA $\mathcal{A}$ on an ultimately periodic word wv^ω, denoted as $\circlearrowleft^{\mathcal{A}} (w,v)$, as follows. Let π be the run of $\mathcal{A}$ on wv^ω, then $\circlearrowleft^{\mathcal{A}} (w,v)$ is the smallest number j such that for some $c > 0$ and all $i > j$ we have $\pi[i] = \pi[i+c]$ and $wv^\omega[i+1] = wv^\omega[i+1+c]$.

Example 1. *Consider the DBA A_{ex} depicted in Figure 1 and the words $w = ab$ and $v = b$. The run of $\mathcal{A}$ on wv^ω is $q_0q_0q_1q_2^\omega$ and so $\circlearrowleft^{\mathcal{A}} (w,v) = 3$. The run of $\mathcal{A}$ on ww^ω is $q_0(q_0q_1)^\omega$ and so $\circlearrowleft^{\mathcal{A}}$ $(w,w) = 1$. The run of $\mathcal{A}$ on vv^ω is $q_0q_1q_2^\omega$ and so $\circlearrowleft^{\mathcal{A}} (v,v) = 2$.*

3 Related work and technical motivations

Before we present our framework in Section 4, we discuss the elusiveness of learning ω-regular languages and other approaches to similar problems.

PAC-learning of DFA is believed to be impossible. The hardness of PAC-learning of DFA has been shown under cryptographic assumptions [18] and under average-case SAT assumptions [15]. This shifted the focus of automata learning from the passive setting (PAC-learning) to the active setting. In the active setting, Angluin showed the L^* algorithm, which learns minimal DFA in polynomial time using membership and equivalence queries [1]. The L^* algorithm has been adapted to learn NFA [11], alternating automata [3, 10], nominal automata over infinite alphabets [24], weighted automata over words [9] and trees [17, 21].

Extending the L^* algorithm to infinite-word automata has been a long-standing open problem with motivations from automatic verification [20]. In the DFA case, the L^* algorithm relies on the *right congruence relation* $\sim_L$ for a language $\mathcal{L}$ defined as follows: for $u, v \in \Sigma^*$ we have $u \sim_L v$ if and only if for all words $w \in \Sigma^*$ we have $uw \in \mathcal{L} \leftrightarrow vw \in \mathcal{L}$. Myhill-Nerode theorem implies that the minimal DFA recognising $\mathcal{L}$ is isomorphic to the automaton, whose states are equivalence classes of $\sim_L$ and the transitions are defined based on $\sim_L$.

This approach fails in the infinite-word case. Consider the DBA $\mathcal{A}_{ex}$ over $\Sigma = \{a,b\}$ depicted in Figure 1. This automaton recognizes the language $\mathcal{L}_{ex}$ of words containing infinitely often bb. Observe that the right congruence relation for $\mathcal{L}_{ex}$ consists of a single equivalence class. Indeed, $\mathcal{L}_{ex}$ is *prefix-independent*, i.e., the membership to this language does not depend on finite prefixes. Formally, for all $u, v \in \Sigma^*$ and $w \in \Sigma^\omega$ we have $uw \in \mathcal{L}_{ex} \leftrightarrow vw \in \mathcal{L}_{ex}$. Indeed, either w contains bb infinitely often and both uw and vw belong to $\mathcal{L}_{ex}$ or w contains only finitely many bb and so do the words vw and uw. However, single-state DBA can recognise only the language of all words or the empty language. Furthermore, it is not difficult to see that no DBA of 2 states recognises $\mathcal{L}_{ex}$. In summary, the right congruence relation has a single equivalence class, but a minimal-size DBA recognising $\mathcal{L}_{ex}$ has 3 states.

The above problem could be addressed by considering a different definition of the right congruence relation. However, any relation defined based on recognised language fails to characterise infinite-word automata uniquely because there are ω-languages recognised by a DBA such that a minimal-size DBA recognising it is not unique. There are various sources for this ambiguity. Again, consider the DBA $\mathcal{A}_{ex}$ (see Figure 1) and its language $\mathcal{L}_{ex}$. The automaton $\mathcal{A}_{ex}$ is strongly connected and its language is prefix-independent, therefore we can pick any state of $\mathcal{A}_{ex}$ to be its initial state and it will not affect the recognised language. Second, we can change self-loop in q_2 over b to a transition (q_2, b, q_0) or (q_2, b, q_1) and still the altered automata would recognise the language of words with infinitely many bb.

Despite the problems with learning DBA discussed above, it could have been still possible to construct an L^*-style learning algorithm for DBA, which returns some minimal-size DBA recognising the target language. Observe that such an algorithm could be used to minimise DBA. Having a DBA $\mathcal{A}$, we can answer membership and equivalence queries for the language of $\mathcal{A}$ in polynomial time in $|\mathcal{A}|$. Therefore, we could run the learning algorithm, compute answers to queries based on $\mathcal{A}$, and return the learned automaton.

However, it has been shown that deciding, given k and a DBA $\mathcal{A}$, whether $\mathcal{A}$ admits a language-equivalent DBA $\mathcal{A}'$ of at most k states, is NP-complete [26]. It follows that there is no polynomial-time learning algorithm for DBA that returns a minimal-size DBA (unless P=NP).

There have been considered three approaches to bypass the above limitations.

- ω-regular languages are equivalent if they agree on all ultimately-periodic words. Based on this observation, we define, given an ω-regular language L, the language $L_\$ = \{u\$v \mid uv^\omega \in L\}$. The language $L_\$$ is regular and can be learned with the L^*-algorithm and the resulting DFA can be transformed into a non-deterministic Büchi automaton [12]. However, the size of the minimal DFA recognising $L_\$$ can be exponential in the size of a DBA recognising L.
- Another approach is to consider restricted classes of languages. It has been shown that ω-regular languages that can be recognized with a DBA and with a deterministic co-Büchi automaton can be learned in polynomial time [12].
- It has been shown that the full class of ω-regular languages can be learned in polynomial time, if the representing formalism is Families of DFA (FDFA) [4]. The FDFA considered can be transformed to deterministic parity automata. The translation, however, involves exponential blow-up [2].

Our results in this context

In the automata learning frameworks from the literature (discussed above), the queries involve only the *language* of the automaton. We present a new approach, in which besides membership and equivalence queries, we consider loop-index queries, which depend on the particular automaton and its structure. This approach enables us to overcome the limitations discussed in this section. We show that DBA are learnable in polynomial time in our framework, and the DBA we learn is always at most as big as the target automaton. It is beneficial to learn DBA rather than $L_\$$ or FDFA, because DBA is a standard formalism, which can be fed into state-of-the-art verification tools such as PRISM [19]. We focus on learning ω-languages represented with DBA, but we also discuss how to extend our approach to learning deterministic parity automata that recognise all ω-regular languages.

Other related work

Recently, there has been a renewed interest in learning weighted automata over finite words [8, 21, 6, 5, 7]. These results apply to weighted automata over fields [16]. In the infinite-word case, it has been shown that weighted automata over the limit-average value function can be almost-exactly learned in polynomial time [22]. Almost exactly here refers to the approximation notion based on probability; the learning algorithm from [22] returns an automaton, which agrees with the target automaton on the set of words of probability 1. In this work, we require the learned automaton to agree with the target automaton on all words. Furthermore, we consider infinite-word weighted automata over the limit infimum and limit supremum value functions [13].

4 Framework

We consider the active learning setting, in which the learning algorithm asks queries to an oracle called *the teacher*. In our setting we learn DBA and the teacher answers queries about the target DBA $\mathcal{T}$, which is concealed from the learning algorithm. There are three types of queries:

- *membership queries*: given an ω-generator (w, v), the teacher returns whether $\mathcal{T}$ accepts wv^ω,
- *equivalence queries*: given an automaton $\mathcal{A}$, if $\mathcal{A}$ is language-equivalent to $\mathcal{T}$, the teacher returns YES, otherwise the teacher returns an ω-generator(w, v) such that wv^ω distinguishes $\mathcal{A}$ and $\mathcal{T}$, i.e., exactly one of $\mathcal{A}$ and $\mathcal{T}$ accepts wv^ω, and
- *loop-index queries*: given an ω-generator (w, v), the teacher returns the loop index of $\mathcal{T}$ on wv^ω

We define the learning problem as follows:

Problem 2 (Learning problem)**.** *Given a teacher for a (hidden) target DBA $\mathcal{T}$, construct a DBA $\mathcal{A}$, which is language equivalent to $\mathcal{T}$ and has at most as many states as $\mathcal{T}$.*

Remark. In our learning framework, we allow the learned automaton to be different from the target automaton. Consider $\Sigma = \{a\}$ and a DBA $\mathcal{A}$, which forms a cycle (over transitions labeled a) and all states are accepting. Then, the target automaton is indistinguishable with queries from a single-state automaton (with the state being accepting).

The algorithm presented in Section 6 returns an automaton, which is language-equivalent to the target automaton, and has the same loop-index on all the words that appear in the communication between the algorithm and the teacher.

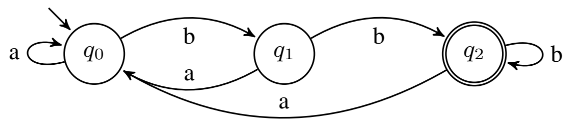

Figure 1. The automaton $\mathcal{A}_{ex}$ used in our examples.

5 Theory

In this section, we develop the theoretical underpinnings for the learning algorithm. For a given target DBA $\mathcal{T}$ and a set of ω-generators C, we show how to define an automaton $\mathcal{A}_C^{\mathcal{T}}$ based on C. Then, if such an automaton is not equivalent to $\mathcal{T}$, we show how to use a counterexample to extend C. We repeat the process and show that after a polynomial number of steps we obtain an automaton equivalent to $\mathcal{T}$.

Let us start with the following fact that is an important factor in our construction.

Fact 3. *Let $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ be a DBA. Consider $F' \subseteq Q$ that consists of all states q such that for all $wv^\omega \notin \mathcal{L}(A)$, q appears only finitely many times in the run of $\mathcal{T}$ on wv^ω. Then $\mathcal{L}(\mathcal{T}) = \mathcal{L}((\Sigma, Q, q_0, F', \delta))$.*

To understand the need for this fact, consider three automata over the alphabet a, b with states q_0, q_1, q_2 and the transitions function δ such that $\delta(q_0, a) = q_0$, $\delta(q_1, a) = q_2$, $\delta(q_2, a) = q_0$ and $\delta(q_i, b) = q_{i+1 \mod 3}$ for all i. The accepting states of the three automata are $\{q_1\}$, $\{q_2\}$ and $\{q_1, q_2\}$. Then all the automata represent the same language, the language of words with infinitely many b's. The above fact states that, while there may be many sets of accepting states that define the same language, one can always consider the maximal set, which is defined based on cycles on rejected words. The proof of this fact is straightforward and thus omitted.

5.1 Automata based on ω-generators

In this section we discuss how to define a DBA $\mathcal{A}_C^{\mathcal{T}}$ based on a set of ω-generators C and the values of the loop and membership queries regarding C on the automaton $\mathcal{T}$. Note that for the second component it suffices to have an oracle that answers to the loop and membership queries for $\mathcal{T}$.

Let us fix an alphabet Σ for the rest of the section. Let $\circledast^{\mathcal{A}}(w, r, v) = \max(0, \circlearrowleft^{\mathcal{A}}(wr, v) - |w|)$ be the function which says, after reading w, how many further letters are needed to reach the cycle on wrv^ω. The maximum function is to avoid nuances of cycles starting prematurely.

For a set of ω-generators C and a DBA $\mathcal{T}$, we define the relation $\sim_C^{\mathcal{T}} \subseteq \Sigma^* \times \Sigma^*$ such that $w_1 \sim_C^{\mathcal{T}} w_2$ if for all $(r, c) \in C$

- $\mathcal{T}(w_1 r c^\omega) = \mathcal{T}(w_2 r c^\omega)$, and
- $\circledast^{\mathcal{T}}(w_1, r, c) = \circledast^{\mathcal{T}}(w_2, r, c)$.

States of $\mathcal{A}_C^{\mathcal{T}}$. For a set of ω-generators C, we define the set of states $Q_C^{\mathcal{T}}$ as the set of equivalence classes of $\sim_C^{\mathcal{T}}$. We show that these states are, intuitively, unions of states of the target automaton, i.e., for each equivalence class $[W]_{\sim_C^{\mathcal{T}}}$ there is a set of states S such that $[W]_{\sim_C^{\mathcal{T}}} = \{w \mid \hat{\delta}(q_0, w) \in S\}$.

Lemma 4. *Let $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ be a DBA and C ω-generators. For all finite words w, v, if $\hat{\delta}(q_0, w) = \hat{\delta}(q_0, v)$, then $w \sim_C^{\mathcal{T}} v$.*

Proof. Consider words w, v such that $\hat{\delta}(q_0, w) = \hat{\delta}(q_0, v)$. Then, for every ω-generator (r, c) we have $\mathcal{T}(wrc^\omega) = \mathcal{T}(vrc^\omega)$ and $\circledast^{\mathcal{T}}(w, r, c) = \circledast^{\mathcal{T}}(v, r, c)$. In particular it holds for the ω-generators from C. It follows that $w \sim_C^{\mathcal{T}} v$. ☐

Transition function of $\mathcal{A}_C^{\mathcal{T}}$. For $a \in \Sigma$, we define

$$a^{-1}C = \{(u, c) \mid (au, c) \in C\} \cup \{(\epsilon, ca) \mid (\epsilon, ac) \in C\}.$$

The intuition behind $a^{-1}C$ is that equivalence classes of $\sim_C^{\mathcal{T}}$ determine equivalence classes of $\sim_{a^{-1}C}^{\mathcal{T}}$. We say that C is closed if for all $a \in \Sigma$ we have $a^{-1}C \subseteq C$. The closure of C, denoted by $cl(C)$, is the smallest closed set containing C. For every set C, its size of the closure $|cl(C)|$ is $O(|C|^2)$. Observe that $cl(C_1 \cup C_2) = cl(C_1) \cup cl(C_2)$.

We define the *may* relation $\Delta_C^{\text{may}} \subseteq Q_C^{\mathcal{T}} \times \Sigma \times Q_C^{\mathcal{T}}$ that contains all the possible transitions among the states $Q_C^{\mathcal{T}}$. That is, we say that $\Delta_C^{\text{may}}(W_1, a, W_2)$ holds if and only if for all $w_1 \in W_1, w_2 \in W_2$ we have $w_1 a \sim_{a^{-1}C}^{\mathcal{T}} w_2$.

Example 5. *Consider the automaton presented in Figure 1 and $C = \{(\epsilon, a)\}$. Then we have $Q_C^{\mathcal{T}} = \{q_0, q_1\}$, where q_1 contains all the words ending with b and q_0 all the remaining words. The relation Δ_C^{may} consists of tuples (q, a, q_0) and (q, b, q') for all $q, q' \in Q_C^{\mathcal{T}}$.*

We show that for all closed C the relation Δ_C^{may} contains a function.

Lemma 6. *For all automata $\mathcal{T}$, a closed set of ω-generators C, $W_1 \in Q_C^{\mathcal{T}}$ and $a \in \Sigma$ there is $W_2 \in Q_C^{\mathcal{T}}$ such that $\Delta_C^{\text{may}}(W_1, a, W_2)$.*

Proof. Observe that if $w_1 \sim_C^{\mathcal{T}} w_2$, then $w_1 a \sim_{a^{-1}C}^{\mathcal{T}} w_2 a$ for any closed set of ω-generators C. Consider a state $W_1 \in Q_C^{\mathcal{T}}$, a word $w \in W_1$ and a letter $a \in \Sigma$. Let W_2 be a state such that $wa \in W_2$. Then, for all $w' \sim_C^{\mathcal{T}} w$ we have $wa \sim_{a^{-1}C}^{\mathcal{T}} w'a$ (by the above observation), thus $\Delta_C^{\text{may}}(W_1, a, W_2)$. ☐

Let δ_C^{may} be any function that is a subset of Δ_C^{may}, i.e., for all $W_1 \in Q_C^{\mathcal{T}}$ and $a \in \Sigma$ we define $\delta_C^{\text{may}}(W_1, a) = W_2$ for some arbitrary chosen W_2 such that $\Delta_C^{\text{may}}(W_1, a, W_2)$. Such W_2 exists thanks to Lemma 6. In order to define $\mathcal{A}_C^{\mathcal{T}}$ uniquely, we assume a mapping from closed sets C to δ_C^{may} and fix one of such functions. The particular choice is irrelevant, but it needs to be consistent.

Accepting states of $\mathcal{A}_C^{\mathcal{T}}$. To define accepting states, we recall Fact 3 stating that we can focus on automata where non-accepting states are those that appear on some cycle in a run on a rejected ultimately periodic word. Thus, we define $F_C^{\mathcal{T}} \subseteq Q_C^{\mathcal{T}}$ to be the set of classes of states that are not known to be on a negative cycle, i.e., $W \in F_C^{\mathcal{T}}$ if and only if for all $w \in W$ and all $(\epsilon, c) \in C$, if $\circledast^{\mathcal{T}}(w, \epsilon, c) = 0$, then $\mathcal{T}$ accepts wc^ω.

Example 7. *In the setting of Example 5, the state q_0 will be rejecting and q_1 will be accepting.*

Finally, we define the automaton $\mathcal{A}_C^{\mathcal{T}}$ as the tuple $(\Sigma, Q_C^{\mathcal{T}}, [\epsilon]_{\sim_C^{\mathcal{T}}}, F_C^{\mathcal{T}}, \delta_C^{\text{may}})$. We will show that C can be constructed in a way to make $\mathcal{A}_C^{\mathcal{T}}$ equivalent to the target automaton.

5.2 Improving the automata

We now show how to extend C based on a counterexample for $\mathcal{A}_C^{\mathcal{T}}$ in order to obtain a better approximation of the target automaton. First, we show that extending C either increases the number of states of $\mathcal{A}_C^{\mathcal{T}}$ or does not increase $\Delta_{C'}^{\text{may}}$ and $F_{C'}^{\mathcal{T}}$.

Lemma 8 (Weak monotonicity). *Consider $C \subseteq C'$. Then, either $\mathcal{A}_{C'}^{\mathcal{T}}$ has more states than $\mathcal{A}_C^{\mathcal{T}}$, or both automata share the set of states and we have $\Delta_{C'}^{\text{may}} \subseteq \Delta_C^{\text{may}}$ and $F_{C'}^{\mathcal{T}} \subseteq F_C^{\mathcal{T}}$.*

Proof. Observe that the definition of $\sim_{C'}^{\mathcal{T}}$ involves the universal quantification over C and hence it is monotonic in C, i.e., we have $\sim_{C'}^{\mathcal{T}} \subseteq \sim_C^{\mathcal{T}}$. Thus, $|Q_{C'}^{\mathcal{T}}| > |Q_C^{\mathcal{T}}|$ or $Q_{C'}^{\mathcal{T}} = Q_C^{\mathcal{T}}$. Assuming that $Q_{C'}^{\mathcal{T}} = Q_C^{\mathcal{T}}$, the definitions of Δ_C^{may} and $F_C^{\mathcal{T}}$ involve universal quantification over C and hence are monotonic in C. □

Now, we discuss how to extend C to ensure progress, i.e., one of the inequalities from Lemma 8 becomes strict. Consider a word wv^ω accepted by exactly one of automata $\mathcal{T}$ and $\mathcal{A}_C^{\mathcal{T}}$. We show how to extend C to C' such that the automaton $\mathcal{A}_{C'}^{\mathcal{T}}$ accepts wv^ω if and only if $\mathcal{T}$ does. It follows that $\mathcal{A}_C^{\mathcal{T}}$ and $\mathcal{A}_{C'}^{\mathcal{T}}$ differ and we get strict in Lemma 8.

First, for an ω-generator (w, v) and $i \in \mathbb{N}$. We define $\textsc{Witness}(w, v, C, i)$ as a closed set of ω-generators that ensures that in both automata $\mathcal{T}$ and $\mathcal{A}_{\textsc{Witness}(w,v,C,i)}^{\mathcal{T}}$: (*) the states reached upon $wv^\omega[1, i]$ are either both accepting of both rejecting. We achieve this by making sure that $\textsc{Witness}(w, v, C, i)$ contains C and the ω-generator (w, v), and it satisfies (**): if $(\epsilon, c) \in \textsc{Witness}(w, v, C, i)$, then $(wv^\omega[1, i], c) \in \textsc{Witness}(w, v, C, i)$. Recall that for any C', acceptance of a state in $\mathcal{A}_{C'}^{\mathcal{T}}$ is defined based on its behaviour on ω-generators of the form (ϵ, c) from C', therefore (**) implies (*).

Formally, we define $\textsc{Witness}(w, v, C, i)$ as follows:

$$C \cup cl\big(\{(wv^\omega[1, i], c) \mid (\epsilon, c) \in cl(C \cup \{(w, v)\})\}\big)$$

Note that all ω-generators of the form (ϵ, c) from $\textsc{Witness}(w, v, C, i)$ belong to $cl(C \cup \{(w, v)\})$ and hence (**) holds.

While $\textsc{Witness}(w, v, C, i)$ ensures (*), which involves the prefix $wv^\omega[1, i]$, the main goal is to extend C to C' such that the resulting automaton $\mathcal{A}_{C'}^{\mathcal{T}}$ accepts wv^ω if and only if $\mathcal{T}$ does. This follows from the above properties of $\textsc{Witness}(w, v, C, i)$ and the choice of the index i. We shall pick i to be big enough so that the states reach upon $wv^\omega[1, i]$ in both automata are in the ultimate cycle over wv^ω. For such i, we know that if $\mathcal{T}$ accepts wv^ω, all states of $\mathcal{T}$ in the ultimate cycle are accepting. Conversely, if $\mathcal{T}$ rejects wv^ω, some state of $\mathcal{T}$ in the ultimate cycle is rejecting.

The following lemma formally states the properties enforced by $\textsc{Witness}(w, v, C, i)$.

Lemma 9. *Let $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ be a DBA, C be a closed set of ω-generators, (w, v) be an ω-generator and $i \geq \circlearrowleft^{\mathcal{T}} (w, v)$. For $C' = \textsc{Witness}(w, v, C, i)$, we have*

(a) if $\hat{\delta}(q_0, wv^\omega[1, i])$ is accepting in $\mathcal{T}$, then $\delta_{C'}^{\text{may}}(q_0, wv^\omega[1, i])$ is accepting in $\mathcal{A}_{C'}^{\mathcal{T}}$, and

(b) if wv^ω is rejected by $\mathcal{T}$, then for all $j \geq i$ $\delta_{C'}^{\text{may}}(q_0, wv^\omega[1, j])$ is rejecting.

Proof. For a state W, by Pos_W we denote the set of witnesses $(r, c) \in C'$ such that for every $u \in W$ satisfying $\circledast^{\mathcal{T}} (u, r, c) \leq |r|$ the DBA $\mathcal{A}_{C'}^{\mathcal{T}}$ accepts urc^ω. Notice that W is accepting if and only if for all $(\epsilon, c) \in C'$ and all $u \in W$ such that $\circledast^{\mathcal{T}} (u, \epsilon, c) = 0$ we have $(\epsilon, c) \in Pos_W$.

Let Π be the run of $\mathcal{A}_{C'}^{\mathcal{T}}$ on wv^ω and $(\epsilon, c) \in C'$. First, we show the following property

$$(wv^\omega[1, i], c) \in Pos_{\Pi[0]} \Leftrightarrow (\epsilon, c) \in Pos_{\Pi[i]}. \qquad (\$)$$

To see (\$), observe that for all j we have $(wv^\omega[j, i], c) \in C'$ as C' is closed. Furthermore, for all $j > 0$ we have $\Delta_C^{\text{may}}(\Pi[j-1], wv^\omega[j], \Pi[j])$, i.e., $\Pi[j-1]a \sim_{a^{-1}C}^{\mathcal{T}} \Pi[j]$ for $a = wv^\omega[j]$. The definition of $\sim_{a^{-1}C}^{\mathcal{T}}$ implies that

$$(wv^\omega[j+1, i], c) \in Pos_{\Pi[j]} \Leftrightarrow (wv^\omega[j+2, i], c) \in Pos_{\Pi[j+1]}.$$

A straightforward induction leads to (\$).

To show (a), assume that q_i is the accepting state reached in $\mathcal{T}$ upon $wv^\omega[1, i]$, i.e., $q_i = \delta_{C'}^{\text{may}}(q_0, wv^\omega[1, i])$. Then, for every $(\epsilon, c) \in C'$, if $\circledast^{\mathcal{T}} (wv^\omega[1, i], \epsilon, c) = 0$, then $\mathcal{T}$ reaches upon $wv^\omega[1, i]$ a cycle over c. This cycle contains q_i, which is accepting. Therefore, $wv^\omega[1, i]c^\omega$ is accepted by $\mathcal{T}$ and hence $(wv^\omega[1, i], c) \in Pos_{\Pi[0]}$. We apply (\$) and get $(\epsilon, c) \in Pos_{\Pi[i]}$. This holds for every $(\epsilon, c) \in C'$ and hence $\Pi[i]$ is accepting.

Now we prove (b). First, for v, we say that z is a *rotation* of v if $z = xy$ and $v = yx$ for some words x, y.

Assume that wv^ω is rejected by $\mathcal{T}$. Let z be a rotation of v that *completes* $wv^\omega[1, i]$, i.e.,

$$wv^\omega[1, i]z^\omega = wv^\omega$$

Observe that $(wv^\omega[1, i], z) \in C' \setminus Pos_{\Pi[0]}$. Note that by (\$) we have $(\epsilon, z) \in C' \setminus Pos_{\Pi[i]}$. Since $i \geq \circlearrowleft^{\mathcal{T}} (w, v)$, we have $\circledast^{\mathcal{T}} (wv^\omega[1, i], \epsilon, v) = 0$. Therefore, $\Pi[i]$ is rejecting.

Now, observe that for all $j \geq i$, the state $\Pi[j]$ is rejecting as well. Indeed, the definition of $\sim_{a^{-1}C}^{\mathcal{T}}$ implies that for all j we have

$$(\epsilon, au) \in Pos_{\Pi[j]} \Leftrightarrow (\epsilon, ua) \in Pos_{\Pi[j+1]}$$

Furthermore, the closure implies that for all rotations z of v we have $(\epsilon, z) \in C'$.

It follows that for the rotation r of v that completes $wv^\omega[1, j]$, i.e.,

$$wv^\omega = wv^\omega[1, j]r^\omega$$

we have $(\epsilon, r) \in C' \setminus Pos_{\Pi[j]}$. Clearly, $\circledast^{\mathcal{T}} (wv^\omega[1, j], \epsilon, v) = 0$ and hence $\Pi[j]$ is rejecting. □

The above lemma shows that if wv^ω is rejected by the target automaton, then it is also is rejected by $\mathcal{A}_{C'}^{\mathcal{T}}$ for $C' = \textsc{Witness}(w, v, C, \circlearrowleft^{\mathcal{A}} (w, v))$. If wv^ω is accepted, then the accepting state in its ultimate cycle can be only reached after $|w| + |\mathcal{T}|$ steps; however, while learning $\mathcal{T}$, we do not know its size, and in this case we cannot estimate it as an underestimation could violate the termination property of our algorithm. To avoid such problems, we define a fixed-point like method for finding a correct value of i.

The saturate function. We define the function $saturate_C^{\mathcal{T}}$ on ω-generators such that for all (w, v) we put

$$saturate_C^{\mathcal{T}}(w, v) = \textsc{Witness}(w, v, C, i)$$

where $i > |w| + |\mathcal{A}_{C'}^{\mathcal{T}}| \cdot |v|$ is the smallest number such that the state after reading $wv^\omega[1, i]$ in $\mathcal{A}_{\textsc{Witness}(w,v,C,i)}^{\mathcal{T}}$ is accepting if and only if $\mathcal{T}$ accepts wv^ω.

Observe that the smallest i exists and is bounded by $\circlearrowleft^{\mathcal{T}}(w,v) + |\mathcal{T}| \cdot |v|$. If the run is rejecting, then $i = |w| + |\mathcal{A}^{\mathcal{T}}_{C'}| \cdot |v| + 1$ suffices. Otherwise, the cycle in the run of $\mathcal{T}$ on wv^{ω} starts at $\circlearrowleft^{\mathcal{T}}(w,v)$. The length of the cycle is bounded by $|\mathcal{T}| \cdot |v|$. Since this run is accepting, we know that the cycle contains an accepting state on some position i satisfying

$$|w| + |\mathcal{A}^{\mathcal{T}}_{C'}| \cdot |v| < i \leq |w| + 2 \cdot |\mathcal{T}| \cdot |v|. \qquad (\#)$$

The second inequality follows from the fact that $|\mathcal{A}^{\mathcal{T}}_{C'}| \leq |\mathcal{T}|$ and $\circlearrowleft^{\mathcal{T}}(w,v) \leq |w| + |\mathcal{T}| \cdot |v|$. Lemma 9 implies that the equivalence class of $wv^{\omega}[1,i]$ is accepting in $\mathcal{A}^{\mathcal{T}}_{C'}$. We conclude with the following:

Lemma 10. *Let $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ be a DBA and C be a set of ω-generators and (w,v) be an ω-generator. For $C' = saturate^{\mathcal{T}}_{C}(w,v)$, we have $\mathcal{T}(wv^{\omega}) = \mathcal{A}^{\mathcal{T}}_{C'}(wv^{\omega})$.*

Proof. Assume that $\mathcal{T}$ accepts wv^{ω}. Let i be the number from the definition of $saturate^{\mathcal{T}}_{C}(w,v)$. The equivalence class W_i is accepting. Since $i > |w| + |\mathcal{A}^{\mathcal{T}}_{C'}| \cdot |v|$, the position i belongs to the cycle of $\mathcal{A}^{\mathcal{T}}_{C'}$ on wv^{ω} and hence this cycle contains an accepting state W_i. Therefore, $\mathcal{A}^{\mathcal{T}}_{C'}$ accepts wv^{ω}.

Assume that $\mathcal{T}$ rejects wv^{ω}. Then, Lemma 9 (b) states that $\mathcal{A}^{\mathcal{T}}_{C'}$ rejects wv^{ω}. □

Lemma 10 implies that if wv^{ω} is a word accepted by exactly one of the automata $\mathcal{T}$, $\mathcal{A}^{\mathcal{T}}_{C}$ and $C' = saturate^{\mathcal{T}}_{C}(w,v)$, then then word wv^{ω} distinguishes $\mathcal{A}^{\mathcal{T}}_{C}$ and $\mathcal{A}^{\mathcal{T}}_{C'}$. This, together with Lemma 8 gives us the following lemma.

Lemma 11 (Strong monotonicity). *If wv^{ω} is a word accepted by exactly one of the automata $\mathcal{T}$, $\mathcal{A}^{\mathcal{T}}_{C}$ and $C' = saturate^{\mathcal{T}}_{C}(w,v)$ is such that $Q^{\mathcal{T}}_{C} = Q^{\mathcal{T}}_{C'}$ and $F^{\mathcal{T}}_{C'} = F^{\mathcal{T}}_{C}$, then $\delta^{may}_{C} \not\subseteq \Delta^{may}_{C'}$.*

6 Algorithm

We now present the learning algorithm. We assume that the alphabet Σ is fixed and known to the algorithm. The main procedure is presented as Algorithm 1. It assumes a teacher T that provides three functions, realising the three kinds of queries:

- EQUIVALENCE(A) that verifies whether a given automaton is equivalent to the target one, and returns true if it is or an ω-generator of a word that distinguishes the target automaton and A otherwise,
- LOOPINDEX(w,v) that returns the loop index for wv$^{\omega}$, and
- VALUE(w,v) that returns whether the target automaton accepts wv$^{\omega}$.

Algorithm 1 The algorithm learning a DBA.

```
1: procedure LEARN(T)
2:     C := ∅
3:     A := GENERATE_AUTOMATON(C, T)
4:     while T.EQUIVALENCE(A) ≠ true do
5:         (w,v) := T.EQUIVALENCE(A)
6:         C := SATURATE(w, v, C, T)
7:         A := GENERATE_AUTOMATON(C, T)
8:     return A
```

The procedure LEARN(T) starts with C being empty and then, in each loop iteration, it extends C based on a provided counterexample. In Section 6.3, we argue that the number of loop iterations is at most cubic in the number of states of the target automaton. But first, we discuss the procedures SATURATE(w, v, C, T) and GENERATE_AUTOMATON(C, T).

6.1 Automata generation

The general scheme of the procedure GENERATE_AUTOMATON(C, T) is presented as Algorithm 2.

Essentially, the states of the automaton $\mathcal{A}$ constructed by the algorithm are functions $q\colon C \to \{true, false\} \times \mathbb{N}$. For each state, we keep a *selector* w_q such that for every $(r,c) \in C$ we have $q(r,c) = (\mathcal{T}(w_q rc^{\omega}), \circledast^{\mathcal{T}}(w_q, r, c))$.

The number of possible states is infinite, but Lemma 4 guarantees that the number of such functions realised in any automaton $\mathcal{A}^{\mathcal{T}}_{C}$ is bounded by the number of states of $\mathcal{T}$. To benefit from this, the procedure GENERATE_AUTOMATON(C, T) performs a BFS search on the state space; it starts from the single-state (incomplete) automaton with the empty transition function δ, and then adds transitions (function ADD_SUCCESSORS(A)) in the following manner: for any state q and letter a such that the value of $\delta(q,a)$ is not set, take the selector w_q and compute q_a such that for all $(r,c) \in C$ we have $q_a(r,c)$ =(T.VALUE($warc^{\omega}$), $\circledast^{\mathcal{T}}(wa,r,c)$). If q_a is in the set of states, then $\delta(q,a) := q_a$. Otherwise, add q_a to the set of states, set $\delta(q,a) := q_a$ and $w_{q_a} := wa$.

Depending on the choice of successors above, we may add successors in a different order and end up with different functions δ. We show, however, that $\delta \subseteq \Delta^{\text{may}}_{C}$. Indeed, for any q,a,q', if $(q,a,q') \in \delta$, then $w_{q'} \sim_C w_q a$, so in particular $w_{q'} \sim_{a^{-1}C} w_q a$, thus $\Delta^{\text{may}}_{C}(q,a,q')$.

The last step is to compute the accepting states – this amounts to marking a state q as an accepting state if for all $(\epsilon, c) \in C$ the first element of $q(\epsilon, c)$ is $true$.

This procedure works in polynomial time: it takes at most $|\mathcal{T}| \cdot |\Sigma|$ loop iterations, as the size of the generated automaton is at most the size of the target automaton $\mathcal{T}$. Each loop iteration asks at most $|C|$ queries and processes them in polynomial time.

Algorithm 2 The automaton generating procedure.

```
1: procedure GENERATE_AUTOMATON(C, T)
2:     A := $(\Sigma, \{q_0\}, q_0, \{\}, \{\})$
3:     while A ≠ ADD_SUCCESSORS(A) do
4:         A := ADD_SUCCESSORS(A)
5:     A := COMPUTE_ACCEPTING_STATES(A)
6:     return A
```

6.2 Saturation

The procedure SATURATE(w, v, C, T) is a straightforward implementation of $saturate^{\mathcal{T}}_{C}(wv^{\omega})$. It is crucial that $saturate^{\mathcal{T}}_{C}$ depends on the target automaton $\mathcal{T}$ only to the extent provided by the teacher T. The procedure uses auxiliary functions ACCEPTING(A, q) checking whether q is an accepting state in A and EVAL(A, w) which returns the state of A after reading w.

6.3 Complexity and correctness

The procedure LEARN(T) works in time polynomial in size of the target automaton $\mathcal{T}$. To see this, observe that by Lemmas 8 and 11 each step of the main loop of LEARN(T) increases C so one of two things happens:

Algorithm 3 The saturation procedure.

```
 1: procedure SATURATE(x, y, C, T)
 2:     li := T.LOOPINDEX(x,y)
 3:     (w, v) := $(xy^\omega)[1,li]$, $(xy^\omega)[li+1,li+LENGTH(v)]$
 4:     i := LENGTH(w)+LENGTH(v)+1
 5:     repeat
 6:         C' := C ∪ $cl(\{(wv^\omega[1,i], c) \mid (\epsilon, c) \in cl(C \cup \{(w,v)\})\})$
 7:         Ac := GENERATE_AUTOMATON(C)
 8:         class_acc := ACCEPTING(Ac, EVAL(Ac,$(wv)^\omega[1,i]$))
 9:         i := i+1
10:     until class_acc = T.VALUE(w,v)
11:     return C'
```

1. the number of states of $\mathcal{A}_C^{\mathcal{T}}$ increases, or
2. the set of states remains the same, the sizes of the set of accepting states and Δ_C^{may} do not increase and one of them decreases.

By Lemma 4, the number of states of $\mathcal{A}_C^{\mathcal{T}}$ is bounded by the number of states of $\mathcal{T}$, so 1 happens less than $|\mathcal{T}|$ times. For a fixed set of states Q, the set F has at most $|Q|$ elements and Δ_C^{may} has at most $|Q|^2|\Sigma|$ elements, so they can decrease that many times. Therefore, the number of loop iterations is at most cubic in $|\mathcal{T}|$.

It is not hard to check that GENERATE_AUTOMATON(C, T) works in time polynomial in the target automaton and C, and for SATURATE(w, v, C, T), polynomial time follows from the estimation (#) in Section 5. Thus, the whole procedure works in polynomial time.

The correctness of the learning algorithm follows from the fact that it only terminates when there are no more counterexamples, i.e., when it finds an automaton equivalent to the target one.

Theorem 12. *Active learning DBA with teacher answering equivalence queries, membership queries and loop-index queries can be done in time polynomial in the size of the target automaton and the size of teachers' responses.*

7 Extensions

The algorithm for learning DBA can be straightforwardly adapted to learn deterministic co-Büchi automata. It stems from the fact that a DBA can be complemented by swapping accepting and rejecting states and changing the Büchi acceptance condition into co-Büchi. We discuss further extension of our technique to different types of deterministic ω-automata.

7.1 Deterministic limit infimum and limit supremum automata.

A deterministic LIMSUP-automaton $\mathcal{A}$ is similar to DBA, except that the set of accepting states F is now replaced by a cost function $\gamma\colon Q \to \mathbb{Z}$ which assigns integers, called *weights*, to states. Runs of LIMSUP-automata are defined in the same way as for DBA. The value of a word w assigned by the LIMSUP-automaton $\mathcal{A}$, denoted by $\mathcal{A}(w)$, is defined as the limit supremum of weights of states along the run of $\mathcal{A}$ on w, i.e., the maximal weight that appears infinitely often. We say that LIMSUP-automata $\mathcal{A}_1, \mathcal{A}_2$ are *equivalent* if for all words w we have $\mathcal{A}_1(w) = \mathcal{A}_2(w)$. The definition of $\mathcal{A}(w)$ for LIMINF-automata is symmetric; in the rest of this section we focus on LIMSUP-automata.

The learning framework for LIMSUP-automata is a straightforward extension of the framework for DBA: the membership queries now return the value of a given word rather than just whether the word is accepted, and the other queries remain virtually the same. The learning problem is defined analogously to the DBA case.

Observe that LIMSUP-automata satisfy the following counterpart of Fact 3.

Fact 13. *Let $\mathcal{T} = (\Sigma, Q, q_0, \gamma, \delta)$ be a deterministic* LIMSUP-*automaton. Consider $\gamma'\colon Q \to \mathbb{Z}$ such that $\gamma'(q)$ is the minimum of values of all the words whose runs visit q infinitely often. Then $\mathcal{T}$ and $(\Sigma, Q, q_0, \gamma', \delta)$ are equivalent.*

Proof. Let $\mathcal{T}' = (\Sigma, Q, q_0, \gamma', \delta)$. Clearly, for any q we have $\gamma'(q) \geq \gamma(q)$, thus for any word w, $\mathcal{T}'(w) \geq \mathcal{T}(w)$. On the other hand, $\mathcal{T}'(w) \leq \mathcal{T}(w)$ because the value of each state that $\mathcal{T}'$ visits infinitely often while reading w is bounded by $\mathcal{T}(w)$. Thus, for any w, $\mathcal{T}'(w) = \mathcal{T}(w)$, meaning that $\mathcal{T}'$ and $\mathcal{T}$ are equivalent. □

We briefly discuss how to adapt the learning algorithm for DBA to learn LIMSUP-automata.

Theorem 14. *Active learning* LIMSUP-*automata and* LIMINF-*automata with teacher answering equivalence queries, membership queries and loop-index queries can be done in time polynomial in the size of the target automaton and the size of teachers' responses.*

The proof is very similar to the proof of Theorem 12. The way the automaton $\mathcal{A}_C^{\mathcal{T}}$ is constructed and improved is almost the same as before, except we now include the weights. The algorithm chances very little as well.

The automaton $\mathcal{A}_C^{\mathcal{T}}$. First, for a set of ω-generators C and a LIMSUP-automaton $\mathcal{A}$, we define the relation $\sim_C^{\mathcal{T}}$ in the same way as for DBA. Note, however, that $\mathcal{A}(wv^\omega)$ denotes the value that the LIMSUP-automaton returns on wv^ω. The transition relation is defined in the same way, as in the DBA case.

Finally, we define the cost function based on Fact 13. Intuitively, $\gamma_C^{\mathcal{A}}(W) = x$ if we cannot show that the value of $\gamma_C^{\mathcal{A}}(W)$ should be smaller. Formally, for every $W \in Q_C^{\mathcal{T}}$, we define $\gamma_C^{\mathcal{A}}(W)$ as the minimum $\mathcal{A}(wc^\omega)$ over all $w \in W$ and all $(\epsilon, c) \in C$, where the minimum is ∞ if there are no $(\epsilon, c) \in C$.

Automata improvement. Counterexample-guided automaton improvement works similarly to the DBA case, except that now the $saturate_C^{\mathcal{T}}$ procedure ensures that the new automaton while processing a counterexample reaches a state in the ultimate cycle of its run with weight equal to the value of the counterexample. It is also ensured that there is no higher weight in the ultimate cycle.

Algorithm. The algorithm from Section 6 can be adapted to the LIMSUP-automata case simply by adjusting the functions operating on accepting states to work with weights. In particular, in GENERATE_AUTOMATON(C, T), we need to replace the function computing the accepting states with a function that computes for each state q the minimum of values in q of all the loops $(\epsilon, c) \in C$, following the intuition provided by Fact 13.

Furthermore, in SATURATE(x, y, C, T), we replace ACCEPTING(Ac, EVAL(Ac,$(wv)^\omega[1, i]$)) with WEIGHT(Ac, EVAL(Ac,$(wv)^\omega[1, i]$)), which returns the weight of the states reached upon $(wv)^\omega[1, i]$. It can be shown that this weight reaches T.VALUE(w,v) in a desirable number of iterations.

The proof of polynomial complexity and acceptance remains the same and thus is omitted.

7.2 Deterministic parity automata

A deterministic parity automaton (DPA) $\mathcal{A}$ is a tuple $(\Sigma, Q, q_0, \delta, \ell)$, where Σ, Q, q_0, δ are as in a DBA, and γ is a function labelling states with natural *priorities*, i.e., $\ell\colon Q \to \mathbb{N}$. A run π of DPA is defined in the same way as for DBA; the run π is accepting if the maximal priority occurring infinitely often is even, i.e., limit supremum of priorities along π is even.

Observe that DPA and deterministic LIMSUP-automata are closely related. In fact, we can learn DPA if we consider these automata as deterministic LIMSUP-automata. That is, assuming that in the membership queries the teacher returns the maximal priority that occurs infinitely often, we can use the learning algorithm for deterministic LIMSUP-automata to learn DPA.

7.3 Other acceptance conditions

Our technique does not extend to Street and Rabin conditions. The reason is that there is no counterpart of Fact 3 from these automata. In the DBA case, a rejected word is a proof that all the states on its ultimate cycle are rejecting; in the Street and Rabin cases, this translates into an alternative; for example, in the Rabin case, an ultimate periodic word is rejected if for all the pairs (B, G), the ultimate cycle does not visit G or visits B. Overcoming this difficulty is an interesting future work.

For Muller automata, the acceptance condition is a family of sets of states and a run of an ultimately-periodic word is accepting if the set of states of its ultimate cycle is in this family. Interestingly, a counterpart of Fact 3 holds for Muller automata: every rejected ultimately-periodic word is a proof that the set of states of its ultimate cycle does not belong to the acceptance condition. Thus, our technique can be extended to work with Muller automata. However, this requires the acceptance condition to be represented explicitly. For a target automaton with a small acceptance condition, our technique is likely to consider during its computation an automaton $\mathcal{A}_C^{\mathcal{T}}$ whose acceptance condition is of exponential size. Therefore, the obtained (pessimistic) complexity is as good as of the naive algorithm („check all the automata, starting from the smallest"). Whether this can be avoided by considering some compressed representation remains an open question.

8 Conclusions and future work

We have shown that various types of infinite-word automata can be actively learned in polynomial time if we allow the algorithm to ask three types of queries: membership, equivalence and loop-index. Loop-index queries depend on the structure of the target automaton and have not been considered before. This new approach allowed us to bypass various obstacles, which appear in the infinite-word case.

The approach can be employed to work with limit infimum and limit supremum automata. Whether it can be extended to other types of automata is an open problem.

In future work, we plan to study active learning of other types of infinite-word weighted automata such as automata with the limit average value function.

ACKNOWLEDGEMENTS

This work was supported by the National Science Centre (NCN), Poland under grant 2017/27/B/ST6/00299.

REFERENCES

[1] Dana Angluin, 'Learning regular sets from queries and counterexamples', *Information and computation*, **75**(2), 87–106, (1987).

[2] Dana Angluin, Udi Boker, and Dana Fisman, 'Families of DFAs as acceptors of ω-regular languages', *LMCS*, **14**(1), (2018).

[3] Dana Angluin, Sarah Eisenstat, and Dana Fisman, 'Learning regular languages via alternating automata', in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, (2015).

[4] Dana Angluin and Dana Fisman, 'Learning regular omega languages', *Theor. Comput. Sci.*, **650**, 57–72, (2016).

[5] Borja Balle and Mehryar Mohri, 'Learning weighted automata', in *CAI 2015*, pp. 1–21, (2015).

[6] Borja Balle and Mehryar Mohri, 'On the rademacher complexity of weighted automata', in *ALT 2015*, pp. 179–193, (2015).

[7] Borja Balle and Mehryar Mohri, 'Generalization bounds for learning weighted automata', *Theor. Comput. Sci.*, **716**, 89–106, (2018).

[8] Borja Balle, Prakash Panangaden, and Doina Precup, 'A canonical form for weighted automata and applications to approximate minimization', in *LICS 2015*, pp. 701–712, (2015).

[9] Amos Beimel, Francesco Bergadano, Nader Bshouty, Eyal Kushilevitz, and Stefano Varricchio, 'Learning functions represented as multiplicity automata', *Journal of the ACM*, **47**, (10 1999).

[10] Sebastian Berndt, Maciej Liśkiewicz, Matthias Lutter, and Rüdiger Reischuk, 'Learning residual alternating automata', in *Thirty-First AAAI Conference on Artificial Intelligence*, (2017).

[11] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker, 'Angluin-style learning of NFA', in *IJCAI 2009*, pp. 1004–1009, (2009).

[12] Hugues Calbrix, Maurice Nivat, and Andreas Podelski, 'Ultimately periodic words of rational *w*-languages', in *MFPS 1993*, pp. 554–566, (1993).

[13] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger, 'Quantitative languages', *ACM TOCL*, **11**(4), 23, (2010).

[14] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem, *Handbook of model checking*, volume 10, Springer, 2018.

[15] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz, 'From average case complexity to improper learning complexity', in *STOC 2014*, pp. 441–448, (2014).

[16] Manfred Droste, Werner Kuich, and Heiko Vogler, *Handbook of Weighted Automata*, Springer, 1st edn., 2009.

[17] Amaury Habrard and José Oncina, 'Learning multiplicity tree automata', in *ICGI 2006*, pp. 268–280, (2006).

[18] Michael Kearns and Leslie Valiant, 'Cryptographic limitations on learning boolean formulae and finite automata', *Journal of the ACM (JACM)*, **41**(1), 67–95, (1994).

[19] M. Kwiatkowska, G. Norman, and D. Parker, 'PRISM 4.0: Verification of probabilistic real-time systems', in *CAV 2011*, volume 6806 of *LNCS*, pp. 585–591. Springer, (2011).

[20] Martin Leucker, 'Learning meets verification', in *FMCO 2006*, pp. 127–151, (2006).

[21] Ines Marusic and James Worrell, 'Complexity of equivalence and learning for multiplicity tree automata', *Journal of Machine Learning Research*, **16**, 2465–2500, (2015).

[22] Jakub Michaliszyn and Jan Otop, 'Approximate learning of limit-average automata', in *CONCUR 2019*, pp. 17:1–17:16, (2019).

[23] Ian Millington, *AI for Games*, CRC Press, 2019.

[24] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski, 'Learning nominal automata', in *POPL 2017*, pp. 613–625, (2017).

[25] Amir Pnueli and Aleksandr Zaks, 'On the merits of temporal testers', in *25 Years of Model Checking - History, Achievements, Perspectives*, pp. 172–195, (2008).

[26] Sven Schewe, 'Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete', in *FSTTCS 2010*, pp. 400–411, (2010).