Passive Learning of Deterministic Büchi Automata by Combinations of DFAs

León Bohn
RWTH Aachen University, Germany

Christof Löding
RWTH Aachen University, Germany

Abstract

We present an algorithm that constructs a deterministic Büchi automaton in polynomial time from given sets of positive and negative example words. This learner constructs multiple DFAs using a polynomial-time active learning algorithm on finite words as black box using an oracle that we implement based on the given sample of ω-words, and combines these DFAs into a single DBA. We prove that the resulting algorithm can learn a DBA for each DBA-recognizable language in the limit by providing a characteristic sample for each DBA-recognizable language. We can only guarantee completeness of our algorithm for the full class of DBAs through characteristic samples that are, in general, exponential in the size of a minimal DBA for the target language. But we show that for each fixed k these characteristic samples are of polynomial size for the class of DBAs in which each subset of pairwise language-equivalent states has size at most k.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases deterministic Büchi automata, learning from examples, learning in the limit, active learning

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.114

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *León Bohn*: Supported by DFG grant LO 1174/7-1.

1 Introduction

The problem of constructing finite automata from example words (also referred to as passive learning of automata) has been investigated since the 1970ies [8, 26, 14], see [17] for a survey. The task is to develop an algorithm that infers a finite automaton from a given sample $S = (S_+, S_-)$ such that all words from the finite set S_+ of positive examples are accepted, and all words from the finite set S_- are rejected. Such an algorithm can be viewed as a way of learning an automaton from a given set of examples, and we therefore refer to such algorithms as learners in the following.

Besides the running time of a learner, it is also of interest for which languages it can learn a finite automaton. In order to characterize learners that are robust and generalize from the sample, Gold proposed the notion of "learning in the limit" [13]. Given a class $\mathcal{C}$ of regular languages, a learner is said to learn every language in $\mathcal{C}$ in the limit, if for each language $L \in \mathcal{C}$ there is a characteristic sample S^L that is consistent with L and such that the learner returns a DFA for L for each sample that is consistent with L and contains all examples from S^L. In this case, we also say that the learner is complete for the class $\mathcal{C}$.

In [14] Gold presents a learner that constructs in polynomial time a DFA for a given sample and that can learn every regular language in the limit. Moreover, for each regular language L there is a characteristic sample of size polynomial in the minimal DFA for L. Such a learner is said to learn every regular language in the limit from polynomial data.

The key property that is used in most learning algorithms for regular languages is the characterization of regular languages by the Myhill/Nerode congruence: For a language L, two words u, v are equivalent if they cannot be distinguished by the language, that is, if for each word w, either both uw, vw are in L, or both are not in L. It is a basic result from automata theory that L is regular iff the Myhill/Nerode congruence has finitely many classes, and that these classes can be used as state set of the minimal DFA for L (see basic textbooks on automata theory, e.g. [15]). The idea for Gold's algorithm is to infer the Myhill/Nerode congruence from the sample $S = (S_+, S_-)$ based on the following idea. Two words u, v cannot be equivalent for any language that is consistent with S, if there is a word w such that $uw \in S_+$ iff $vw \in S_-$. Roughly speaking, Gold's algorithm uses such "obviously distinguishable" words as states for a DFA. If the sample does not contain enough information, it can happen that the resulting DFA is not consistent with the sample, and then the algorithm simply defaults to returning a DFA for S_+. The algorithm RPNI [21] avoids this problem by starting from a tree-shaped DFA for S_+, and then trying to merge states of this DFA in a specific order. If a merge leads to a DFA that is inconsistent with the sample, then the merge is discarded. Otherwise, the two states are merged, and the algorithm continues with this smaller DFA. This algorithm runs in polynomial time and learns every regular language in the limit from polynomial data [21]. Since then, many variations of such state merging techniques have been proposed and implemented, e.g., in the framework learnlib [16] and the library flexfringe [27]

While there is a lot of work on construction of DFAs from samples, very little is known about this problem for automata on infinite words, so called ω-automata. These have been studied since the early 1960s as a tool for solving decision problems in logic [11] (see also [24]), and are nowadays used in procedures for formal verification and synthesis of reactive systems (see, e.g., [6, 25, 19] for surveys and recent work). Syntactically, ω-automata are very similar to NFA resp. DFA (standard nondeterministic resp. deterministic finite automata on finite words), and they also share many closure and algorithmic properties. However, while the definition of the Myhill/Nerode congruence can easily be lifted to ω-languages, it does not give a characterization of the regular ω-languages. In particular, deterministic ω-automata may need several different language equivalent states in order to accept some regular ω-languages (as opposed to DFAs). As a consequence, deterministic ω-automata do not share the good properties of DFAs, e.g., minimization of deterministic Büchi automata is NP-hard [23], and also the methods for learning DFAs cannot be directly transferred to ω-automata. In fact, the current algorithms for constructing deterministic ω-automata from examples are adaptions of the algorithm of Gold [5] and of RPNI [9], and they are only known to learn ω-languages with informative right congruence (IRC) in the limit. The class IRC [4] consists of those languages that can be accepted by deterministic ω-automata without different language equivalent states, and thus can be handled by an adaption of the methods from finite words.

In this paper we propose a passive learning algorithm that is complete for the class of ω-languages that can be accepted by deterministic Büchi automata (DBA) (meaning that it can learn every language form that class in the limit). To the best of our knowledge, this is the first learning algorithm that is complete for a relevant class of ω-languages beyond languages with IRC. While DBA languages still form a strict subclass of the regular ω-languages, each regular ω-language is a finite Boolean combination of DBA languages [24], and therefore the DBA languages form an important class for understanding learning problems for ω-automata.

Our algorithm uses a learning algorithm for DFAs as sub-procedure, but interestingly an active learning algorithm. Such algorithms have to infer the DFA of a target language based on queries that are answered by an oracle. Angluin proposed an algorithm that learns the minimal DFA for a target language L based on membership and equivalence queries in polynomial time [1]. A membership query asks for a specific word if it is in the target language, and the oracle answers "yes" or "no". An equivalence query asks if a hypothesis DFA accepts the target language, and the oracle provides a word as counterexample if it does not. We make use of such an active learning algorithm as black box in order to infer a set of DFAs that are then used to build the DBA. Roughly, our algorithm works as follows for a given ω-sample $S = (S_+, S_-)$ consisting of ultimately periodic ω-words (the use of ultimately periodic words is standard in learning of ω-automata [18, 3, 5, 9]):

- Infer a right congruence $\sim$ that is consistent with S. This can be done using the same methods as for finite words.
- For each class of $\sim$ learn a DFA using an active learning algorithm as black box, answering the queries of this algorithm based on the information in S.
- Combine the DFAs into a DBA: Start in the DFA for the initial class of $\sim$. Whenever a DFA reaches an accepting state, redirect the transition into the initial state of the DFA for the current class, and make this transition accepting for the Büchi condition.[1]

This algorithm runs in polynomial time and can learn every DBA language in the limit. The characteristic sample for a DBA language L that we use for showing the learning in the limit result can be of size exponential in a smallest DBA for L. But we also show that it is polynomial if we fix the size of sets of pairwise language equivalent states in the DBAs. This generalizes known results for learning in the limit from polynomial data for languages with IRC [5, 9].

Besides the work on passive learning, there are also some papers on active learning of ω-languages from queries. In general, it seems that efficient active learning of deterministic omega-automata is more difficult than efficient passive learning. This is witnessed by the fact that a polynomial time active learner (with membership and equivalence queries) yields an efficient passive learner, and that polytime active learning IRC ω-languages is not easier than polytime active learning general omega-languages (see [9] for both results). For this reason, the currently known active learning algorithms either learn different representations for ω-languages, as for example families of DFAs in [3], or include syntactic information on the target automaton in the form of loop index queries [20].

The remainder of the paper is structured as follows. In Section 2 we introduce basic terminology and definitions used in the paper. Subsequently, in Section 3 we present our learning algorithm and prove that it is a consistent polynomial time DBA-learner. In Section 4 we prove the completeness result that our algorithm can learn every DBA language in the limit. In Section 5 we analyze the size of the characteristic samples from the completeness proofs, and we conclude in Section 6.

2 Preliminaries

For a finite alphabet Σ we denote by Σ^* and Σ^ω the set of all finite and infinite words over Σ, respectively and define $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, where ε denotes the empty word. A language is a subset of Σ^* and an ω-language a subset of Σ^ω. We sometimes simply write language instead of ω-language, the meaning should always be clear from the context. We use $\preceq$ to denote the canonical or length-lexicographic order on finite words over Σ, in which words are first ordered by length, and words of same length are ordered lexicographically for some underlying linear order of the alphabet. For a finite word u and a finite or infinite word v,

[1] We use Büchi acceptance on transitions, i.e., a run is accepting if it passes infinitely many accepting transitions.

we write $u \sqsubseteq v$ if $ux = v$ for some $x \in \Sigma^*$ or $x \in \Sigma^\omega$, respectively, and call u a *prefix* of v. We use $\mathsf{Prf}(w)$ to denote the set of all prefixes of a finite or infinite word w and extend this notation to a set X of words in the natural way, i.e. $\mathsf{Prf}(X) = \bigcup_{w \in X} \mathsf{Prf}(w)$.

We say that $(u, v) \in \Sigma^* \times \Sigma^+$ is a *representation of* $w \in \Sigma^\omega$ if $uv^\omega = w$. If an ω-word w has such a representation, we call w *ultimately periodic*, and we denote the set of all possible representations $\Sigma^* \times \Sigma^+$ with $\mathbb{UP}$. For a finite word u and a (ω)-language L, we define $u^{-1}L = \{w \mid uw \in L\}$. Note that if L is an ω-language, then $u^{-1}L$ is an ω-language as well. For a word w, we use $w[i, j]$ to denote the infix of w from position i to position j and set $w[i, j] = \varepsilon$ if $i > j$. We write $(u, v) \preceq (u', v')$ if $u \preceq u'$ or $u = u'$ and $v \preceq v'$. For a set $X \subseteq \mathbb{UP}$ of representations, we define $[\![X]\!]_\omega = \{uv^\omega \mid (u, v) \in X\}$.

A *deterministic transition system* (TS) is given as a tuple $\mathcal{T} = (Q, \Sigma, \iota, \delta)$, where Q is a finite, non-empty set of states, Σ is the finite alphabet, $\iota \in Q$ is the initial state and $\delta : Q \times \Sigma \to Q$ is the transition function. We use $|\mathcal{T}|$ to refer to the size of $\mathcal{T}$, which is the number of states in Q. The run of $\mathcal{T}$ on some word $w = w_0 w_1 \ldots \in \Sigma^* \cup \Sigma^\omega$ is the unique (possibly infinite) sequence $\rho = \mathsf{run}(\mathcal{T}, w) = q_0 w_0 q_1 w_1 \ldots$ with $q_0 = \iota$ and $q_{i+1} = \delta(q_i, w_i)$ for $i < |w|$. For a finite word w, we use $\mathcal{T}(w)$ to denote the last state of $\mathsf{run}(\mathcal{T}, w)$. We define the *infinity set*, referred to as $\mathsf{inf}(\rho)$, of a run $\rho = q_0 w_0 q_1 w_1 \ldots$, as the set containing all pairs (q, a) such that there exist infinitely many positions i with $q_i = q$ and $w_i = a$. For a finite run $\rho = q_0 w_0 \ldots w_{n-1} q_n$ and a set $X \subseteq Q \times \Sigma$, we use $\rho \cap X$ to denote the transitions of ρ that are in X, i.e. $\rho \cap X = \{(q_i, w_i) \in X \mid i < n\}$.

By equipping $\mathcal{T}$ with a set of *final states* $F \subseteq Q$, we obtain a *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, \iota, \delta, F)$. The language accepted by $\mathcal{A}$ is denoted as $L(\mathcal{A})$ and contains all words $w \in \Sigma^*$ such that $\mathcal{T}(w) \in F$. By combining a transition system $\mathcal{T}$ with a *Büchi condition* (on transitions), which is a set $\beta \subseteq Q \times \Sigma$, we obtain a *deterministic Büchi automaton* (DBA) $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$. Its accepted language $L(\mathcal{A})$ contains all infinite words w such that $\mathsf{inf}(\mathsf{run}(\mathcal{T}, w)) \cap \beta \neq \emptyset$. When we depict a DBA with acceptance condition β in a figure, we underline the label a of a transition leaving the state q, if $(q, a) \in \beta$. We can use the terminology for transition systems also for automata by simply ignoring the acceptance component. Similarly, the size of an automaton is equal to the size of the underlying transition system.

The class of DBA-recognizable languages is a strict subclass of the class of the regular ω-languages, which are defined in terms of nondeterministic Büchi automata. We do not detail the definition here because it is not relevant for our paper (see [24] for a survey of the topic). It is well known that two regular ω-languages, and hence also two DBA-recognizable languages, are equal iff they coincide on the ultimately periodic words [11] (see also [12]).

We call a relation $\sim \subseteq \Sigma^* \times \Sigma^*$ a *right congruence* if $u \sim w$ implies $ua \sim wa$ for all $a \in \Sigma$. For $u \in \Sigma^*$ we write $[u]_\sim$ for the set of all $v \in \Sigma^*$ with $u \sim v$, also referred to as the *equivalence class* of u. For a language $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$, we define the right congruence of L, denoted by $\sim_L$, as $u \sim_L v$ iff $u^{-1}L = v^{-1}L$.

A right congruence $\sim$ defines the canonical TS $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$ where $Q_\sim$ is the set of classes in $\sim$ and $\delta([u]_\sim, a) = [ua]_\sim$. Due to this correspondence, we write $c \in Q_\sim$ to denote that c is a class of $\sim$. Similarly, we can associate with every TS $\mathcal{T}$ a congruence $\sim_\mathcal{T} \subseteq \Sigma^* \times \Sigma^*$, where $u \sim_\mathcal{T} v$ iff $\mathcal{T}(u) = \mathcal{T}(v)$. Let p, q be states of some automaton $\mathcal{A}$ that accepts the language L, and let c be a class of $\sim_L$. By abuse of notation, we use $q \in c$ if $\mathcal{A}$ reaches q on some word $u \in c$. Further, we write $p \sim q$ if $p \in c$ and $q \in c$ for some class c. We use L_c for a class c of $\sim_L$ to denote the language $u^{-1}L$ for a $u \in c$.

An ω-*sample* is a pair $S = (S_+, S_-)$ with $S_+, S_- \subseteq \mathbb{UP}$ and $[\![S_+]\!]_\omega \cap [\![S_-]\!]_\omega = \emptyset$. We refer to words in S_+ as *positive*, while those in S_- are called negative. Since we only use ω-samples in our algorithm, we often simply write sample instead of ω-sample. We only consider finite samples, and the size of S, denoted as $|S|$, is defined as the sum of all $|u| + |v|$ for $(u, v) \in S_+ \cup S_-$. Furthermore, we call a sample $S = (S_+, S_-)$ consistent with a language L, if $L \cap [\![S_-]\!]_\omega = \emptyset$ and $[\![S_+]\!]_\omega \subseteq L$. Similarly, an automaton $\mathcal{A}$ is called *consistent* with S if $L(\mathcal{A})$ is consistent with S. We say that $u, v \in \Sigma^*$ are *separated by* S if there exists a word $w \in \Sigma^\omega$ such that $uw, vw \in \big([\![S_+]\!]_\omega \cup [\![S_-]\!]_\omega\big)$ and $\big(uw \in [\![S_+]\!]_\omega \Leftrightarrow vw \in [\![S_-]\!]_\omega\big)$. A DFA $\mathcal{D}$ is called *prefix-consistent* with the ω-sample $S = (S_+, S_-)$ if for all $w \in [\![S_+]\!]_\omega$ we have that $\mathsf{Prf}(w) \cap L(\mathcal{D}) \neq \emptyset$ and $\mathsf{Prf}([\![S_-]\!]_\omega) \cap L(\mathcal{D}) = \emptyset$.

A right congruence $\sim$ is consistent with S if $u \not\sim v$ for all $u, v \in \Sigma^*$ that are separated by S. In a *partial* TS $\mathcal{T}$, the transition function δ can have undefined values. While the definition of run also applies to partial TS, it is possible that the run of a partial TS on some words is undefined. A partial TS $\mathcal{T}$ has a conflict with a sample S if there are $u, v \in \Sigma^*$ that are separated by S but lead to the same state in $\mathcal{T}$ (which in particular means that the runs of $\mathcal{T}$ on u and v must exist). Note that $\sim$ is consistent with S iff the corresponding TS $\mathcal{T}_\sim$ has no conflict with S.

A passive learner (for DBAs) is a function f that maps ω-samples to DBAs. f is called a polynomial-time learner if f can be computed in polynomial time. A learner f is *consistent* if it constructs from each ω-sample $S = (S_+, S_-)$ a DBA $\mathcal{A}$ such that S is consistent with $L(\mathcal{A})$. We say that f can learn every DBA language in the limit if for each DBA language L there is a *characteristic sample* S^L such that $L(f(S^L)) = L$ and $f(S^L) = f(S)$ for each sample S that is consistent with L and contains S^L (so f produces the same DBA for L for all samples containing S^L). For a class $\mathcal{C}$ of DBA languages we say that f can learn every language in $\mathcal{C}$ in the limit *from polynomial data* if the characteristic samples for the languages in $\mathcal{C}$ are of polynomial size (in the smallest DBA for the corresponding language).

We also consider the standard active learning scenario for DFAs, in which the learning algorithm can obtain information on the target language L by posing membership and equivalence queries to an oracle [1]. In the following let L be a regular language. An oracle T for L answers a membership query $\mathsf{mem}(w)$ with either "yes" or "no", depending on whether the w is in the target language L or not. In an equivalence query $\mathsf{equiv}(\mathcal{A}_H)$, the active learner proposes a hypothesis $\mathcal{A}_H$ to T. If $\mathcal{A}_H$ accepts precisely the target language L, the oracle returns "yes". Otherwise, T answers "no" and additionally provides the learner with a word from the symmetric difference of $L(\mathcal{A}_H)$ and L, which serves as a counterexample. We make use of the following result.

▶ **Theorem 1** ([1], Theorem 6)**.** *Let L be a regular language and T be an oracle for L. There exists an active learner* AL *that returns the minimal DFA $\mathcal{A}_L$ for L. Moreover, the runtime of* AL *is polynomial in the size of the minimal DFA and the length of the longest counterexample provided by T, and the size of all hypotheses used in equivalence queries is polynomial in the size of $\mathcal{A}_L$.*

3 A Learner for DBAs

In this section we introduce DBAInf (for DBA inference), a passive learner for deterministic Büchi automata, and describe how it constructs a DBA from a finite ω-sample. We start with an informal description of the underlying idea, then provide a formal description of the learning algorithm, and finally show that DBAInf is a consistent polynomial time learner for the class of DBAs.

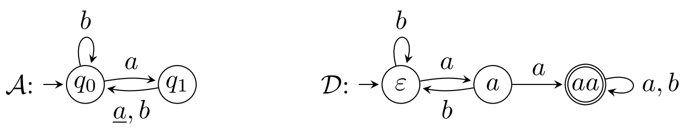

Figure 1 Illustrations for the high-level description of DBAInf. On the left a DBA $\mathcal{A}$ accepting the words in which the infix aa appears infinitely often. The DFA $\mathcal{D}$ on the right is constructed in an intermediate step of DBAInf from the ω-sample given in Example 2.

High-Level Description of the Idea

DFA-learners for languages of finite words are based on the fact that for the minimal DFA of a regular language, there is a one-to-one correspondence between states and equivalence classes of the right congruence of the language. DFA-learners basically extract a right congruence from the sample (consisting of finite words) that is then used for defining the transition system for the resulting DFA. The central obstacle in passive learning for ω-automata is the lack of such a one-to-one correspondence between states and equivalence classes. For example, the language L over the alphabet $\{a, b\}$ consisting of all words in which the infix aa occurs infinitely often cannot be recognized by a DBA with only one state. However, L has only one equivalence class, since for all $u \in \Sigma^*, w \in \Sigma^\omega$ we have $uw \in L$ if and only if $w \in L$. Thus, given an ω-sample that is consistent with L, the extracted right congruence will only have one class since the ω-sample does not separate any words.

We explain the idea underlying DBAInf for languages with only one equivalence class such as L, that is, if the sample does not separate any two words. The formal description then also covers the general case.

Basically, DBAInf attempts to identify specific subwords, also referred to as the *positive patterns of L* in the following description, that appear infinitely often precisely in words belonging to L. Consider, for example, the DBA $\mathcal{A}$ depicted in Figure 1 on the left, which accepts L. Regardless of which state $\mathcal{A}$ is in, reading the pattern aa makes $\mathcal{A}$ use an accepting transition, which means aa is a positive pattern of L. On the right-hand side of Figure 1 a DFA $\mathcal{D}$ that accepts all positive patterns of L is depicted. From $\mathcal{D}$ we can now obtain a DBA for L, which we refer to as DBA($\mathcal{D}$), by replacing each transition leading into a final state (state aa in the example) with an accepting transition that leads to the initial state (state ε in the example). In the example in Figure 1, the operation DBA($\mathcal{D}$) returns a DBA isomorphic to $\mathcal{A}$.

The core idea of DBAInf is to learn such a DFA for the positive patterns. For this purpose, given some ω-sample $S = (S_+, S_-)$, DBAInf uses an active learning algorithm AL for DFAs as black-box, and answers the queries of this algorithm based on S. To ensure that DBA($\mathcal{D}$) is consistent with S, we require that the active learning algorithm learns a DFA $\mathcal{D}$ that does not accept any infix of a loop of a negative example, and that for any position in a positive example an infix starting at this position is accepted. So whenever AL asks an equivalence query for a hypothesis $\mathcal{D}$, DBAInf checks whether $\mathcal{D}$ satisfies this condition and stops if yes. Otherwise, a positive or negative example is found on which $\mathcal{D}$ violates the condition and a corresponding finite word is returned.

▶ **Example 2.** Consider an ω-sample $S = (S_+, S_-)$ with $[\![S_+]\!]_\omega = \{(aba)^\omega\}$ and $[\![S_-]\!]_\omega = \{b^\omega, (ab)^\omega\}$. Recall that $[\![S_+]\!]_\omega$ refers to the ω-words, while S_+ itself is specified by pairs (u, v) of finite words representing uv^ω. The precise representation of the ultimately periodic words in $[\![S_+]\!]_\omega$ and $[\![S_-]\!]_\omega$ does not play any role. In the first step, DBAInf infers a right congruence that is consistent with S. Since S does not separate any finite words, this

Input: An ω-sample $S = (S_+, S_-)$.
Output: The deterministic Büchi automaton $\mathcal{A}$.

1. Extract a right congruence $\sim$ from S by constructing a TS $\mathcal{T}_\sim$
2. Build a sample $S^{(c)}$ for each $\sim$-class c
3. Compute a DFA $\mathcal{D}_c = \mathsf{PrfCons}(S^{(c)})$ for each class c of $\sim$
4. Return the DBA $\mathcal{A} := \mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c\in\sim})$

Figure 2 An overview of the algorithm DBAInf. The individual steps are explained in the text.

right congruence has only one class $c = [\varepsilon]_\sim$. For this class, DBAInf now uses an active learning algorithm AL for DFAs. In order to answer the queries of AL, DBAInf constructs an ω-sample $S^{(c)}$ with $S^{(c)}_+ = \{(\varepsilon, aab), (\varepsilon, aba), (\varepsilon, baa)\}$, representing all the suffixes of $[\![S_+]\!]_\omega$, and $S^{(c)}_- = \{(\varepsilon, b), (\varepsilon, ab), (\varepsilon, ba)\}$, representing all the suffixes of $[\![S_-]\!]_\omega$ starting at a position in a loop (which is the same in this example as all suffixes of $[\![S_-]\!]_\omega$). The precise execution of DBAInf now depends on the active learning algorithm AL that is used. We do not detail a precise active learner here but simply explain how queries would be answered. The queries that we use in this example correspond to a version of Angluin's algorithm by Rivest and Schapire [22], see also [7, Section 19.4].

AL starts by asking membership queries for ε, a, b, which are all answered negatively because these are all prefixes of $[\![S^{(c)}_-]\!]_\omega$ (and hence infixes of loops of negative examples from S). Then AL asks an equivalence query for the DFA that rejects all words. Since this DFA does not accept prefixes of all words in $[\![S^{(c)}_+]\!]_\omega$, DBAInf selects the smallest $(u, v) \in S^{(c)}_+$ such that no prefix of uv^ω is accepted by the DFA. In this example this is (ε, aab). The counterexample that is given to AL is the shortest prefix of $[\![(\varepsilon, aab)]\!]_\omega = (aab)^\omega$ that is not a prefix of $[\![S^{(c)}_-]\!]_\omega$, in this case aa. After this counterexample, AL asks a few more membership queries, namely for $ba, aaa, ab, aba, aaaa, aaba$ which are answered negatively for prefixes of $[\![S^{(c)}_-]\!]_\omega$ (that is, ba, ab, aba), and positively for all other words. With this information, AL asks an equivalence query for the DFA $\mathcal{D}$ shown in Figure 1. Since $\mathcal{D}$ accepts a prefix for each word in $[\![S^{(c)}_+]\!]_\omega$ and rejects all prefixes of words in $[\![S^{(c)}_-]\!]_\omega$, the execution of AL ends. Then DBAInf returns the DBA obtained by the operation $\mathsf{DBA}(\mathcal{D})$, which is isomorphic to $\mathcal{A}$ in Figure 1. ◀

This idea can be generalized to more than one equivalence class by first extracting a right congruence $\sim$ from S that is consistent with S, and then learning one DFA for each equivalence class c of $\sim$ with a more refined definition of the samples $S^{(c)}$ that is illustrated for a single class in Example 2. These DFAs are combined into a DBA by taking the product of $\mathcal{T}_\sim$ with the union of the $\mathcal{D}_c$, and redirecting transitions of the DFAs that lead to an accepting state into the initial state of the $\mathcal{D}_c$ for the current class c given by $\mathcal{T}_\sim$. The details are given in the formal description of the algorithm.

Formal Description of the Learner

The overall structure of the algorithm DBAInf in shown in Figure 2. The individual steps are described in more detail below. In the description, $S = (S_+, S_-)$ always refers to the ω-sample that is the input for DBAInf. After the description, we illustrate the steps with an example.

Step 1. The algorithm DBAInf starts by constructing a right congruence $\sim$ that is consistent with S, represented by a TS $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$. For obtaining an algorithm that can learn every DBA language in the limit, it is important to use a method that can infer the right congruence $\sim_L$ of any DBA language L in the limit. This is possible by slightly modifying one of the algorithms described in [5, 10], which learn ω-automata with an acceptance condition, while we are only interested in a right congruence in this step. Because of this different setting, we briefly describe a possible construction for $\mathcal{T}_\sim$ similar to the algorithm in [10]. The transition system $\mathcal{T}_\sim$ is built incrementally by considering prefixes of positive examples in $[\![S_+]\!]_\omega$ that exit the transition system (via an undefined transition). In order to guarantee termination in polynomial time, only prefixes up the length $k \cdot \ell^2$ are considered, with $k = \max\{|u| \mid (u,v) \in S_+\}$ and $\ell = \max\{|v| \mid (u,v) \in S_+\}$. The construction of $\mathcal{T}_\sim$ proceeds as follows:

- Start with only the initial state $\varepsilon \in Q_\sim$ and no transitions. Then repeat:
  - Pick the smallest u and a in canonical order such that ua is a prefix of a positive example word, $|ua| \leq k \cdot \ell^2$, and $\delta_\sim(u,a)$ is not yet defined. If no such ua exists, exit the loop.
  - If there is $v \in Q_\sim$ such that setting $\delta_\sim(u,a) = v$ does not lead to a conflict of $\mathcal{T}_\sim$ with S, then define $\delta_\sim(u,a) = v$ for the least such v in canonical order. Otherwise, add ua to $Q_\sim$ and let $\delta_\sim(u,a) = ua$.
- If there remain positive examples in $[\![S_+]\!]_\omega$ for which the run in $\mathcal{T}_\sim$ is not defined (because of missing transitions), complete $\mathcal{T}_\sim$ by adding appropriate paths to disjoint loops for all such elements of $[\![S_+]\!]_\omega$.
- Finally, add a sink state as target for all remaining undefined transitions.

This construction runs in polynomial time (because of the limit on the length of the considered ua, and since all tests can be carried out in polynomial time). By construction, $\mathcal{T}_\sim$ has no conflict with S, and hence the associated right congruence $\sim$ is consistent with S, as required. In the following, we identify states in $Q_\sim$ with the corresponding equivalence classes of $\sim$.

Step 2. For each equivalence class $c \in Q_\sim$, we define a sample $S^{(c)} = (S^{(c)}_+, S^{(c)}_-)$ based on the infixes of example words starting in positions i such that $\mathcal{T}_\sim(w[1,i-1]) = c$. Formally, we have that $S^{(c)}_+$ contains for each $xw \in [\![S_+]\!]_\omega$ with $\mathcal{T}_\sim(x) = c$ the minimal $(u,v) \in \mathbb{UP}$ such that $w = uv^\omega$. We define $S^{(c)}_-$ to contain for each $xw \in [\![S_-]\!]_\omega$ with $\mathcal{T}_\sim(x) = c$ the minimal (ε, v) such that $w = v^\omega$ and $\mathcal{T}_\sim(xv) = c$, if such a word v exists.

Note that this definition formally ranges over infinitely many x, but for each (u,v) in S it suffices to consider the prefixes x of length at most $|u| + |v||Q_\sim|$ because the run of uv^ω in $\mathcal{T}_\sim$ has an initial part of length at most u followed by a period of length at most $|v||Q_\sim|$. This observation gives a polynomial bound on the number of elements in $S^{(c)}$ and also on their size.

Step 3. In the third step, DBAInf constructs for each class c of $\sim$ a DFA $\mathcal{D}_c$ that is prefix-consistent with $S^{(c)}$. This construction is achieved by passing $S^{(c)}$ to the algorithm PrfCons, which is depicted in Algorithm 1. Given an ω-sample $R = (R_+, R_-)$, PrfCons uses a polynomial-time active learning algorithm AL for DFAs, and answers the queries of AL based on R. We assume that R_- only contains periodic words, which is satisfied by the samples $S^{(c)}$ on which PrfCons is applied.

▶ **Lemma 3.** *For every ω-sample $R = (R_+, R_-)$ with $R_- \subseteq \{\varepsilon\} \times \Sigma^+$,* PrfCons *terminates in polynomial time and returns a DFA that is prefix-consistent with R.*

Proof. We first show that all answers given to AL during the execution of PrfCons are consistent with the language $P = \Sigma^* \setminus \mathsf{Prf}([\![R_-]\!]_\omega)$: If a membership query $\mathsf{mem}(x)$ is answered positively, then $x \notin \mathsf{Prf}([\![R_-]\!]_\omega)$ and hence $x \in P$. Analogously, a negative answer

Algorithm 1 PrfCons.

```
Input: An $\omega$-sample $R = (R_+, R_-)$ with $R_- \subseteq (\{\varepsilon\} \times \Sigma^+)$.
Output: A DFA $\mathcal{D}$ that is prefix-consistent with $R$.
Simulate an active learning algorithm AL for DFAs that satisfies the properties of
  Theorem 1, and answer its queries as follows:
Query mem($x$)
    if $x \in \mathsf{Prf}([\![R_-]\!]_\omega)$ then
        answer "no"
    else
        answer "yes"
Query equiv($\mathcal{D}$)
    if $\mathcal{D}$ is prefix-consistent with $R$ then
        stop simulation and output $\mathcal{D}$
    else if there exists a $w \in [\![R_+]\!]_\omega$ with $\mathsf{Prf}(w) \cap L(\mathcal{D}) = \emptyset$ then
        pick the minimal $(u, v) \in R_+$ such that $uv^\omega \cap L(\mathcal{D}) = \emptyset$
        return shortest $x \sqsubseteq uv^\omega$ with $x \notin \mathsf{Prf}([\![R_-]\!]_\omega)$ as counterexample
    else
        let $(\varepsilon, v) \in R_-$ be minimal such that $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$
        return shortest prefix $x$ of $v^\omega$ with $x \in L(\mathcal{D})$ as counterexample
```

to $\mathsf{mem}(x)$ implies $x \in \mathsf{Prf}([\![R_-]\!]_\omega)$, meaning $x \notin P$. For an equivalence query $\mathsf{equiv}(\mathcal{D})$, we distinguish the two cases in PrfCons. In the first case, the returned counterexample x is not accepted by $\mathcal{D}$ and is not in $\mathsf{Prf}([\![R_-]\!]_\omega)$. So $x \in P$ and giving x as counterexample means that it should be accepted. In the other case, the query is answered with a prefix $x \in \mathsf{Prf}(v^\omega) \cap L(\mathcal{D})$ for some $(\varepsilon, v) \in R_-$ such that x is accepted by $\mathcal{D}$. So $x \notin P$ and should be accepted. Hence, all answers provided to AL are consistent with the language P.

Since a DFA that accepts P is prefix-consistent with R, the execution of AL either terminates with a DFA for P, or it terminates earlier if a prefix-consistent DFA $\mathcal{D}$ is used in an equivalence query. Therefore, all hypotheses $\mathcal{D}$ used by AL are of size polynomial in the size of a minimal DFA for P (see Theorem 1). It is not hard to verify that P can be accepted by a DFA whose size is polynomial in $|R|$ using the prefixes of $[\![R_-]\!]_\omega$ as non-accepting states, and introducing a loop on v for each $(\varepsilon, v) \in R_-$ when reaching a prefix of v^ω that is not a prefix of any other word in $[\![R_-]\!]_\omega$. Adding an accepting sink for all missing transitions results in a DFA for P. (A similar construction is described in more detail in [5] for the construction of the "table look-up DPA".)

To conclude that the execution of AL terminates in polynomial time, it remains to verify that the lengths of the provided counterexamples are indeed polynomial in $|R|$. The counterexample for an equivalence query on $\mathcal{D}$ is a shortest word that is prefix of an example from R and is accepted/rejected by $\mathcal{D}$ (depending on the case). These words are clearly polynomial in the size of R and $\mathcal{D}$, and hence polynomial in the size of R. ◀

Step 4. The constructed DFAs are combined into a DBA by taking the product of $\mathcal{T}_\sim$ with the union of the $\mathcal{D}_c$, and redirecting transitions of the DFAs that lead to an accepting state into the initial state of the $\mathcal{D}_c$ for the current class c given by $\mathcal{T}_\sim$.

Formally, we assume that the state sets of the $\mathcal{D}_c$ are pairwise disjoint, and define $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim}) = (Q, \Sigma, \iota, \delta, \beta)$ as follows:

- $Q := Q_\sim \times \bigcup_{c \in Q_\sim} Q_c$
- $\iota := ([\varepsilon]_\sim, \iota_{[\varepsilon]_\sim})$ (note that $\iota_{[\varepsilon]_\sim}$ is the initial state of $\mathcal{D}_{[\varepsilon]_\sim}$)

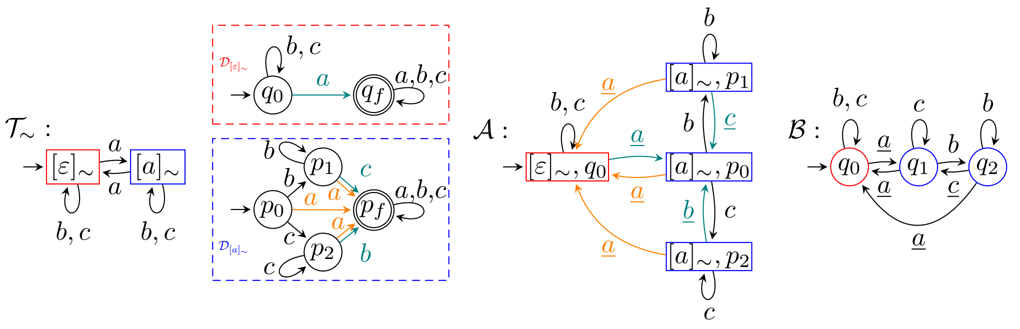

Figure 3 Illustrations for the execution of DBAInf on the sample S given in Example 4.

- For $(c, q) \in Q$ and $a \in \Sigma$ let $q' = \delta_{\hat{c}}(q, a)$ for the unique $\hat{c} \in Q_\sim$ with $q \in Q_{\hat{c}}$, and let $c' = \delta_\sim(c, a)$. Then
  - $\delta((c,q),a) = \begin{cases} (c', q') & \text{if } q' \notin F_{\hat{c}} \\ (c', \iota_{c'}) & \text{if } q' \in F_{\hat{c}} \end{cases}$
  - $((c, q), a) \in \beta$ if $q' \in F_{\hat{c}}$.

This DBA is returned by the algorithm DBAInf. Note that syntactically, a tuple of the form $(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ is the same as a family of DFAs, which are considered for active learning of ω-languages in [3]. The semantics is, however, very different. Families of DFAs can represent all regular ω-languages but turning them into a deterministic ω-automaton for the represented language involves an exponential blow-up, in general [2]. So the way we use these tuples of DFAs here is not related to these families of DFAs. Before we prove that DBAInf is a consistent polynomial time DBA-learner, we illustrate it with an example.

▶ **Example 4.** As an example for the execution of DBAInf, consider $S = (S_+, S_-)$ with

$[\![S_+]\!]_\omega = \{a^\omega, (ba)^\omega, (ca)^\omega, a(bc)^\omega, a(bcb)^\omega, a(cbc)^\omega\}$ and
$[\![S_-]\!]_\omega = \{b^\omega, c^\omega, (bc)^\omega, aa(bc)^\omega, ab^\omega, ac^\omega\}$.

DBAInf first extracts a right congruence $\sim$ consisting of two equivalence classes. The corresponding transition system $\mathcal{T}_\sim$ is shown in Figure 3 on the left. The words ε and a are separated by S, since $a(bc)^\omega \in S_+$ and $(bc)^\omega \in S_-$. The b, c-loops on $[\varepsilon]_\sim$ do not introduce a conflict, as well as the a-transition from $[a]_\sim$ to $[\varepsilon]_\sim$. The b-transition from $[a]_\sim$ cannot go back to $[\varepsilon]_\sim$ because this would introduce a conflict for the words b and ab, which are separated by S, again because of $a(bc)^\omega \in S_+$ and $(bc)^\omega \in S_-$. Hence, there is a b-loop on $[a]_\sim$. Similarly, for the c-transition from $[a]_\sim$. The resulting transition system is complete, so no further states have to be added.

In the second step, the samples $S^{[\varepsilon]_\sim}$ and $S^{[a]_\sim}$ are computed. The positive components of these samples are $S_+^{[\varepsilon]_\sim} = \{(\varepsilon, a), (\varepsilon, ba), (\varepsilon, ab), (\varepsilon, ca), (\varepsilon, ac), (a, bc), (a, bcb), (a, cbc)\}$ and $S_+^{[a]_\sim} = \{\varepsilon\} \times \{a, ba, ab, ca, ac, bc, cb, bcb, bbc, cbb, cbc, bcc, ccb\}$.

To illustrate how these samples are constructed, consider $(ba)^\omega \in S_+$. It induces the sequence of classes $([\varepsilon]_\sim[\varepsilon]_\sim[a]_\sim[a]_\sim)^\omega$ in $\mathcal{T}_\sim$ and thus contributes (ε, ba) and (ε, ab) to both $S_+^{[\varepsilon]_\sim}$ and $S_+^{[a]_\sim}$. The example $a(bcb)^\omega$ induces the sequence $[\varepsilon]_\sim([a]_\sim)^\omega$, and hence contributes (a, bcb) to $S_+^{[\varepsilon]_\sim}$, and $\{\varepsilon\} \times \{bcb, cbb, bbc\}$ to $S_+^{[a]_\sim}$. Similarly, for the other positive examples.

The negative components are $S_-^{[\varepsilon]_\sim} = \{\varepsilon\} \times \{b, c, bc, cb\}$ and $S_-^{[a]_\sim} = \{\varepsilon\} \times \{b, c\}$. For example, $aa(bc)^\omega$ induces the class sequence $[\varepsilon]_\sim[a]_\sim([\varepsilon]_\sim)^\omega$. This adds (ϵ, bc) and (ε, cb) to $S_-^{[\varepsilon]_\sim}$ and nothing to $S_-^{[a]_\sim}$ because the only position with class $[a]_\sim$ after the first a is not in the periodic part, and hence there is no v such that $av^\omega = aa(bc)^\omega$. Similarly, for the other negative examples.

Executing PrfCons on these samples gives rise to two DFAs $\mathcal{D}_{[\varepsilon]_\sim}$ and $\mathcal{D}_{[a]_\sim}$ with 2 and 4 states, respectively, which are depicted in the middle of Figure 3. Next to that (on the right) in Figure 3, the DBA $\mathcal{A}$ that is the result of $\mathsf{DBA}(\mathcal{T}_\sim, \mathcal{D}_{[\varepsilon]_\sim}, \mathcal{D}_{[a]_\sim})$ is shown. The states of $\mathcal{A}$ are pairs of states of $\mathcal{T}_\sim$ and of the union of $\mathcal{D}_{[\varepsilon]_\sim}$ and $\mathcal{D}_{[a]_\sim}$. Accepting transitions that reset into $\mathcal{D}_{[\varepsilon]_\sim}$ are depicted in orange and those that reset into $\mathcal{D}_{[a]_\sim}$ are colored teal. On the very right of Figure 3, a minimal DBA $\mathcal{B}$ that accepts the same language as $\mathcal{A}$ is depicted. The states of $\mathcal{B}$ are colored according to the class of $\sim$ that they belong to.

Consistency

We now prove that DBAInf always infers a DBA that is consistent with the given ω-sample.

▶ **Theorem 5.** DBAInf *is a consistent polynomial time DBA-learner.*

Proof. Let $S = (S_+, S_-)$ be an ω-sample and $\sim$ be a right congruence that is consistent with S, as constructed by DBAInf in the first step (represented by $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$). As in Figure 2, let $\mathcal{D}_c$ be the DFA constructed from $S^{(c)}$ for each class c of $\sim$. Further, let $\mathcal{A} = \mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c\in Q_\sim})$ be the DBA returned by DBAInf. The extraction of $\mathcal{T}_\sim$ and the construction of the samples $S^{(c)}$ can be done in polynomial time. By Lemma 3 PrfCons runs in polynomial time for each class c and returns a DFA $\mathcal{D}_c$ whose size is polynomial in $|S|$. Hence, the construction of $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c\in Q_\sim})$ and thus the overall procedure DBAInf can be completed in polynomial time.

For consistency, first note that in the construction of $\mathcal{A}$, the first component of the states is always updated according to $\delta_\sim$. Therefore, for each $u \in \Sigma^*$, the state (c, q) reached by $\mathcal{A}$ after reading u is such that c is the class of u, i.e., $\mathcal{T}_\sim$ reaches c when reading u.

We start by showing that $\mathcal{A}$ rejects all words in S_-. Let $w = uv^\omega$ for a $(u, v) \in S_-$. As w is ultimately periodic, we can write the run of $\mathcal{A}$ on w as $\rho\pi^\omega$. Pick a position $i \geq \max\{|u|, |\rho|\}$ such that $\mathcal{A}$ uses an accepting transition when reaching position i in w. If no such position exists, then $\mathcal{A}$ uses only finitely many accepting transitions and hence rejects w. It is easily verified that if such a position i exists, we can write $uv^\omega = xy^\omega$ and $\rho\pi^\omega = \hat{\rho}(\hat{\pi})^\omega$ such that $|x| = |\hat{\rho}| = i$ and $|y| = |\hat{\pi}|$. Let c be the class reached after reading x, meaning $\mathcal{T}_\sim(x) = c$ and $\mathcal{A}$ is in state (c, ι_c) after reading x. This means $\hat{\pi}$ starts and ends in (c, ι_c) and thus $\delta_\sim(c, y) = c$. Since $xy^\omega = uv^\omega \in [\![S_-]\!]_\omega$, we have $(\varepsilon, y) \in S_-^{(c)}$ by the definition of $S^{(c)}$. Because the DFA $\mathcal{D}_c$ is prefix-consistent with $S^{(c)}$ by Lemma 3, it does not accept any prefix of y^ω. Thus, $\mathcal{D}_c$ will never reach a final state when reading the suffix of w starting at position i. By definition this means $\mathcal{A}$ uses no accepting transitions after position i in w, and hence rejects w.

Now let us show that $\mathcal{A}$ accepts all words in S_+. Let $w = uv^\omega \in [\![S_+]\!]_\omega$ and let ρ be the run of $\mathcal{A}$ on w. Let i be a position such that ρ is in a state of the form (c, ι_c) at position i. Then $\mathcal{A}$ uses an accepting transition after position i: If ρ is in state (c, ι_c) at position i, this means that also $\mathcal{T}_\sim(w[1, i-1]) = c$, and we can find words $u \in \Sigma^*, w' \in \Sigma^\omega$ such that $|u| = i$ and $w = uw'$. By definition of $S_+^{(c)}$, we have that $w' \in [\![S_+^{(c)}]\!]_\omega$. Since $\mathcal{D}_c$ is prefix-consistent with $S^{(c)}$, there exists a prefix $x \sqsubseteq w'$ that is accepted by $\mathcal{D}_c$. Hence, $\mathcal{A}$ uses an accepting transition on the prefix x of w'. From that we can conclude that ρ uses infinitely many accepting transitions since ρ starts in $([\varepsilon]_\sim, \iota_{[\varepsilon]_\sim})$ and each accepting transition ends in a state of the form (c, ι_c) for a class c of $\sim$. ◀

4 Completeness of the Learning Algorithm

In this section we establish that the class of DBA languages can be learned in the limit by DBAInf. For obtaining this completeness result, we show how to construct a sample S^L for a DBA language L such that DBAInf constructs a DBA for L from each sample S that contains S^L and is consistent with L. We separate the construction of S^L in two parts. The first part ensures that DBAInf infers the right congruence $\sim_L$ in the first step. The second part ensures that, based on the correct right congruence $\sim_L$, a DBA for L is constructed. We keep the first part short as the idea for this is the same as for RPNI on finite words [21] and the passive learner for ω-automata from [9]. Roughly, the sample has to cover all transitions and states of $\mathcal{T}_{\sim_L}$ and provide examples that separate all different states. This has to be done in a specific way using minimal words reaching the states and transitions of $\mathcal{T}_{\sim_L}$.

▶ **Lemma 6.** *For every DBA language L there exists an ω-sample $S^{\sim_L}$ such that $|S^{\sim_L}|$ is polynomial in the size of L and* DBAInf *extracts $\sim_L$ from every sample that is consistent with L and contains $S^{\sim_L}$.*

The remainder of this section is about the second part of the construction of S^L: DBAInf returns a DBA that consists of smaller DFAs $\mathcal{D}_c$ for each class c of the extracted congruence $\sim$. We want to ensure that these DFAs satisfy certain conditions that are captured in the definition of "safe" below.

▶ **Definition 7.** *Let L be a DBA language and*

$$\mathcal{K}(L) = \{u \in \Sigma^+ \mid \textit{for all } y \in \Sigma^* \textit{ if } (uy)^{-1}L = L \textit{ then } (uy)^\omega \in L\}.$$

We call a DFA $\mathcal{D}$ safe *for L, if*
1. *for each word $w \in L$, $\mathcal{D}$ accepts some prefix of w, and*
2. *$L(\mathcal{D}) \subseteq \mathcal{K}(L)$.*

Our goal is to build the sample S^L in such a way that the samples $S^{(c)}$ constructed in the second step of DBAInf ensure that AL learns a DFA that is safe for L_c according to Definition 7 above. Then Lemma 10 further below shows that the DBA returned by DBAInf accepts L. For this purpose, we consider for each class c an execution of AL in which the answers given to the queries of AL are consistent with $\mathcal{K}(L_c)$. The answers used in this run of AL are used to define the part of the sample that ensures that PrfCons($S^{(c)}$) returns a DFA that is safe for L_c (see Algorithm 2, Lemma 11, and Lemma 12). In order to ensure that the execution of AL terminates, we need to show that $\mathcal{K}(L_c)$ is regular, which we do first in Lemma 8 and Lemma 9. For this we assume that $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$ is some DBA that accepts L. Let

$$\mathcal{K}_q(\mathcal{A}) = \{u \in \Sigma^* \mid \text{ for all } v \in \Sigma^* : \text{ if } q \xrightarrow{uv} q \text{ then } \mathsf{run}(\mathcal{A}, uv) \cap \beta \neq \emptyset\}$$

be the set of all words u, that do not lie on a rejecting loop starting in the state q. As an example consider the DBA $\mathcal{B}$ on the right of Figure 3. We have $\mathcal{K}_{q_0}(\mathcal{B}) = \Sigma^* a \Sigma^*$, $\mathcal{K}_{q_1}(\mathcal{B}) = \Sigma^*(a+b)\Sigma^*$, and $\mathcal{K}_{q_2}(\mathcal{B}) = \Sigma^*(a+c)\Sigma^*$. In general, a DFA for $\mathcal{K}_q(\mathcal{A})$ can be constructed by using $\mathcal{A}$ with initial state q, and one new accepting sink state to which all accepting transitions of $\mathcal{A}$ are redirected, and all transitions from states p such that each path from p to q contains an accepting transition.

▶ **Lemma 8.** *For every DBA $\mathcal{A}$ and each state $q \in Q$, the language $\mathcal{K}_q(\mathcal{A})$ is regular and can be recognized by a DFA of size at most $|\mathcal{A}| + 1$.*

Proof. To construct a DFA that recognizes $\mathcal{K}_q(\mathcal{A})$, we first extract the transition system $\mathcal{T}$ underlying $\mathcal{A}$. Then we remove from $\mathcal{T}$ all transitions which are accepting in $\mathcal{A}$. Subsequently, we build the set G consisting of all transitions $q \xrightarrow{a} p$ such that p and q lie in different SCCs of $\mathcal{T}$. This allows us to finally define the DFA $\mathcal{B} = (Q \cup \{\top\}, \Sigma, q, \delta', \{\top\})$ with

$$\delta'(q,a) = \begin{cases} p & \text{if } (q \xrightarrow{a} p) \notin (\beta \cup G) \\ \top & \text{otherwise.} \end{cases}$$

It is easily verified that the size of $\mathcal{B}$ is indeed linear in $|Q|$ as only one state is added. Further, we can show that $\mathcal{B}$ accepts $\mathcal{K}_q(\mathcal{A})$ by considering both directions of the mutual inclusion:

- Let $w \in L(\mathcal{B})$, then the run of $\mathcal{B}$ on w must end in the only final state, $\top$. This state can clearly only be reached if a transition $\tau \in beta \cup G$ is used. If $\tau \in \beta$, then $\mathcal{A}$ must use the accepting transition τ on w and hence any extension of w leading back to q is also guaranteed to use an accepting transition. Otherwise, $\tau \in G$ and thus from q the word w reaches a state $p \in \mathcal{T}$, which lies in a different SCC. This either happens if p and q lie in different SCCs in $\mathcal{A}$ or if all paths leading from p to q use at least one accepting transition in G. Clearly in both cases we have $w \in \mathcal{K}_q(\mathcal{A})$.
- On any word $w \in \mathcal{K}_q(\mathcal{A})$, the DBA $\mathcal{A}$ reaches from q some state p such that either p is in a different SCC or every path returning to q uses an accepting transition. In the former case the two states also lie in different SCCs and hence there exists some transition τ connecting the SCCs. By construction, however, we know that $\tau \in G$ and hence it is redirected to the final state $\top$ in $\mathcal{B}$. On the other hand if both states lie in the same SCC, then removing all accepting transitions guarantees that q can no longer be reached from p, as every path uses an accepting transition. Thus, p and q are in different SCCs in $\mathcal{T}$ and thus by an analogous argument, $\mathcal{B}$ accepts w. ◀

We can now express $\mathcal{K}(L_c)$ as intersection of the languages $\mathcal{K}_q(\mathcal{A})$ for states q in class c. As example, consider again the DBA $\mathcal{B}$ on the right of Figure 3. The states q_1 and q_2 belong to the class $[a]_\sim$. The intersection $\mathcal{K}_{q_1}(\mathcal{B}) \cap \mathcal{K}_{q_2}(\mathcal{B})$ consists of all words that contain a or contain both b and c. This is the language $\mathcal{K}(L(\mathcal{B})_{[a]_\sim})$.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a DBA, $L = L(\mathcal{A})$ and c be a class of $\sim_L$. Then $\mathcal{K}(L_c) = \bigcap_{q\in c} \mathcal{K}_q(\mathcal{A})$, and in particular $\mathcal{K}(L_c)$ is regular.*

Proof. For the first inclusion of the equality, let $u \in \mathcal{K}(L_c)$. Assume to the contrary that there is some $q \in c$ with $u \notin \mathcal{K}_q(\mathcal{A})$. Then there is $v \in \Sigma^*$ such that in $\mathcal{A}$ we have $q \xrightarrow{uv} q$ without using an accepting transition. Note that $q \in c$ implies that $\mathcal{A}$ with initial state q accepts L_c. But then $(uv)^{-1}L_c = L_c$ and $(uv)^\omega \notin L_c$, which is a contradiction to $u \in \mathcal{K}(L_c)$.

For the inclusion from right to left, let $u \in \mathcal{K}_q(\mathcal{A})$ for all states q with $q \in c$. Consider an arbitrary $v \in \Sigma^*$ with $(uv)^{-1}L_c = L_c$. We need to show that $(uv)^\omega \in L_c$ because then $u \in \mathcal{K}(L_c)$. We pick some state $q_0 \in c$ and consider the run $\rho = q_0 \xrightarrow{uv} q_1 \xrightarrow{uv} q_2 \xrightarrow{uv} \dots$ of $\mathcal{A}$ on $(uv)^\omega$ from q_0. For all i, we have $q_i \in c$ since $(uv)^{-1}L_c = L_c$. There is some state q such that the set of indices $I = \{i \in \mathbb{N} \mid q = q_i\}$ is infinite. But then for any $i, j \in I$ with $i < j$ there must be an accepting transition between q_i and q_j in ρ by definition of $\mathcal{K}_q(\mathcal{A})$, and hence $(uv)^\omega \in \mathcal{K}_c(L)$.

Regularity of $\mathcal{K}(L_c)$ follows as it is a finite intersection of languages that are regular by Lemma 8. ◀

In the following Lemma, we show that if each constructed DFA $\mathcal{D}_c$ for a class c satisfies the conditions of Definition 7 with respect to the language L_c, then the resulting DBA accepts precisely L.

▶ **Lemma 10.** *Let $L \subseteq \Sigma^\omega$ be a DBA-recognizable language, and for each $c \in Q_{\sim_L}$ let $\mathcal{D}_c$ be a DFA that is safe for L_c. Then $\mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c\in Q_{\sim_L}})$ accepts L.*

Proof. In the following let $\mathcal{A} = \mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c\in Q_{\sim_L}})$. For the first inclusion consider a word $w \in L$. The run of $\mathcal{A}$ on w begins in (c_0, ι_{c_0}), where $c_0 = [\varepsilon]_{\sim_L}$ and ι_{c_0} refers to the initial state of $\mathcal{D}_{c_0}$. As $w \in L$, we clearly also have that $w \in L_{c_0}$, which by the first condition in the definition of "safe" (Definition 7) implies that $v_0 \in L(\mathcal{D}_{c_0})$ for some prefix v_0 of w. Thus, on the prefix v_0, $\mathcal{A}$ uses an accepting transition and reaches some state (c_1, ι_{c_1}) for

$c_1 = \mathcal{T}_{\sim_L}(v_0)$. Now let w_1 be the infinite word such that $w = v_0 w_1$. Since $\mathcal{T}_{\sim_L}$ reaches c_1 on v_0, we have that $w_1 \in L_{c_1}$ and thus by the first condition there exists a prefix $v_1 \sqsubseteq w_1$ such that $v_1 \in L(\mathcal{D}_{c_1})$. Thus, the run of $\mathcal{A}$ from (c_1, ι_{c_1}) on v_1 uses an accepting transition and reaches the state (c_2, ι_{c_2}) for some class c_2 with $\mathcal{T}_{\sim_L}(v_0 v_1) = c_2$. Repeated application of this argument yields a factorization $w = v_0 v_1 \dots$ and an infinite sequence $(c_i)_{i \in \mathbb{N}}$ with $c_0 = [\varepsilon]_{\sim_L}$, such that $c_{i+1} = \delta^*_{\sim_L}(c_i, v_i)$ and $v_i \in L(\mathcal{D}_{c_i})$ for all $i \in \mathbb{N}$. The run ρ of $\mathcal{A}$ on $w = v_0 v_1 \dots$ can similarly be factorized into $\rho = \rho_0 \rho_1 \dots$, such that ρ_i begins in (c_i, ι_{c_i}) and ends in $(c_{i+1}, \iota_{c_{i+1}})$. Additionally, each of these factors ρ_i uses an accepting transition. This means that the run of $\mathcal{A}$ on w uses infinitely many accepting transitions and thus $w \in L(\mathcal{A})$.

For the other inclusion, let $w \in L(\mathcal{A})$ and ρ be the run of $\mathcal{A}$ on w. Since ρ uses infinitely many accepting transitions, there exists a factorization $w = v_0 v_1 \dots$ and a sequence $(c_i)_{i \in \mathbb{N}}$ with $c_0 = [\varepsilon]_{\sim_L}$ such that $(c_0, \iota_{c_0}) \xrightarrow{v_0} (c_1, \iota_{c_1}) \xrightarrow{v_1} \dots$ in $\mathcal{A}$ and on each v_i the last transition is accepting and all the other transitions are non-accepting. By construction of $\mathcal{A}$, each v_i is accepted by the DFA $\mathcal{D}_{c_i}$ and hence by the second condition of "safe" (Definition 7) also $v_i \in \mathcal{K}(L_c)$. Consider now any DBA $\mathcal{B}$ that accepts L. By Lemma 9 we know that $\mathcal{K}(L_c) = \bigcap_{q \in c} \mathcal{K}_q(\mathcal{B})$ for all classes $c \in Q_{\sim_L}$. Let π be the unique run of $\mathcal{B}$ on $w = v_0 v_1 \dots$, then π can be written as $\iota_B = q_0 \xrightarrow{v_0} q_1 \xrightarrow{v_1} \cdots$. As $\mathcal{B}$ has a finite number of states, we know that there exists an infinite set I of indices such that $q_i = q_j$ for all $i, j \in I$. Note that because $\mathcal{B}$ accepts L, we have $q_i \in c_i$ for all $i \in \mathbb{N}$. But then since $v_i \in \mathcal{K}_{c_i}(L)$ and $\mathcal{K}_{c_i}(L) \subseteq \mathcal{K}_{q_i}(\mathcal{B})$ (by Lemma 9) for all $i \in I$, between any two visits to a state q_i with $i \in I$, $\mathcal{B}$ uses an accepting transition. As I is an infinite set, this guarantees that the run of $\mathcal{B}$ on w uses infinitely many accepting transitions and hence $w \in L(\mathcal{B}) = L$. ◀

In the following, we describe how a sample can be constructed to guarantee that PrfCons yields a DFA that is safe for some DBA language L. For this we use a specific execution of the active learning algorithm AL, shown in Algorithm 2. We refer to this specific execution as the L-run of AL and denote the sample that it produces with $S^{\mathsf{AL},L}$. We present the L-run in form of an algorithm, but we are only interested in the definition of the sample $S^{\mathsf{AL},L}$ and not its computation. Therefore, we do not go into further detail regarding the computation of each individual operation. Note that the L-run of AL is defined along the same lines as the algorithm PrfCons shown in Algorithm 1, but now the queries are answered based on the language L and not based on a sample. The ω-words added to the sample $S^{\mathsf{AL},L}$ ensure that PrfCons will give the same answers to the queries of AL for any sample that includes $S^{\mathsf{AL},L}$ and is consistent with L.

▶ **Lemma 11.** *For a DBA-recognizable language L, the size of $S^{\mathsf{AL},L}$ is polynomial in the size of a minimal DFA for $\mathcal{K}(L)$. Furthermore, the DFA $\mathcal{D}$ returned by the L-run of* AL *is safe for L.*

Proof. We first explain why all the answers of the L-run to the queries of AL are consistent with $\mathcal{K}(L)$. For the membership queries this is obvious from the definition of $\mathcal{K}(L)$. For equivalence queries, consider the first case. The word x that is given as counterexample satisfies the definition of $\mathcal{K}(L)$ and is not accepted by the current hypothesis $\mathcal{D}$, so giving this counterexample is consistent with $\mathcal{K}(L)$. Let us argue that such a word x always exists. Let $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$ be a DBA with $L(\mathcal{A}) = L$. Since $uv^\omega \in L$, it is accepted from all states that are equivalent to the initial state. So there is a prefix $z \sqsubseteq uv^\omega$ such that for every q that is equivalent to ι, the run from q on uv^ω uses an accepting transition on the prefix z. Now assume that y is such that $(zy)^{-1}L = L$. Then the run of $\mathcal{A}$ on $(zy)^\omega$ is accepting

Algorithm 2 Definition of $S^{\mathsf{AL},L}$ by a specific execution of AL, called L-run of AL.

```
Input: A DBA accepting $L \subseteq \Sigma^\omega$.
Output: The $\omega$-sample $S^{\mathsf{AL},L} = (S_+^{\mathsf{AL},L}, S_-^{\mathsf{AL},L})$ and the DFA $\mathcal{D}$
$S_+^{\mathsf{AL},L} \leftarrow \emptyset, S_-^{\mathsf{AL},L} \leftarrow \emptyset$
Simulate an active learning algorithm AL that satisfies the properties of Theorem 1,
  and answer its queries as follows:
Query mem($x$)
    if $(xy)^\omega \notin L$ for some $y \in \Sigma^*$ with $(xy)^{-1}L = L$ then
        $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, xy)\}$ for the shortest such $y$ and answer “no”
    else
        answer “yes”
Query equiv($\mathcal{D}$)
    if there exists a word $w \in L$ with $\mathsf{Prf}(w) \cap L(\mathcal{D}) = \emptyset$ then
        pick the minimal $(u, v)$ such that $uv^\omega \in L$ and $\mathsf{Prf}(uv^\omega) \cap L(\mathcal{D}) = \emptyset$
        $S_+^{\mathsf{AL},L} \leftarrow S_+^{\mathsf{AL},L} \cup \{(u, v)\}$
        choose minimal $x \sqsubseteq uv^\omega$ with $(xy)^{-1}L = L \Rightarrow (xy)^\omega \in L$ for all $y \in \Sigma^*$
        forall $x' \sqsubseteq x$ with $x' \neq x$ do
            choose the minimal $y'$ such that $(x'y')^\omega \notin L$ and $(x'y')^{-1}L = L$
            $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, x'y')\}$
        return $x$ as counterexample
    else if there is some $v \in \Sigma^+$ with $v^\omega \notin L, v^{-1}L = L$ and $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$ then
        pick $v$ to be the minimal word with this property
        $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, v)\}$
        return minimal $x \sqsubseteq v^\omega$ such that $x \in L(\mathcal{D})$ as counterexample
    else
        terminate the execution of AL and output $S^{\mathsf{AL},L}$, $\mathcal{D}$
```

because it uses an accepting transition on each z-segment. Hence, z is in $\mathcal{K}(L)$ and there exists a minimal prefix x of uv^ω with this property that can be given as counterexample. In the second case of an equivalence query, the selected $x \sqsubseteq v^\omega$ is accepted by $\mathcal{D}$, but it is not in $\mathcal{K}(L)$ because there is a y such that $xy = v^k$ for some k.

Thus, all answers to AL are consistent with $\mathcal{K}(L)$, and therefore all hypothesis DFAs $\mathcal{D}$ are of polynomial size in $\mathcal{A}_{\mathcal{K}(L)}$ (since AL satisfies the conditions of Theorem 1). From that one can derive that the counterexamples given to AL, and also the examples added to $S^{\mathsf{AL},L}$ for equivalence queries are of polynomial size in $\mathcal{A}_{\mathcal{K}(L)}$. Hence, the computation time taken by AL is also polynomial in $\mathcal{A}_{\mathcal{K}(L)}$, and the size of the words in membership queries is polynomial in $\mathcal{A}_{\mathcal{K}(L)}$. This implies that also the size of the examples added to $S^{\mathsf{AL},L}$ for the answers of membership queries is polynomial in $\mathcal{A}_{\mathcal{K}(L)}$.

It remains to show that the DFA $\mathcal{D}$ computed by the L-run of AL is safe for L (see Definition 7). The first case of an equivalence query guarantees that for each $w \in L$, some prefix $x \sqsubseteq w$ is accepted by $\mathcal{D}$. To verify that the second condition of Definition 7 is satisfied, let $u \in L(\mathcal{D})$. If $u \notin \mathcal{K}(L)$, then there exists a word $y \in \Sigma^*$ such that $(uy)^{-1}L = L$ and $(uy)^\omega \notin L$. This means that the second case of the equivalence query matches with $v = uy$, and hence the simulation of $\mathcal{D}$ will not stop as long as $\mathcal{D}$ accepts a word outside $\mathcal{K}(L)$. ◀

For a DBA-recognizable language L, we now define the sample

$$S^L = S^{\sim_L} \cup \big(\bigcup_{c \in Q_{\sim_L}} r_c \cdot S^{\mathsf{AL},L_c} \big) \text{ for the smallest } r_c \in \Sigma^* \text{ such that } \mathcal{T}_{\sim_L}(r_c) = c$$

where the union and concatenation operation on samples is done component wise, and "smallest" refers to the length-lexicographic ordering. The next lemma establishes that each DFA $\mathcal{D}_c$ for a class c that DBAInf constructs from a consistent sample containing S^L is safe for L_c.

▶ **Lemma 12.** *Let L be a DBA language with right congruence $\sim$. If $S = (S_+, S_-)$ is consistent with L and contains S^L, then for each class c, $\mathsf{PrfCons}(S^{(c)})$ returns a DFA $\mathcal{D}_c$ that is safe for L_c.*

Proof. Since S is consistent with L, we know that $[\![S_+^{(c)}]\!]_\omega \subseteq L_c$ and $[\![S_-^{(c)}]\!]_\omega \cap L_c = \emptyset$. We show that $\mathsf{PrfCons}(S^{(c)})$ gives the same answers to AL as the L_c-run.

- For a membership query $\mathsf{mem}(x)$:
  - If "no" is answered during the L_c-run of AL, then there exists some $(\varepsilon, xy) \in S_-^{\mathsf{AL},L_c}$ with $(xy)^\omega \notin L_c$ and $(xy)^{-1}L_c = L_c$. By definition, we have $r_c(xy)^\omega \in [\![S_-^L]\!]_\omega$ for some $r_c \in \Sigma^*$ with $\mathcal{T}_\sim(r_c) = c$. Because $(xy)^{-1}L_c = L_c$, it further holds that also $\mathcal{T}_\sim(r_c xy) = c$. This means $(xy)^\omega \in [\![S_-^{(c)}]\!]_\omega$ and thus $\mathsf{PrfCons}(S^{(c)})$ also answers "no".
  - If the L_c-run of AL answers "yes", then $(xy)^\omega \in L_c$ for all $y \in \Sigma^*$ with $(xy)^{-1}L_c = L_c$. But then x cannot be a prefix of $[\![S_-^{(c)}]\!]_\omega$ because $S_-^{(c)}$ contains only periodic words and $[\![S_-^{(c)}]\!]_\omega \cap L_c = \emptyset$.
- Consider an equivalence query $\mathsf{equiv}(\mathcal{D})$:
  - Whenever the L_c-run goes to the first case, the minimal (u, v) with $uv^\omega \in L_c$ and $\mathsf{Prf}(uv^\omega) \cap L(\mathcal{D}) = \emptyset$ is in S_+^{AL,L_c}. By definition, this means $(r_c u, v) \in S_+^L \subseteq S_+$ and thus $\mathsf{PrfCons}(S^{(c)})$ also goes to the first case. Note that there cannot be a $(\hat{u}, \hat{v}) \in S_+^{(c)}$ with $\mathsf{Prf}(\hat{u}\hat{v}^\omega) \cap L(\mathcal{D}) = \emptyset$ and $(\hat{u}, \hat{v}) \prec (u, v)$, as otherwise $(\hat{u}, \hat{v})$ would have been selected by the L_c-run. Further, $\mathsf{PrfCons}(S^{(c)})$ picks the same prefix $x \sqsubseteq uv^\omega$: For all strict prefixes $x' \sqsubset x$, we have $(r_c, x'y') \in S_-^{\mathsf{AL},L}$ for the minimal $y' \in \Sigma^*$ such that $(x'y')^{-1}L_c = L_c$ and some $r_c \in \Sigma^*$ with $\mathcal{T}_\sim(r_c) = c$. But then $x' \in \mathsf{Prf}([\![S_-^{(c)}]\!]_\omega)$ and thus $\mathsf{PrfCons}(S^{(c)})$ cannot return x' as a counterexample.
  - If the L_c-run goes to the second case, then $\mathcal{D}$ accepts a prefix of all words in L_c and hence for each $w \in [\![S_+^{(c)}]\!]_\omega$ we have $\mathsf{Prf}(w) \cap L(\mathcal{D}) \neq \emptyset$. Thus, $\mathsf{PrfCons}(S^{(c)})$ also goes to the second case. By construction, we have that $(\varepsilon, v) \in S_-^{\mathsf{AL},L}$ for the minimal v with $v^\omega \notin L_c, v^{-1}L_c = L_c$ and $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$. Thus, $(r_c, v) \in S_-^L \subseteq S_-$, which guarantees that $(\varepsilon, v) \in S_-^{(c)}$. There can be no $(\varepsilon, \hat{v}) \in S_-^{(c)}$ with $\hat{v} \prec v$ and $\mathsf{Prf}(\hat{v}^\omega) \cap L(\mathcal{D}) \neq \emptyset$, since $(\varepsilon, \hat{v}) \in S_-^{(c)}$ implies $\hat{v}^\omega \notin L_c$ and $\hat{v}^{-1}L_c = L_c$, which would contradict the minimality of v. Therefore, $\mathsf{PrfCons}(S^{(c)})$ returns the same counterexamples as the L_c-run.

So $\mathsf{PrfCons}(S^{(c)})$ computes the same DFA as the L_c-run and therefore this DFA is safe for L_c by Lemma 11. ◀

By combining the previous results, we are now able to establish that DBAInf can learn all DBA-recognizable languages in the limit.

▶ **Theorem 13.** DBAInf *is a polynomial-time DBA-learner that learns every DBA-recognizable language in the limit.*

Proof. Let L be a DBA-recognizable language and $S = (S_+, S_-)$ be a sample that is consistent with L and contains S^L. Polynomial runtime of DBAInf was already established in Theorem 5, so it remains to show that DBAInf constructs a DBA for L from the sample S. By Lemma 6, we know that DBAInf extracts $\sim_L$ from S. In the third step, DBAInf constructs a DFA $\mathcal{D}_c$ from $S^{(c)}$ for each $\sim_L$-class c. It is guaranteed by Lemma 12 that each $\mathcal{D}_c$ is safe for L_c. Thus, we can conclude from Lemma 10 that $L(\mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c \in Q_{\sim_L}})) = L$ and hence DBAInf returns a DBA which accepts precisely L. ◀

5 Sample Size

As stated in Theorem 13, DBAInf can learn every DBA language in the limit, i.e., for each DBA language L there is a characteristic ω-sample S^L such that DBAInf constructs a DBA for L when executed on an ω-sample S that is consistent with L and contains S^L. In this section, we analyze the size of this sample in terms of the size of L, which is the size of a smallest DBA for L. The characteristic sample S^L that is constructed in Section 4 consists of the following components:

- The sample $S^{\sim_L}$ whose size is polynomial in the size of L, see Lemma 6.
- The samples S^{AL,L_c} (prefixed by a word r_c) that ensure that the DFAs $\mathcal{D}_c$ satisfy the properties of Lemma 10. The size of these samples is polynomial in the size of a minimal DFA for $\mathcal{K}(L_c)$ according to Lemma 11.

This raises the question on the size of the minimal DFA for $\mathcal{K}(L_c)$ compared to the size of L. It turns out that this size can be exponential in the size of L.

▶ **Proposition 14.**
1. *Let L be recognizable by a DBA $\mathcal{A}$ with n states, and let c be a class of $\sim_L$. The number of states of the minimal DFA for $\mathcal{K}(L_c)$ is in $O((n+1)^k)$ where k is the number of states of $\mathcal{A}$ that are in class c.*
2. *There is a family $(L^{(k)})_{k \geq 1}$ of DBA languages such that $\sim_{L^{(k)}}$ has only one class, L_k can be accepted by a DBA with k states, and the minimal DFA for $\mathcal{K}(L^{(k)})$ has 2^k many states for each $k \geq 1$.*

Proof (sketch). The first claim directly follows from $\mathcal{K}(L_c) = \bigcap_{q \in c} \mathcal{K}_q(\mathcal{A})$ (see Lemma 9) and Lemma 8. For the second claim, let $\Sigma_k = \{1, \dots, k\}$ for $k > 0$, and define the language $L^{(k)} \subseteq \Sigma_k^\omega$ as the set of all ω-words that contain infinitely many occurrences of each symbol from Σ_k. Since membership in $L^{(k)}$ does not depend on any finite prefix, each $\sim_{L^{(k)}}$ has only one class. Each $L^{(k)}$ can be accepted by a DBA $\mathcal{A}^{(k)}$ with k states that repeatedly verifies the occurrence of each symbol from Σ_k in ascending order. The language $\mathcal{K}(L^{(k)})$ consists of all u that contain all letters from Σ_k. It is not difficult to check that the minimal DFA for $\mathcal{K}(L^{(k)})$ has 2^k states. ◀

This means that the size of the characteristic sample for L that we construct in our completeness proof can be of size exponential in the size of L. But if we fix the number of states in each equivalence class, then we obtain a class of DBA languages that can be learned in the limit from polynomial data by DBAInf. For this purpose, we say that a DBA language L has a k-informative right congruence if it can be accepted by a DBA with at most k different states per $\sim_L$ equivalence class. By k-IRC(DBA) we denote the corresponding class of languages. The DBA languages with informative right congruence from [4] correspond to the class 1-IRC(DBA). As a direct consequence of Proposition 14 and the fact that the characteristic sample for L is polynomial in the size of L and $\mathcal{K}(L)$, we obtain:

▶ **Theorem 15.** *For every fixed k,* DBAInf *can learn every language in k-IRC(DBA) in the limit from polynomial data (the degree of the polynomial depends on k).*

The existing polynomial time learners for deterministic ω-automata used with a Büchi condition are known to learn every language in 1-IRC(DBA) from polynomial data [5, 9]. The algorithm in [5] cannot learn any DBA language outside of 1-IRC(DBA), and while the algorithm in [9] can learn DBA languages outside 1-IRC(DBA), there are very simple DBA languages that it cannot learn.

For understanding the worst-case exponential size of the characteristic sample for L, we take a closer look at the operation $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ that is used for constructing the DBA $\mathcal{A}$ from the DFAs $(\mathcal{D}_c)_{c \in Q_\sim}$ computed in Step 3. (see Figure 2). Lemma 10 asserts that $\mathcal{A}$ accepts L if each $\mathcal{D}_c$ is safe for L_c (see Definition 7). In the completeness proof we use DFAs $\mathcal{D}_c$ for the languages $\mathcal{K}(L_c)$, which are safe of L_c but can be of exponential size in $\mathcal{A}$ as shown in Proposition 14. This raises the question whether DFAs that are safe for L_c need to be, in general, of exponential size. We show that this is not the case, formally stated in Lemma 17 further below. For the proof of Lemma 17 we use the following lemma.

▶ **Lemma 16.** *Let $\mathcal{A}$ be a DBA with n states and $w \in \Sigma^*$ be word. If $\mathcal{A}$ uses n^2+1 accepting transitions on the run on w starting in a state q, then $\mathcal{A}$ uses at least one accepting transition on its run on w from each state p with $p \sim_{L(\mathcal{A})} q$.*

Proof (sketch). If this is not the case, one can find two language equivalent states p, q and a word $u \in \Sigma^*$ such that u loops on p without an accepting transition, and u loops on q with an accepting transition in the loop, contradicting the language equivalence of p and q. ◀

▶ **Lemma 17.** *Let $\mathcal{A}$ be a DBA with n states, $L = L(\mathcal{A})$ and $\sim$ be the right congruence of L. For each class c of $\sim$ there is a DFA D_c of size polynomial in $|\mathcal{A}|$ such that $\mathcal{D}_c$ is safe for L_c, and thus $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ accepts L.*

Proof (sketch). For a class c of $\sim$ pick a state $q_c \in c$ as initial state of $\mathcal{D}_c$, and simulate $\mathcal{A}$ while incrementing a counter each time $\mathcal{A}$ uses an accepting transition. If this counter exceeds n^2+1, then go to an accepting sink. From Lemma 16 and Lemma 9 it follows that $\mathcal{D}_c$ is safe for L_c. ◀

This shows that the way how we compose a DBA from DFAs can, in principle, produce polynomial size DBAs for each DBA language L. So the reason for the exponential size of the characteristic sample of L in the completeness proof is not enforced by the operation that is used to build the DBA from the DFAs, but is rather coming from the way how we extract information from that sample to obtain the DFAs.

6 Conclusion

We have presented a passive learning algorithm DBAInf for DBAs that constructs a consistent DBA in polynomial time for a given ω-sample, and can learn every DBA language in the limit. Previously, the only known class of ω-languages learnable in the limit was the class of languages with informative right congruence [5, 9], whose definition eliminates one of the most difficult properties of ω-automata, namely that deterministic ω-automata often need several language equivalent states for accepting a language. While the characteristic samples for DBAInf that are used in the completeness proof are of exponential size in the worst-case, we obtain learnability in the limit from polynomial data for the class of DBAs that have no more than k states that are all pairwise language equivalent, for each fixed k. This includes the class of DBA languages with informative right congruence for $k = 1$.

Our algorithm uses an active learning algorithm for DFAs as black-box. Our first attempts to build a DBA learner, following the same basic idea but using a passive learner for DFAs instead of an active one, failed. The reason for this seems to be that one carefully needs to select the information from the ω-sample that used for building the DFAs, in order to obtain a robust DBA learner with the learning in the limit property. An active learning algorithm selects this information with its queries. In future work we plan to investigate whether these ideas can be extended to deal with all regular ω-languages by learning deterministic parity automata. It is also an open question whether our approach can be improved in order to obtain learnability in the limit from polynomial data for all DBA languages.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

2 Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as acceptors of omega-regular languages. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: http://www.dagstuhl.de/dagpub/978-3-95977-016-3.

3 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.

4 Dana Angluin and Dana Fisman. Regular omega-languages with an informative right congruence. In *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*, volume 277 of *EPTCS*, pages 265–279, 2018. doi:10.4204/EPTCS.277.19.

5 Dana Angluin, Dana Fisman, and Yaara Shoval. Polynomial identification of ω-automata. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part II*, volume 12079 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2020. doi:10.1007/978-3-030-45237-7_20.

6 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

7 Therese Berg and Harald Raffelt. Model checking. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, chapter 19. Springer, 2005. doi:10.1007/11498490_25.

8 Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. doi:10.1109/TC.1972.5009015.

9 León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.20.

10 León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the rpni algorithm, 2021. arXiv:2108.03735.

11 J Richard Büchi. On a decision method in restricted second order arithmetic, logic, methodology and philosophy of science (proc. 1960 internat. congr.), 1962.

12 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Mathematical Foundations of Programming Semantics*, pages 554–566, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

13 E Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. doi:10.1016/S0019-9958(67)91165-5.

14 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.

15 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.

16 Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015. doi:10.1007/978-3-319-21690-4_32.

17 Damian López and Pedro Garcíía. On the inference of finite state automata from positive and negative data. In Sempere J. In: Heinz J., editor, *Topics in Grammatical Inference.* Springer, 2016. doi:10.1007/978-3-662-48395-4_4.

18 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi:10.1006/inco.1995.1070.

19 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.

20 Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2370–2377. IOS Press, 2020. doi:10.3233/FAIA200367.

21 Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. doi:10.1142/9789812797902_0004.

22 Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Machine Learning: From Theory to Applications*, volume 661 of *Lecture Notes in Computer Science*, pages 51–73. Springer, 1993.

23 Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.400.

24 Wolfgang Thomas. *Automata on Infinite Objects*, pages 133–191. MIT Press, Cambridge, MA, USA, 1991.

25 Wolfgang Thomas. Facets of synthesis: Revisiting Church's problem. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009.

26 Boris A. Trakhtenbrot and Y.M. Barzdin. *Finite Automata: Behavior and Synthesis.* North-Holland Publishing Company, Amsterdam, 1973.

27 Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pages 638–642. IEEE Computer Society, 2017. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8090480.