Angluin-Style Learning of Deterministic Büchi and Co-Büchi Automata

Yong Li[1,2], Sven Schewe[1] and Qiyi Tang[1]

[1] Department of Computer Science, University of Liverpool
[2] Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

{liyong, svens, qiyitang}@liverpool.ac.uk

Abstract

While recently developed Angluin-style learning algorithms for ω-automata have much in common with her classic DFA learning algorithm, there is a huge difference in the cost of the queries. These active learning algorithms work with an oracle that can answer membership and equivalence queries. For ω-regular languages, however, the target is to learn nondeterministic Büchi automata through the vehicle of Families of DFAs (FDFAs). While the assumption that membership queries are relatively cheap remains reasonable, equivalence queries for nondeterministic automata are PSPACE-complete, which restricts their use.

We develop efficient techniques for the cases, where we learn *deterministic* Büchi (or co-Büchi) automata. This is based on the observation that some classes of FDFAs can be used to learn deterministic Büchi automata for DBA recognisable languages, rather than having to resort to nondeterministic ones. Different to the high—PSPACE—cost of testing language equivalence for NBAs, this operation is cheap—NL—for DBAs (and DCAs), which makes equivalence queries realistic.

1 Introduction

In her seminal paper, Angluin [Angluin, 1987] proposed a learning framework that can learn an automaton representation of an unknown regular language R from an oracle. The learning algorithm or the learner can interact with the oracle by means of two types of queries, namely membership and equivalence queries. While membership queries ask whether a word u belongs to R, equivalence queries ask whether a given automaton correctly recognises the target language R. After asking a certain number of membership queries, the learner is able to propose a conjectured automaton and ask an equivalence query about the conjecture. When the oracle returns a positive answer to an equivalence query, the learner has completed his task and successfully learned R; otherwise, the learner will receive a counterexample from the oracle, which he will use to refine the current conjectured automaton. This learning procedure will continue until a correct automaton of R has been learned.

Since its introduction, Angluin-style learning frameworks have, for example, been applied in learning assumptions for compositional verification [Cobleigh *et al.*, 2003], detecting bugs in network protocol implementations [de Ruiter and Poll, 2015], and extracting automata models for recurrent neural networks [Weiss *et al.*, 2018].

Angluin-style learning has initially focused on learning automata that represent regular languages, especially deterministic finite automata (DFAs) [Angluin, 1987; Isberner *et al.*, 2014; Vaandrager *et al.*, 2022], but also nondeterministic finite automata [Bollig *et al.*, 2009], and alternating automata [Angluin *et al.*, 2015]. More recently, they have branched out into learning ω-regular languages represented by ω-automata, so far focusing on nondeterministic Büchi automata (NBAs) [Farzan *et al.*, 2008; Li *et al.*, 2021], where the current vehicle for learning them are families of DFAs (FDFAs) [Angluin and Fisman, 2016; Li *et al.*, 2023a].

While NBAs are popular in verification, they are hard to reason about, because equivalence checking of NBAs is PSPACE-complete. While FDFAs themselves are easy to manipulate [Angluin *et al.*, 2018], they have not yet found applications outside of learning.

For languages recognisable by deterministic Büchi automata (DBAs) or deterministic co-Büchi automata (DCAs), we may well encounter a situation, where the oracle is in effect in possession of a DBA or a DCA to evaluate. It will then be easy for her to answer equivalence questions to DBAs or DCAs, respectively, whereas the answer to a PSPACE-hard question might require a trip to Delphi, while we can turn to a run-of-the-mill oracle if our conjecture automata are also presented as DBAs and DCAs, respectively.

But can we make use of these cheap equivalence queries? Considering that FDFAs naturally translate to NBAs, the answer to this question is not straightforward. However, we observe that a translation from FDFAs in a particular normal form—limit FDFAs [Li *et al.*, 2023a]—to a DBA that recognises a sub-language of the limit FDFAs, but will, for DBA recognisable languages, converge to the full language when the learning of the limit FDFA is complete. The tricky bit is to cover the case, where the counterexample is not in the language of the conjectured DBA, but is both in the language of the FDFA (or: the language of the NBA that represents it) and the target language.

The refinement of the FDFA for this case is slightly more involved than usual, but this complication is minor compared to the significant decrease in complexity—from PSPACE to NL—for the equivalence query itself. This balance of the expressiveness of learned languages and the complexity of equivalence queries provides the *first* Angluin-style learning algorithm for DBAs, the main contribution of this paper. Moreover, since DCAs are dual to DBAs, our algorithm can be easily adapted for learning DCAs by learning a DBA of the complement language of the target co-Büchi language.

Related work. Recent work has studied learning DBAs (and even deterministic parity automata) [Michaliszyn and Otop, 2022]; however, this work not only requires the oracle to answer membership and equivalence queries, but also needs to know the loop index of each queried infinite word in the target automaton. Such queries about the loop index of infinite words may not be feasible in some scenarios such as learning a representation of black-box systems where the inner structure of the system is unknown; it therefore does not fit into Angluin's learning framework. Angluin's learning framework can be classified as *active learning*, in contrast to *passive learning*, where automata are learned from a given set of labelled samples. We note that there is a passive learning algorithm for DBAs proposed in [Bohn and Löding, 2022], which is orthogonal to our work. Angluin-style learning framework has been suggested for the *smaller* class of weak Büchi automata [Maler and Pnueli, 1995]. This work, however, only covers a strict subset of DBA languages behaving like DFAs in which the states of weak Büchi automata simply represent the right congruence classes [Myhill, 1957; Nerode, 1958].

2 Preliminaries

In the whole paper, we fix a finite *alphabet* Σ. A *word* is a finite or infinite sequence of letters in Σ; ε denotes the empty word. Let Σ^* and Σ^ω denote the set of all finite and infinite words (or ω-words), respectively. In particular, we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A *finitary language* is a subset of Σ^*; an ω*-language* is a subset of Σ^ω. Let ρ be a sequence; we denote by $\rho[i]$ the i-th element of ρ and by $\rho[i..k]$ the subsequence of ρ starting at the i-th element and ending at the $(k-1)$-th element when $0 \leq i < k$, and the empty sequence ε when $i \geq k$. We denote by $\rho[i\ldots]$ the subsequence of ρ starting at the i-th element when $i < |\rho|$, and the empty sequence ε when $i \geq |\rho|$. Given a finite word u and a word w, we denote by $u \cdot w$ (uw, for short) the concatenation of u and w.

Transition system. A (nondeterministic) transition system (TS) is a tuple $\mathcal{T} = (Q, q_0, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to 2^Q$ is a transition function. We also lift δ to sets as $\delta(S, \sigma) := \bigcup_{q \in S} \delta(q, \sigma)$. We also extend δ to words in a usual way, by letting $\delta(S, \varepsilon) = S$ and $\delta(S, u \cdot a) = \delta(\delta(S, u), a)$, where $u \in \Sigma^*$ and $a \in \Sigma$.

Automata. An automaton on finite words is called a *nondeterministic finite automaton* (NFA). An NFA $\mathcal{A}$ is formally defined as a tuple $(\mathcal{T}, F)$, where $\mathcal{T}$ is a TS and $F \subseteq Q$ is a set of *final* states. An automaton on ω-words is called a *nondeterministic Büchi automaton* (NBA). An NBA $\mathcal{B}$ is represented as a tuple $(\mathcal{T}, \Gamma)$ where $\mathcal{T}$ is a TS and $\Gamma \subseteq \{(q, a, q') : q, q' \in Q, a \in \Sigma, q' \in \delta(q, a)\}$ is a set of *accepting* transitions. An NFA $\mathcal{A}$ is a *deterministic* finite automaton (DFA) if, for each $q \in Q$ and $a \in \Sigma$, $|\delta(q, a)| \leq 1$. Deterministic Büchi automata (DBAs) are defined similarly and thus Γ is a subset of $\{(q, a) : q \in Q, a \in \Sigma\}$, since the successor q' is determined by the source state and the input letter.

A *run* of an NFA $\mathcal{A}$ on a finite word u of length $n \geq 0$ is a sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \leq i < n$, $q_{i+1} \in \delta(q_i, u[i])$. We write $q_0 \xrightarrow{u} q_n$ if there is a run from q_0 to q_n over u. A finite word $u \in \Sigma^*$ is *accepted* by an NFA $\mathcal{A}$ if there is a run $q_0 \cdots q_n$ over u such that $q_n \in F$. Similarly, an ω*-run* of $\mathcal{A}$ on an ω-word w is an infinite sequence of transitions $\rho = (q_0, w[0], q_1)(q_1, w[1], q_2) \cdots$ such that, for every $i \geq 0$, $q_{i+1} \in \delta(q_i, w[i])$. Let $\inf(\rho)$ be the set of transitions that occur infinitely often in ρ. An ω-word $w \in \Sigma^\omega$ is *accepted* by an NBA $\mathcal{A}$ if there is an ω-run ρ of $\mathcal{A}$ over w such that $\inf(\rho) \cap \Gamma \neq \emptyset$. The *finitary language* recognised by an NFA $\mathcal{A}$, denoted $\mathcal{L}_*(\mathcal{A})$, is defined as the set of finite words accepted by it. Similarly, we denote by $\mathcal{L}(\mathcal{A})$ the ω*-language* recognised by an NBA $\mathcal{A}$, i.e. the set of ω-words accepted by $\mathcal{A}$. NFAs/DFAs accept exactly *regular* languages while NBAs recognise exactly ω*-regular* languages.

Deterministic co-Büchi automata (DCA) are dual to DBAs and have the same structure as DBAs except that w is accepted by a DCA if its run satisfies that $\inf(\rho) \cap \Gamma = \emptyset$. For DCAs, Γ is called the set of *rejecting* transitions.

Right congruences. A *right congruence* (RC) relation is an equivalence relation $\backsim$ over Σ^* such that $x \backsim y$ implies $xv \backsim yv$ for all $v \in \Sigma^*$. We denote by $|\backsim|$ the index of $\backsim$, i.e. the number of equivalence classes of $\backsim$. A *finite RC* is an RC with a finite index. We denote by $\Sigma^*/_{\backsim}$ the set of equivalence classes of Σ^* under $\backsim$. Given $x \in \Sigma^*$, we denote by $[x]_{\backsim}$ the equivalence class of $\backsim$ that x belongs to.

For a given regular language R, one can define the RC $\backsim_R$ of R as $x \backsim_R y$ if, and only if, $\forall v \in \Sigma^*.\ xv \in R \iff yv \in R$ [Myhill, 1957; Nerode, 1958]. The RC $\backsim_R$ also defines the minimal DFA $\mathcal{D}$ of R, in which each state of $\mathcal{D}$ corresponds to an equivalence class in $\Sigma^*/_{\backsim}$. Formally, the TS $\mathcal{T}[\backsim]$ of $\mathcal{D}$ is defined as follows.

Definition 1 ([Myhill, 1957; Nerode, 1958]). *Let $\backsim$ be an RC of finite index. The TS $\mathcal{T}[\backsim]$ induced by $\backsim$ is a tuple (S, s_0, δ) where $S = \Sigma^*/_{\backsim}$, $s_0 = [\varepsilon]_{\backsim}$, and for each $u \in \Sigma^*$ and $a \in \Sigma$, $\delta([u]_{\backsim}, a) = [ua]_{\backsim}$.*

The minimal DFA $\mathcal{D}$ of R is the DFA $\mathcal{D} = (\mathcal{T}[\backsim_R], F_{\backsim_R})$ where $F_{\backsim_R}$ collects all classes $[u]_{\backsim_R}$ such that $u \in R$.

Ultimately periodic words. For ω-regular languages, we only need to consider a type of ω-words called *ultimately periodic* (UP) words; a UP-word w is of the form uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$. For an ω-language L, let $\mathrm{UP}(L) = \{\, uv^\omega \in L \mid u \in \Sigma^* \wedge v \in \Sigma^+ \,\}$ denote the set of all UP-words in L. By [Büchi, 1962; Calbrix *et al.*, 1993], two ω-regular languages L and L' are equivalent if, and only if, $\mathrm{UP}(L) = \mathrm{UP}(L')$. That is, the set of UP-words of an ω-regular language L *uniquely* characterises L.

As aforementioned, a UP-word $w = uv^\omega$ can be denoted as a pair of finite words such as (u, v), (uv, v) and other valid pairs; they are all called a *decomposition* of w.

Families of DFAs (FDFAs). FDFAs have been introduced to recognise an ω-regular language L by accepting the decompositions of UP(L) [Angluin *et al.*, 2018].

Definition 2 ([Angluin *et al.*, 2018])**.** *An FDFA is a pair* $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^q\})$ *consisting of a leading DFA* $\mathcal{M}$ *and of a progress DFA* $\mathcal{N}^q$ *for each state* q *in* $\mathcal{M}$*.*

Intuitively, for the FDFA $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^q\})$ to accept a UP-word $uv^\omega \in$ UP(L), the leading DFA $\mathcal{M}$ first consumes the finite prefix u, reaching some state q and, for each state q of $\mathcal{M}$, the progress DFA $\mathcal{N}^q$ accepts the loop word v. Note that the leading DFA $\mathcal{M}$ of every FDFA is in fact only a TS since it does *not* make use of final states.

Let A be a deterministic automaton with TS $\mathcal{T} = (Q, q_0, \delta)$ and $x \in \Sigma^*$. We denote by $A(x)$ the state $\delta(q_0, x)$. Each FDFA $\mathcal{F}$ accepts a set of UP-words UP($\mathcal{F}$) by using the following acceptance condition.

Definition 3 (Acceptance)**.** *Let* $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^q\})$ *be an FDFA and* w *be a UP-word. A decomposition* (u, v) *of* w *is* normalised *with respect to* $\mathcal{F}$ *if* $\mathcal{M}(u) = \mathcal{M}(uv)$*. A decomposition* (u, v) *is accepted by* $\mathcal{F}$ *if* (u, v) *is normalised and* $v \in \mathcal{L}_*(\mathcal{N}^q)$ *where* $q = \mathcal{M}(u)$*. Then,* w *is accepted by* $\mathcal{F}$ *if there exists a decomposition* (u, v) *of* w *accepted by* $\mathcal{F}$*.*

So, we can also see UP($\mathcal{F}$) as the set of words recognised by $\mathcal{F}$. In the remainder of the paper, we fix a target DBA-language L unless stated otherwise.

3 Outline of Our Algorithm

We give an overview of our DBA learning algorithm in this section; the framework is depicted in Fig. 1. Assume that we have a DBA oracle who knows L and can answer membership queries about L and equivalence queries about whether a given DBA recognises L. We note that using equivalence queries that involve NBA operations would significantly increase the complexity for resolving equivalence queries and lose all the advantage we aim to reap.

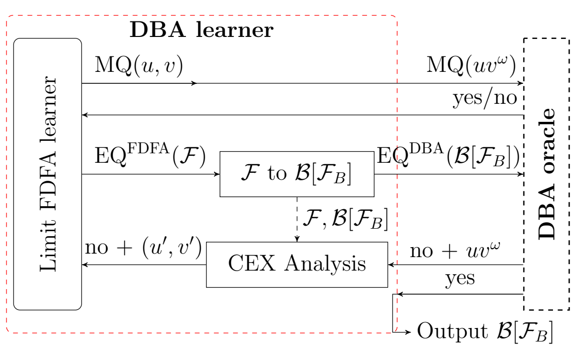


Figure 1: Overview of our DBA learning framework

Our DBA learner is comprised of three components: the limit FDFA learner (cf. Sect. 5), the component transforming an FDFA $\mathcal{F}$ to a DBA $\mathcal{B}[\mathcal{F}_B]$ (cf. Sect. 4), and a counterexample (CEX) analysis component (cf. Sect. 6). Limit FDFAs are a type of canonical FDFAs that can easily decide DBA-languages [Li *et al.*, 2023a] and thus are a natural choice in our DBA learning algorithm. In a nutshell, our DBA learner, corresponding to the dashed box on the left in Fig. 1, tries to use the limit FDFA learner to learn the canonical form of limit FDFAs $\mathcal{F}$ (and thus the sink FDFA $\mathcal{F}_B$ in Sect. 4) [Li *et al.*, 2023a] and then converts the sink FDFA $\mathcal{F}_B$ to a language-equivalent DBA $\mathcal{B}[\mathcal{F}_B]$.

More precisely, the DBA learner uses the FDFA learner to learn the limit FDFA $\mathcal{F}$ (and thus $\mathcal{B}[\mathcal{F}_B]$) by answering membership and equivalence queries posed by the limit FDFA learner, through interacting with the DBA oracle. We will use superscripts, FDFA and DBA, to distinguish the equivalence queries posed by our limit FDFA learner and the DBA learner respectively. To answer a membership query $\mathsf{MQ}(u, v)$, the DBA learner simply forwards the answer to the membership query $\mathsf{MQ}(uv^\omega)$ obtained from the DBA oracle. Answering an equivalence query $\mathsf{EQ}^{\text{FDFA}}(\mathcal{F})$ can be more involved.

The DBA learner needs to first construct a DBA $\mathcal{B}[\mathcal{F}_B]$ from $\mathcal{F}$ using Definition 7. ($\mathcal{F}_B$ is obtained from $\mathcal{F}$ by only allowing sink final states.) Then the DBA learner poses an equivalence query $\mathsf{EQ}^{\text{DBA}}(\mathcal{B}[\mathcal{F}_B])$ to the DBA oracle. If the DBA oracle returns "Yes", the DBA learner can just output the learned DBA $\mathcal{B}[\mathcal{F}_B]$: it has completed the learning task. Otherwise, the DBA learner receives "NO" along with a CEX $uv^\omega \in L \ominus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$. Then the DBA learner has to utilise the CEX analysis component to extract a CEX (u', v'), which may *not* be a decomposition of uv^ω but be *good* for refining $\mathcal{F}$ (cf. Definition 8). Observe that there is a dashed line labelled with $\mathcal{F}$ and $\mathcal{B}[\mathcal{F}_B]$ from the DBA construction component to the CEX analysis component; this means that we will need $\mathcal{F}$ and $\mathcal{B}[\mathcal{F}_B]$ in the CEX analysis. The above procedure will continue until a correct DBA has been learned.

The main challenge here is that the DBA $\mathcal{B}[\mathcal{F}_B]$ is only guaranteed to be language-equivalent if $\mathcal{F}$ is in the canonical form of limit FDFAs (cf. Lemma 1); before that, it will accept a sub-language, i.e., UP($\mathcal{L}(\mathcal{B}[\mathcal{F}_B])$) $\subseteq$ UP($\mathcal{F}$). This is because $\mathcal{B}[\mathcal{F}_B]$ is obtained by first making all final states of $\mathcal{F}$ non-final, except for where a final state is a sink (and we thus refer to these states as sink final states; there need not exist one). However, a standard CEX for the limit FDFA learner to refine $\mathcal{F}$ needs to be in the symmetric difference between UP($\mathcal{F}$) and L, i.e., $u \cdot v^\omega \in$ UP($\mathcal{F}$) $\ominus$ UP(L), but we only have $uv^\omega \in L \ominus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$. As a consequence, the CEX returned for $\mathcal{B}[\mathcal{F}_B]$ from equivalence queries cannot always be directly used to refine the current conjectured FDFA $\mathcal{F}$.

We overcome this challenge by carefully categorising a CEX and then extracting a CEX for $\mathcal{F}$ from $uv^\omega \in L \ominus \mathcal{L}(\mathcal{B}[\mathcal{F}])$ accordingly, possibly with the help of a few membership queries (cf. Sect. 6). Since the intermediate FDFA $\mathcal{F}$ is not perfect, the CEX uv^ω from $\mathsf{EQ}^{\text{DBA}}(\mathcal{B}[\mathcal{F}_B])$ can fall into three categories, shown in Fig. 2: it can be (1) in the language of the conjectured DBA $\mathcal{B}[\mathcal{F}_B]$ (and thus of the FDFA $\mathcal{F}$), but not in the target language L, (2) in the target language L, but not in the language of the FDFA $\mathcal{F}$ (and thus not in the language of $\mathcal{B}[\mathcal{F}_B]$), and (3) in the target language L *and* the language of the FDFA $\mathcal{F}$, but not in the language of $\mathcal{B}[\mathcal{F}_B]$.

While the first two cases are standard (as they are in the symmetric difference between UP($\mathcal{F}$) and UP(L)), the third case poses an additional challenge in FDFA learning, as it is not the FDFA itself, but only the DBA constructed from it,

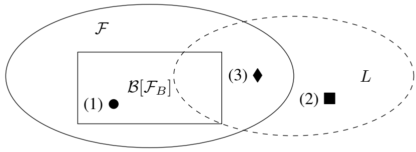


Figure 2: The different cases of counterexamples.

that does not accept the witness word provided by the oracle. We develop a translation that interprets the states of the leading and progress DFAs as their representative word from the rows of the observation table (cf. Fig. 4). The coverage of the third case is our key technical innovation.

We will describe each component of the DBA learner separately with more details in subsequent sections.

4 From Limit/Sink FDFAs to DBAs

We present our DBA construction component in this section. We will first recall the definitions of limit FDFAs as canonical FDFAs for ω-regular languages and then introduce the sink FDFAs we use to construct DBAs.

By Definition 1, the Myhill-Nerode theorem associates each equivalence class of $\backsim_R$ with a state of the minimal DFA $\mathcal{D}$ of the regular language R. The situation in ω-regular languages is, however, more involved. An immediate extension of such RCs for an ω-regular language L is the following.

Definition 4 (Leading RC). *For two $u_1, u_2 \in \Sigma^*$, $u_1 \backsim_L u_2$ if, and only if, $\forall w \in \Sigma^\omega .\ u_1 w \in L \iff u_2 w \in L$ holds.*

We then define the limit FDFAs for ω-regular languages.

Definition 5 (Limit FDFAs [Li *et al.*, 2023a]). *The leading RC $\backsim$ is as defined in Definition 4.*

Let $[u]_\backsim$ be an equivalence class of $\backsim$. For $x, y \in \Sigma^*$, we define limit RC as: $x \approx^u y$ if, and only if, $\forall v \in \Sigma^*$, $(u{\cdot}x{\cdot}v \backsim u \Longrightarrow u{\cdot}(x{\cdot}v)^\omega \in L) \iff (u{\cdot}y{\cdot}v \backsim u \Longrightarrow u{\cdot}(y{\cdot}v)^\omega \in L)$.

The limit FDFA $\mathcal{F}_L = (\mathcal{M}, \{\mathcal{N}_L^u\})$ of L uses the leading DFA $\mathcal{M} = (\mathcal{T}[\backsim], \emptyset)$ as defined in Definition 1; and, for each state $[u]_\backsim \in \Sigma^*/_\backsim$, the progress DFA $\mathcal{N}_L^u$ is the tuple $(\mathcal{T}[\approx_L^u], F_u)$, where $[v]_{\approx_L^u} \in F_u$ if $u \cdot v \backsim u \implies uv^\omega \in L$.

Intuitively, a word v is accepted by $\mathcal{N}_L^u$ if, when $\mathcal{M}$ makes a round trip from state $[u]_\backsim$ over v, we must have $uv^\omega \in L$. This means, in the case of DBAs, v is a word making the DBA of L visit some accepting transition from $[u]_\backsim$-states; so, if the DBA closes a loop over v, then uv^ω must belong to L. The limit RC $\approx^u$ is then naturally defined over the language $\{v \in \Sigma^* : u \cdot v \backsim u \Rightarrow uv^\omega \in L\}$, similarly to the RC $\backsim_R$ defined over a *regular* language R as given in Sect. 2. Limit FDFAs are the class of canonical FDFAs that is useful for the definition of the *sink FDFAs* we use for learning DBAs.

Definition 6 (Sink FDFAs [Li *et al.*, 2023a]). *The sink FDFA $\mathcal{F}_B = (\mathcal{M}, \{\mathcal{N}_B^u\})$ of L is defined so that the leading DFA $\mathcal{M}$ is as in Definition 5, and the TS of each $\mathcal{N}_B^u$ is, for each $[u]_\backsim \in \Sigma^*/_\backsim$, exactly as that of $\mathcal{N}_L^u$ from Definition 5.*

The set of final states F_u contains the equivalence classes $[x]_{\approx_L^u}$ such that, for all $v \in \Sigma^*$, $u{\cdot}xv \backsim u \Longrightarrow u{\cdot}(xv)^\omega \in L$.

While the definition says 'classes', F_u either contains a single state, which is a final sink in $\mathcal{N}_L^u$ (and $\mathcal{N}_B^u$), or is empty (if $\mathcal{N}_L^u$ does not have such a final sink) [Li *et al.*, 2023a]. A final state is said to be a *sink* if it has a self-loop over Σ.

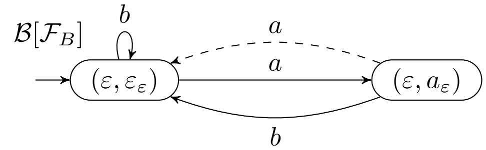


Figure 3: The DBA constructed from $\mathcal{F}_B$ in Fig. 4. The subscript ε indicates the progress states belong to the progress DFA $\mathcal{N}_B^\varepsilon$.

DBA construction. Upon receiving an FDFA $\mathcal{F}$ from $\mathsf{EQ}^{\mathrm{FDFA}}(\mathcal{F})$, which may *not* be in canonical form, we first obtain an FDFA $\mathcal{F}'_B$ by allowing only *final sinks* as final states and construct a DBA below. To make the DBA construction more general, we assume an FDFA $\mathcal{F}'_B = (\mathcal{M}, \{\mathcal{N}^q\}_{q \in Q})$ where $\mathcal{M} = (Q, \Sigma, \iota, \delta)$ and, for each $q \in Q$, we have $\mathcal{N}^q = (Q_q, \Sigma, \iota_q, \delta_q, F_q)$ where $\mathcal{F}_q$ only contains final sinks.

Definition 7 ([Bohn and Löding, 2022]). *Let $\mathcal{F}'_B = (\mathcal{M}, \{\mathcal{N}^q\}_{q \in Q})$ be the FDFA defined above. Let $\mathcal{T}[\mathcal{F}'_B]$ be the TS constructed from $\mathcal{F}'_B$ defined as the tuple $\mathcal{T}[\mathcal{F}'_B] = (Q_\mathcal{T}, \Sigma, \iota_\mathcal{T}, \delta_\mathcal{T})$ and $\Gamma \subseteq \{(q, \sigma) : q \in Q_\mathcal{T}, \sigma \in \Sigma\}$ be a set of transitions where*

- *$Q_\mathcal{T} := Q \times \bigcup_{q \in Q} Q_q$;*
- *$\iota_\mathcal{T} := (\iota, \iota_\iota)$;*
- *For a state $(m, q) \in Q_\mathcal{T}$ and $\sigma \in \Sigma$, let $q' = \delta_{\widetilde{m}}(q, \sigma)$ where $\mathcal{N}^{\widetilde{m}}$ is the progress DFA that q belongs to and let $m' = \delta(m, \sigma)$. Then*

$$\delta((m, q), \sigma) = \begin{cases} (m', q') & \text{if } q' \notin F_{\widetilde{m}} \\ (m', \iota_{m'}) & \text{if } q' \in F_{\widetilde{m}} \end{cases}$$

- *$((m, q), \sigma) \in \Gamma$ if $q' \in F_{\widetilde{m}}$*

An example DBA constructed from an FDFA is provided in Fig. 3. The sink FDFA $\mathcal{F}_B$ of L, as constructed in Definition 6, can be translated to its equivalent DBA.

Lemma 1 ([Li *et al.*, 2023a]). *If $\mathcal{F}'_B$ is an FDFA with only sink final states. Let $\mathcal{B}[\mathcal{F}'_B] = (\mathcal{T}[\mathcal{F}'_B], \Gamma)$ as given in Definition 7. Then, $UP(\mathcal{L}(\mathcal{B}[\mathcal{F}'_B])) \subseteq UP(\mathcal{F}'_B)$.*

Let $\mathcal{F}_B$ be the sink FDFA of a DBA language L, as defined in Definition 6. Let $\mathcal{B}[\mathcal{F}_B]$ be the DBA constructed by Definition 7 from $\mathcal{F}_B$. Then $UP(\mathcal{F}_B) = UP(L) = UP(\mathcal{L}(\mathcal{B}[\mathcal{F}_B]))$.

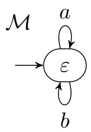
Recall that we learn DBAs by learning the limit FDFA $\mathcal{F}_L$. By Lemma 1, our DBA learner eventually learns the correct DBA when the conjectured FDFA converges to $\mathcal{F}_L$ in the worst case.

5 The Limit FDFA Learner

With the canonical form of limit FDFAs (cf. Definition 5), we can now describe the limit FDFA learner. In [Li *et al.*, 2023b, Appendix E], the authors gave a learning algorithm for limit FDFAs. We follow their description of the limit FDFA learner but allow a more *relaxed* form of counterexamples. For instance, they require the CEX (u, v) to be normalised with respect to the current leading DFA $\mathcal{M}$, while our requirements

$\mathcal{TB}$

	$(\varepsilon,\varepsilon)$
ε	$\bot$
a	$\bot$
b	$\bot$



$\mathcal{TB}_\varepsilon$

$\approx^\varepsilon$	ε	b
ε	$(\top,\bot)$	$(\top,\bot)$
a	$(\top,\top)$	$(\top,\bot)$
aa	$(\top,\top)$	$(\top,\top)$
b	$(\top,\bot)$	$(\top,\bot)$
ab	$(\top,\bot)$	$(\top,\bot)$
aaa	$(\top,\top)$	$(\top,\top)$
aab	$(\top,\top)$	$(\top,\top)$

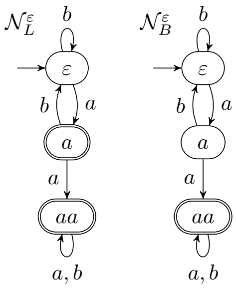


Figure 4: The observation tables for the limit FDFA $\mathcal{F}_L = (\mathcal{M}, \{\mathcal{N}_L^\varepsilon\})$ and the sink FDFA $\mathcal{F}_B = (\mathcal{M}, \{\mathcal{N}_B^\varepsilon\})$ of the DBA language $L = (\{a, b\}^* \cdot aa)^\omega$. Double circles denote final states.

in Definition 8 does not ask for it. The importance of our definition of counterexamples is that it allows to learn the canonical form of limit FDFAs, while theirs only learns an abstract form, which cannot be used to construct DBAs.

As usual, a learner uses an *observation table* [Angluin, 1987] defined as a tuple $\mathcal{TB} = (S, \tilde{S}, E, T)$, where S is a prefix-closed set of finite words, E is a set of experiments trying to distinguish the words in S, and $T : S \times E \to D$ stores the element (membership query results) in entry $T(s, e)$ an element in some domain D, where $s \in S$ and $e \in E$. For the limit FDFA, D is the set of Boolean values $\{\top, \bot\}$ for the leading DFA and a pair of Boolean values for progress DFAs (see Fig. 4). We determine when two words $s_1, s_2 \in S$ are *not* equivalent depending on the RC we are using. The component $\tilde{S} \subseteq S$ is the subset considered as representatives of the equivalence classes, i.e. the state names of the constructed DFA. Take $\mathcal{TB}_\varepsilon$ in Fig. 4 for example: $S = \{\varepsilon, a, aa, aaa, aab, ab, b\}$ (all row names), $\tilde{S} = \{\varepsilon, a, aa\}$ (upper row names), and $E = \{\varepsilon, b\}$ (all column names).

A table is *closed* if S is prefix-closed and, for every $s \in \tilde{S}$ and $\sigma \in \Sigma$, we have $s\sigma \in S$. The procedure *CloseTable* uses two sub-procedures $\mathtt{ENT}$ (read: *entry*) and $\mathtt{DFR}$ (read: difference) to make a given table closed. Here $\mathtt{ENT}(s, e)$ is used to fill the table entry $T(s, e)$ by means of asking membership queries. The procedure $\mathtt{DFR}$ is used to determine which rows (words) of the table should be distinguished.

A learning procedure usually begins by creating an initial observation table by asking membership queries, closing the table with $\mathtt{ENT}$ and $\mathtt{DFR}$ procedures, and then constructing a conjectured automaton for asking an equivalence query. The learner should be able to use the CEX to the equivalence query to find new experiments (columns) for discovering new equivalence classes.

We let $\mathsf{MQ}(x, y)$ be the result of the membership query to the UP-word $x \cdot y^\omega$ to the oracle. The procedures $\mathtt{ENT}_1$, $\mathtt{DFR}_1$ and $\mathtt{Aut}_1$ are used for learning the leading DFA. More precisely, for $u, x, y \in \Sigma^*$, $\mathtt{ENT}_1(u, (x, y)) = \mathsf{MQ}(u \cdot x, y)$; for two finite row words $u_1, u_2 \in S$, $\mathtt{DFR}_1(u_1, u_2) = \top$ iff there exists $(x, y) \in E$ such that $T(u_1, (x, y)) \neq T(u_2, (x, y))$. That is, we can use $x \cdot y^\omega$ to distinguish the finite words u_1 and u_2 according to $\backsim$.

The procedure $\mathtt{Aut}_1$ is simply to construct the leading DFA without final states from $\mathcal{TB}$, by Definition 1. Note that, when a leading DFA is updated, this affects the RC $\approx_L^u$, which in turn affects some of the progress DFAs that then need to be reconstructed. This is why in Algorithm 1 we reconstruct progress DFAs $\mathcal{N}^{\tilde{u}}$ for all $\tilde{u} \in \tilde{S}$ once $\mathcal{M}$ is updated.

Let $\mathcal{M}_u$ denote the DFA obtained from $\mathcal{M}$ by setting the initial state to u. In the table, if $u \in \tilde{S}$, we have $u = \mathcal{M}(u) = \mathcal{M}_u(\varepsilon)$. When learning progress DFAs, for $u, x, v \in \Sigma^*$, we define $\mathtt{ENT}_2^u(x, v) = (\mathcal{M}_u(x \cdot v) \stackrel{?}{=} u, \mathsf{MQ}(u, x \cdot v))$. We can also regard $\mathtt{ENT}_2^u(x, v)$ (and thus $T_u(x, v)$) as $\top$ (Boolean implication of the pairs) in testing equivalence if $\mathcal{M}_u(x \cdot v) \neq u$ or $\mathsf{MQ}(u, x \cdot v) = \top$ holds, corresponding to whether $ux \cdot v \backsim u \implies u \cdot (xv)^\omega \in L$ holds in Definition 5; for two finite row words, $x_1, x_2 \in S_u$, $\mathtt{DFR}_2^u(x_1, x_2)$ returns $\top$ if there exists $v \in E$ such that $T_u(x_1, v) \neq T_u(x_2, v)$. An example table $\mathcal{TB}_\varepsilon$ is depicted in Fig. 4.

The procedure $\mathtt{Aut}_u(\mathcal{TB}_u)$ not only constructs the TS but also sets a state x as final if $T_u(x, \varepsilon) = \top$. Note that here $T_u(x, v)$ is regarded as the result of whether or not $\big(u = \mathcal{M}_u(xv) \implies \mathsf{MQ}(u, xv)\big)$ holds.

We have described above how to fill the observation tables and construct DFAs. Now we show that, as long as the CEX returned for the limit FDFA learner satisfies Definition 8, it is good to refine the current conjecture $\mathcal{F}$. By analysing the CEX, we can add a new *column* e to the corresponding table in order to distinguish two rows $x \cdot a$ and x' that are currently classified as equivalent, where $x, x' \in \tilde{S}, x \cdot a \in S$ and $\mathtt{DFR}(x \cdot a, x') = \top$ with $\mathtt{DFR} \in \{\mathtt{DFR}_1^u, \mathtt{DFR}_2\}$.

In the remainder of the paper, we will regularly make use of the duality of the states in the DFAs and the words in the observation table they represent.

Definition 8. *Let* (u, v) *be a CEX to the conjectured FDFA* $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^x\})$. *We say* (u, v) *is* good for refinement *(GfR) of* $\mathcal{F}$ *if it has the* prefix *or* loop *property described below.*

Prefix. There exist two indices $0 \leq i < j \leq |u|$ *such that* $\mathsf{MQ}(x_i \cdot u[i \ldots], v) \neq \mathsf{MQ}(x_j \cdot u[j \ldots], v)$, *where* $x_i = \mathcal{M}(u[0 \cdots i])$ *and* $x_j = \mathcal{M}(u[0 \cdots j])$.

Loop. There exist two indices $0 \leq i < j \leq |v|$ *such that* $\tilde{u} = \mathcal{M}_{\tilde{u}}(y_i \cdot v[i \ldots]) \implies \mathsf{MQ}(\tilde{u}, y_i \cdot v[i \ldots])$ *and* $\tilde{u} = \mathcal{M}_{\tilde{u}}(y_j \cdot v[j \ldots]) \implies \mathsf{MQ}(\tilde{u}, y_j \cdot v[j \ldots])$ *are not equal, where* $\tilde{u} = \mathcal{M}(u), y_i = \mathcal{N}^{\tilde{u}}(v[0 \cdots i])$ *and* $y_j = \mathcal{N}^{\tilde{u}}(v[0 \cdots j])$.

The refinement procedure of the conjectured FDFA $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^x\})$ has been formalised as Alg. 1. First, we assume that the CEX (u, v) is GfR. Let $\tilde{u} = \mathcal{M}(u)$. If (u, v) is a prefix CEX, the leading DFA $\mathcal{M}$ will be refined. Otherwise, if (u, v) is a loop CEX, the progress DFA $\mathcal{N}^{\tilde{u}}$ will be refined.

Refinement of $\mathcal{M}$. Since $\mathsf{MQ}(x_i \cdot u[i \ldots], v) \neq \mathsf{MQ}(x_j \cdot u[j \ldots], v)$, we have $x_i \cdot u[i \ldots] \not\backsim x_j \cdot u[j \ldots]$. We can find an experiment as follows. Let $x_k = \mathcal{M}(u[0 \cdots k])$ be the state or word representative that $\mathcal{M}$ arrives at after reading the first k letters of u. In particular, $x_i = \mathcal{M}(u[0 \cdots i])$ and $x_j = \mathcal{M}(u[0 \cdots j])$. We construct the sequence by asking membership queries: $\mathsf{MQ}(x_0 \cdot u[0 \ldots], v), \cdots, \mathsf{MQ}(x_i \cdot u[i \ldots], v), \cdots, \mathsf{MQ}(x_j \cdot u[j \ldots], v), \cdots$. Since $\mathsf{MQ}(x_i \cdot u[i \ldots], v) \neq \mathsf{MQ}(x_j \cdot u[j \ldots], v)$ by prefix assumption, this sequence has different results at the indices i and j.

Algorithm 1: Refinement of the conjecture FDFA $\mathcal{F}$

Input: An FDFA $\mathcal{F} = (\mathcal{M}, \{\mathcal{N}^x\})$.
Let (u, v) be GfR for $\mathcal{F}$ and let $\tilde{u} = \mathcal{M}(u)$;
if *(u, v) is a prefix CEX* **then**
  $E = E \cup$ *FindDistinguishingExperiment*(u, v);
  CloseTable$(\mathcal{TB}, \mathtt{ENT}_1, \mathtt{DFR}_1)$ and let $\mathcal{M} = \mathtt{Aut}_1(\mathcal{TB})$;
  forall $\tilde{u} \in \tilde{S}$ **do**
    CloseTable$(\mathcal{TB}_{\tilde{u}}, \mathtt{ENT}_2^{\tilde{u}}, \mathtt{DFR}_2^{\tilde{u}})$ and let $\mathcal{N}^{\tilde{u}} = Aut_2(\mathcal{TB}_{\tilde{u}})$;
else if *(u, v) is a loop CEX* **then**
  $E_{\tilde{u}} = E_{\tilde{u}} \cup$ *FindDistinguishingExperiment*$(\tilde{u}, v)$;
  CloseTable$(\mathcal{TB}_{\tilde{u}}, \mathtt{ENT}_2^{\tilde{u}}, \mathtt{DFR}_2^{\tilde{u}})$ and let $\mathcal{N}^{\tilde{u}} = Aut_2(\mathcal{TB}_{\tilde{u}})$;

Therefore, there must exist the smallest $k \in [i, j)$ such that $\mathsf{MQ}(x_k \cdot u[k] \cdot u[k+1 \ldots], v) \neq \mathsf{MQ}(x_{k+1} \cdot u[k+1 \ldots], v)$, Hence, since $x_{k+1} = \mathcal{M}_{x_k}(u[k])$, we can use the experiment $e = (u[k+1 \ldots], v)$ to distinguish $x_k \cdot u[k]$ and x_{k+1}.

Refinement of $\mathcal{N}^{\tilde{u}}$. Let $y_k = \mathcal{N}^{\tilde{u}}(v[0 \cdots k])$. Similarly, we have a sequence $(m_0, c_0), \cdots, (m_i, c_i), \cdots, (m_j, c_j)$ where $m_k = \top$ iff $\tilde{u} = \mathcal{M}_{\tilde{u}}(y_k \cdot v[k \ldots])$ and $c_k = \top$ iff $\tilde{u} \cdot (y_k \cdot v[k \ldots])^\omega \in L$ (i.e. $c_k = \mathsf{MQ}(\tilde{u}, y_k \cdot v[k \ldots])$).

Since (u, v) is a loop CEX, only one of $m_i \implies c_i$ and $m_j \implies c_j$ holds. There must be a smallest integer $k \in [i, j)$ such that $m_k \implies c_k$ and $m_{k+1} \implies c_{k+1}$ differ. Assume $m_k \implies c_k$ holds (the other case is entirely similar). Thus, $m_{k+1} \implies c_{k+1}$ does not hold. Analogously, we can add the experiment $e = v[k+1 \ldots]$ to distinguish $y_k \cdot v[k]$ and y_{k+1} since we have $\tilde{u} = \mathcal{M}_{\tilde{u}}(y_k \cdot v[k \ldots]) \implies \tilde{u} \cdot (y_k \cdot v[k \ldots])^\omega \in L$ holds but $\tilde{u} = \mathcal{M}_{\tilde{u}}(y_{k+1} \cdot v[k+1 \ldots]) \implies \tilde{u} \cdot (y_{k+1} \cdot v[k+1 \ldots])^\omega \in L$ does not hold.

It immediately follows that the limit FDFA learner is guaranteed to make progress once receiving a *GfR* CEX.

Lemma 2. *A CEX (u, v) satisfying Definition 8 refines the current leading DFA or a progress DFA in Algorithm 1.*

6 CEX Analysis Component

Now we describe the CEX analysis component. By assumption, the input here is a UP-word $w = uv^\omega \in \mathcal{L}(\mathcal{B}[\mathcal{F}_B]) \ominus L$, represented by its decomposition (u, v) (cf. Fig. 1).

Recall that we have the following three cases about w in Fig. 2: (1) $w \in \mathcal{L}(\mathcal{B}[\mathcal{F}_B]) \setminus L$ and $w \in \mathrm{UP}(\mathcal{F})$ since $\mathrm{UP}(\mathcal{L}(\mathcal{B}[\mathcal{F}_B])) \subseteq \mathrm{UP}(\mathcal{F})$, (2) $w \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$ and $w \notin \mathrm{UP}(\mathcal{F})$, and (3) $w \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$ and $w \in \mathrm{UP}(\mathcal{F})$.

We first analyse Case (1) and Case (2), which are already in the symmetric difference between $\mathrm{UP}(\mathcal{F})$ and $\mathrm{UP}(L)$. This means that the CEX is easy and we only need to extract a *normalised* decomposition (u', v') from w as below.

(1) $w \in \mathcal{L}(\mathcal{B}[\mathcal{F}_B]) \setminus L$ and $w \in \mathrm{UP}(\mathcal{L}(\mathcal{B}[\mathcal{F}_B])) \subseteq \mathrm{UP}(\mathcal{F})$. Hence, $w \in \mathrm{UP}(\mathcal{F})$ but $w \notin L$. There must be a normalised decomposition (u', v') of w such that (u', v') is accepted by $\mathcal{F}$. However, w is not in L, so (u', v') should actually have been rejected. We can just return (u', v') as a CEX to further refine $\mathcal{F}$. We now prove that (u', v') satisfies Definition 8.

First, let $x = \mathcal{M}(u')$. We ask $\mathsf{MQ}(x, v') \stackrel{?}{=} \mathsf{MQ}(u', v')$. If their results are not equal, we let $i = 0$ and $j = |u'|$. We can then verify that (u', v') satisfies the prefix requirement. Otherwise their membership results agree. We let $i = 0$ and $j = |v'|$. Hence, $y_i = v'[0 \cdots 0] = \varepsilon$ and $y_j = \mathcal{N}^x(v')$. Since (u', v') is accepted by $\mathcal{F}$, we have $x = \mathcal{M}_x(y_j \cdot \varepsilon) \Longrightarrow \mathsf{MQ}(x, y_j \cdot \varepsilon)$ since y_j is a final state in $\mathcal{N}^x$. However, $x = \mathcal{M}(u') = \mathcal{M}_x(y_i \cdot v')$ because (u', v') is normalised. Together with $\mathsf{MQ}(x, v') = \mathsf{MQ}(u', v') = \bot$, $x = \mathcal{M}_x(y_i) \Longrightarrow \mathsf{MQ}(x, y_i \cdot v'[0 \ldots])$ does not hold. Hence, (u', v') satisfies the loop requirement. Therefore, (u', v') is GfR.

(2) $w \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$ and $w \notin \mathrm{UP}(\mathcal{F})$. Consequently, $w \notin \mathrm{UP}(\mathcal{F})$ and $w \in L$. There must be a normalised decomposition (u', v') of w such that (u', v') is not accepted by $\mathcal{F}$. However, w is in L, so (u', v') should have been accepted. Similarly, we can return (u', v') as a CEX to refine $\mathcal{F}$. Again, we can similarly prove that (u', v') is GfR as Case (1) and we refer to Appendix A for the details.

We only proved the existence of such counterexamples. We refer to [Li *et al.*, 2021] for details about how to extract them.

Note that the first two cases do not make any specific reference to the difference between $\mathrm{UP}(\mathcal{F})$ and $\mathcal{L}(\mathcal{B}[\mathcal{F}_B])$—they are a variation of vanilla FDFA learning. The third case, however, is quite different: the CEX (u, v) is such that $w = uv^\omega \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$ and $w \in \mathrm{UP}(\mathcal{F})$—and not in the symmetric difference of $\mathrm{UP}(\mathcal{F})$ and $\mathrm{UP}(L)$ (cf. Figure 2).

To tackle the CEX analysis in this case, the structure of the DBA $\mathcal{B}[\mathcal{F}_B]$ plays a crucial role. This seems unavoidable, because we have $w \in L$ *and* $w \in \mathrm{UP}(\mathcal{F})$, so that the quest for a normalised decomposition (u', v') of w such that (u', v') is not accepted by $\mathcal{F}$, as we did in case (2), cannot work. This makes case (3) significantly more involved. We will analyse the CEX w by looking carefully at the run of $\mathcal{B}[\mathcal{F}_B]$ over $w = uv^\omega \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$.

Let $\rho = (m_0, \iota_0)(m_1, \iota_1) \cdots$ be the run of $\mathcal{B}[\mathcal{F}_B]$ over w. By assumption, w is *not* accepted by $\mathcal{B}[\mathcal{F}_B]$. So, the sequence of progress DFA states in the run ρ will eventually get stuck in a progress DFA according to Definition 7. Assume that ρ eventually gets stuck in the progress DFA $\mathcal{N}^m$, where m is a state of the leading DFA, and thus also a word representative of that equivalence class. Let $\hat{\rho}$ be the projection on the first element of each pair in ρ. We can see that $\hat{\rho}$ is the run of the leading DFA $\mathcal{M}$ over w.

Since $\mathcal{B}[\mathcal{F}_B]$ has a finite number of states and w is a UP-word, we can decompose w into three finite words $x, v_1 \in \Sigma^*, v_2 \in \Sigma^+$ such that $w = x \cdot v_1 \cdot (v_2)^\omega, m = \mathcal{M}(x), m' = \mathcal{M}(xv_1) = \mathcal{M}(xv_1 \cdot v_2), \mathcal{N}^m(v_1) = \mathcal{N}^m(v_1 \cdot v_2)$, where m' is a leading state that might be different to m. Let $\widetilde{v}_1 = \mathcal{N}^m(v_1)$. Hence, $\widetilde{v}_1 = \mathcal{N}^m(v_1 \cdot v_2)$ holds as well. We can depict the run ρ as follows:

$$\rho := (\iota, \iota_\iota) \xrightarrow{x} (m, \iota_m) \xrightarrow{v_1} (m', \widetilde{v}_1) \xrightarrow{v_2} (m', \widetilde{v}_1) \qquad (1)$$

Next, we find a word $y \in \Sigma^*$ from the observation table for $\mathcal{N}^m$ such that $m = \mathcal{M}_m(\widetilde{v}_1 \cdot y)$ and $m \cdot (\widetilde{v}_1 \cdot y)^\omega \notin L$. To see that such a word exists we assume for contradiction that there is no such word, and thus no entry $(\top, \bot)$ in the row of the observation table for $\widetilde{v_1}$. But then $\widetilde{v_1}$ is the sink final state, which contradicts that $\mathcal{B}[\mathcal{F}_B]$ got stuck in $\mathcal{N}^m$.

With this word y, we can extract the CEX (u', v') by analysing the following three cases.

(3a) $m \neq \mathcal{M}_m(v_1 \cdot y)$. By Definition 5, this entails that y can be used to distinguish $\widetilde{v_1}$ and v_1 but currently $\widetilde{v_1}$ and v_1 are classified as equivalent since $\widetilde{v_1} = \mathcal{N}^m(v_1)$. We can thus choose the loop CEX $(u', v') = (x, v_1 \cdot y)$ to refine $\mathcal{N}^m$. One can verify that $(x, v_1 \cdot y)$ is a valid GfR loop CEX by setting the indices $i = 0$ and $j = |v_1|$ in Definition 8. Note that since $m = \mathcal{M}(x)$, so we use (u', v') to refine $\mathcal{N}^m$.

(3b) $m = \mathcal{M}_m(v_1 \cdot y)$ and $m \cdot (v_1 \cdot y)^\omega \in L$ (tested by $\mathsf{MQ}(m, v_1 \cdot y)$). We can again choose the loop CEX $(u', v') = (x, v_1 \cdot y)$ to refine $\mathcal{N}^m$ since $\widetilde{v_1}$ and v_1 can be distinguished with y. One can verify that $(x, v_1 \cdot y)$ is a valid GfR loop CEX by again setting $i = 0$ and $j = |v_1|$ in Definition 8.

(3c) The remaining case $m = \mathcal{M}_m(v_1 \cdot y)$ and $m \cdot (v_1 \cdot y)^\omega \notin L$ (tested by $\mathsf{MQ}(m, v_1 \cdot y)$) is quite involved, so we dedicate the remainder of this section to it. The analysis method is provided as Algorithm 2.

Algorithm 2: Counterexample generation for Case (3c): $m = \mathcal{M}_m(v_1 \cdot y)$ and $m \cdot (v_1 \cdot y)^\omega \notin L$

Input: $m, x, v_1, y \in \Sigma^*$ and $v_2 \in \Sigma^+$
Output: A GfR counterexample
if $\mathsf{MQ}(m \cdot v_1, v_2) = \bot$ **then**
  return $(x \cdot v_1, v_2)$ as a prefix CEX;
$k := 0$;
while *true* **do**
  $h := 1$;
  while $h \leq k$ **do**
    if $\mathsf{MQ}(m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2) = \bot$ **then**
      return $(x \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2)$ as a prefix CEX;
    $h := h + 1$;
  if $\mathsf{MQ}(m, v_1 \cdot v_2^k \cdot y) = \top$ **then**
    return $(x, v_1 \cdot v_2^k \cdot y)$ as a loop CEX;
  $k := k + 1$;

First, if $m \cdot v_1 \cdot {v_2}^\omega \notin L$, then $v_1 \cdot {v_2}^\omega$ (tested by $\mathsf{MQ}(m, v_1 \cdot y)$) distinguishes x from m, so that we can return the prefix CEX $(x \cdot v_1, v_2)$. One can verify $(x \cdot v_1, v_2)$ by setting $i = 0$ and $j = |x|$ in Definition 8.

We also observe that the prefix CEX and loop CEX we return in the loop/s are GfR counterexamples, as they establish that: (1) $m \not\backsim x \cdot (v_1 \cdot v_2^k \cdot y)^h$ although $m = \mathcal{M}(x \cdot (v_1 \cdot v_2^k \cdot y)^h) = \mathcal{M}_m((v_1 \cdot v_2^k \cdot y)^h)$ (because $m' = \mathcal{M}(x \cdot v_1) = \mathcal{M}_m(v_1) = \mathcal{M}_m(v_1 \cdot v_2^k)$ by decomposition of ρ); and (2) $\widetilde{v_1} \not\approx_L^m v_1 \cdot v_2^k$ although $\widetilde{v_1} = \mathcal{N}^m(v_1) = \mathcal{N}^m(v_1 \cdot v_2^k)$. To prove (1), we first observe that Alg. 2 did not return before while loop, hence $m \cdot v_1 \cdot v_2^\omega \in L$. Moreover, by return condition of inner loop, we have that $m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1 \cdot v_2^\omega \notin L$. It follows that m and $m \cdot (v_1 \cdot v_2^k \cdot y)^h$ can be distinguished by $v_1 \cdot v_2^\omega$. We refer to Appendix A for the proof why the returned prefix CEX is GfR. To prove (2), we first have $m \cdot (v_1 \cdot y)^\omega \notin L$ and $m = \mathcal{M}_m(v_1 \cdot y)$ by assumption. By return condition of the outer loop, we have $m \cdot (v_1 \cdot v_2^k \cdot y)^\omega \in L$. Therefore, it follows that $\widetilde{v_1}$ and $v_1 \cdot v_2^k$ can be distinguished with y by Definition 5. Again, to obtain a valid GfR loop CEX, we return $(x, v_1 \cdot v_2^k \cdot y)$ since $m = \mathcal{M}(x)$. One can verify the returned CEX by setting $i = 0$ and $j = |v_1 \cdot v_2^k|$ in Definition 8.

Now we prove that Alg. 2 terminates. Let us assume that our algorithm does not terminate. For this, we argue towards contradiction that L is recognised by a DBA $\mathcal{D}$ with transition function δ and d states. Once we have completed the outer loop for $k = d+1$, we then know that, for all $h \leq k$, we have that $m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1 \cdot {v_2}^\omega \in L$. (Otherwise the inner loop will eventually return a prefix CEX, which leads to a contradiction.) Let us denote $x_h = \delta(\iota, m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1)$, then the run of $\mathcal{D}_{x_h}$ on v_2^k contains an accepting transition.

We now consider the run of $\mathcal{D}$ on $m \cdot (v_1 \cdot v_2^k \cdot y)^\omega$. We have established that it passes an accepting transition while traversing each of the first k '${v_2}^k$' sequences. Moreover, it cannot be on k different states (as there are only $d = k - 1$ different ones) after the first k iterations of the loop-part '$v_1 \cdot v_2^k \cdot y$', so that the run ends in an accepting loop. This entails $m \cdot (v_1 \cdot v_2^k \cdot y)^\omega \in L$, and we would return a loop CEX, which provides a contradiction and completes the proof.

Therefore, Lemma 3 follows immediately.

Lemma 3. *Algorithm 2 terminates and returns a valid GfR counterexample.*

7 Concluding Remarks

By putting all three components together, we have completed the design of our DBA learner. Theorem 1 follows directly from Lemmas 1, 2 and 3 since in the worst case, the algorithm terminates when the canonical limit FDFA has been learned.

Theorem 1. *Our DBA learner depicted in Fig. 1 terminates and learns a correct DBA of L.*

We remark that all operations individually—and thus our DBA learner as a whole—run in polynomial time with respect to the sizes of the limit FDFA $\mathcal{F}_L$ and the minimal DBA $\mathcal{B}$ of the target language L. Moreover, as mentioned in Section 1, we can easily obtain a DCA learner by learning the complement language of a target co-Büchi language.

The biggest advantage we reap with our DBA learner over other learning algorithms for ω-automata is perhaps that we not only obtain easy resolution of equivalence queries, but also maintain reasonable expressiveness for the learned languages. Our contribution will further advance the frontier of the applications of learning algorithms in various fields, including verification, testing, and modelling, as well as further applications mentioned in the introduction.

Acknowledgements. We thank the anonymous reviewers for their valuable feedback. This work has been supported in part by the NSFC grant 62102407 and the EPSRC through grants EP/X021513/1 and EP/X017796/1.

References

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

[Angluin *et al.*, 2015] Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In Qiang Yang and Michael J. Wooldridge, editors, *IJCAI*, pages 3308–3314. AAAI Press, 2015.

[Angluin *et al.*, 2018] Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as Acceptors of ω-Regular Languages. *Logical Methods in Computer Science*, 14(1), 2018.

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

[Bohn and Löding, 2022] León Bohn and Christof Löding. Passive learning of deterministic Büchi automata by combinations of DFAs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP*, volume 229 of *LIPIcs*, pages 114:1–114:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Bollig *et al.*, 2009] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1004–1009, 2009.

[Büchi, 1962] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[Calbrix *et al.*, 1993] Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *MFPS*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993.

[Cobleigh *et al.*, 2003] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[de Ruiter and Poll, 2015] Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX*, pages 193–206. USENIX Association, 2015.

[Farzan *et al.*, 2008] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008.

[Isberner *et al.*, 2014] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2014.

[Li *et al.*, 2021] Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for Büchi automata based on family of dfas and classification trees. *Inf. Comput.*, 281:104678, 2021.

[Li *et al.*, 2023a] Yong Li, Sven Schewe, and Qiyi Tang. A novel family of finite automata for recognizing and learning ω-regular languages. In Étienne André and Jun Sun, editors, *ATVA*, volume 14215 of *Lecture Notes in Computer Science*, pages 53–73. Springer, 2023.

[Li *et al.*, 2023b] Yong Li, Sven Schewe, and Qiyi Tang. A novel family of finite automata for recognizing and learning ω-regular languages. *CoRR*, abs/2307.07490, 2023.

[Maler and Pnueli, 1995] Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995.

[Michaliszyn and Otop, 2022] Jakub Michaliszyn and Jan Otop. Learning infinite-word automata with loop-index queries. *Artif. Intell.*, 307:103710, 2022.

[Myhill, 1957] John Myhill. Finite automata and the representation of events. In *Technical Report WADD TR-57-624*, page 112–137, 1957.

[Nerode, 1958] Anil Nerode. Linear automaton transformations. In *American Mathematical Society*, page 541–544, 1958.

[Vaandrager *et al.*, 2022] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *TACAS*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022.

[Weiss *et al.*, 2018] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018.

A Proof for GfR CEX

A.1 Case (2)

Recall that $w \in L \setminus \mathcal{L}(\mathcal{B}[\mathcal{F}_B])$ and $w \notin \mathrm{UP}(\mathcal{F})$. We return a normalised decomposition (u', v') of w as a CEX to refine $\mathcal{F}$. Now we show that (u', v') is GfR based on the fact that (u', v') is not accepted by $\mathcal{F}$ but $u' \cdot v'^{\omega} \in L$.

Let $x = \mathcal{M}(u')$. We ask $\mathsf{MQ}(x, v') \stackrel{?}{=} \mathsf{MQ}(u', v')$. If the membership results are not equivalent, we can analogously prove that (u', v') satisfies the prefix requirement as in Case (1). Assume that their membership results agree. We then let $i = 0$ and $j = |v'|$. Hence, $y_i = v'[0 \cdots 0] = \varepsilon$ and $y_j = \mathcal{N}^x(v')$. Since (u', v') is normalised and not accepted by $\mathcal{F}$, we have that $x = \mathcal{M}(u') = \mathcal{M}(u' \cdot y_i)$. Together with $\mathsf{MQ}(x, v') = \mathsf{MQ}(u', v') = \top$, $x = \mathcal{M}(x \cdot y_i) \implies \mathsf{MQ}(x, y_i \cdot v'[1 \cdots])$ indeed holds, while $x = \mathcal{M}_x(y_j) \implies \mathsf{MQ}(x, y_j \cdot \varepsilon)$ must not hold due to the fact that y_j is not a final state in $\mathcal{N}^x$. Therefore, (u', v') satisfies the loop requirement and (u', v') is GfR.

A.2 Cases (3a) and (3b)

We first provide the proof for case (3a). Recall that $m \neq \mathcal{M}_m(v_1 \cdot y)$, $\widetilde{v_1} = \mathcal{N}^m(v_1)$, $m = \mathcal{M}_m(\widetilde{v_1} \cdot y)$ and $m \cdot (\widetilde{v_1} \cdot y)^{\omega} \notin L$. Recall that $m = \mathcal{M}(x)$.

The returned CEX is $(u', v') = (x, v_1 \cdot y)$. We now prove that it is a loop GfR CEX. We set the indices $i = 0$ and $j = |v_1|$ in Definition 8. It follows that $y_i = \mathcal{N}^m(v_1[0 \ldots i]) = \mathcal{N}^m(\varepsilon) = \varepsilon$ and $y_j = \mathcal{N}^m(v_1[0 \ldots]j) = \widetilde{v_1}$. Hence, we have $m = \mathcal{M}_m(y_i \cdot v_1 \cdot y) \implies m \cdot (y_i \cdot v_1 \cdot y)^{\omega} \in L$ hold since $m \neq \mathcal{M}_m(v_1 \cdot y)$ by assumption of Case (3a) and $y_i = \varepsilon$. However, $m = \mathcal{M}_m(y_j \cdot v_1[j \ldots] \cdot y) \implies m \cdot (y_j \cdot v_1[j \ldots] \cdot y)^{\omega} \in L$ does not hold since $v_1[0 \ldots j] = y_j$ (and thus $v_1[j \ldots] = \varepsilon$) and $y_j = \widetilde{v_1}$. Therefore, $(x, v_1 \cdot y)$ is a valid loop GfR CEX according to Definition 8.

For the proof of Case (3b), the proof is entirely similar and thus omitted here.

A.3 Case (3c)

We also observe that the prefix CEX and loop CEX we return in the loop/s are GfR counterexamples, as they establish that: (1) $m \not\sim x \cdot (v_1 \cdot v_2^k \cdot y)^h$ although $m = \mathcal{M}(x \cdot (v_1 \cdot v_2^k \cdot y)^h) = \mathcal{M}_m((v_1 \cdot v_2^k \cdot y)^h)$ (because $m' = \mathcal{M}(x \cdot v_1) = \mathcal{M}_m(v_1) = \mathcal{M}_m(v_1 \cdot v_2^k)$ by decomposition of ρ); and (2) $\widetilde{v_1} \not\approx_L^m v_1 \cdot v_2^k$ although $\widetilde{v_1} = \mathcal{N}^m(v_1) = \mathcal{N}^m(v_1 \cdot v_2^k)$. To prove (1), we first observe that Algorithm 2 did not return before while loop, hence $m \cdot v_1 \cdot v_2^{\omega} \in L$. Moreover, by return condition of inner loop, we have that $m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1 \cdot v_2^{\omega} \notin L$. It follows that m and $m \cdot (v_1 \cdot v_2^k \cdot y)^h$ can be distinguished by $v_1 \cdot v_2^{\omega}$. To obtain a valid GfR prefix counterexample, we return $(x \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2)$, so one can verify it by setting $i = |x|$ and $j = |x \cdot (v_1 \cdot v_2^k \cdot y)^h|$ in Definition 8. Hence by applying Definition 8, we have that $x_i = \mathcal{M}(x) = m = x_j = \mathcal{M}(x \cdot (v_1 \cdot v_2^k \cdot y)^h))$. It follows that $\mathsf{MQ}(x_i \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2) = \mathsf{MQ}(m \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2) = \bot$ while $\mathsf{MQ}(x_j \cdot v_1, v_2) = \mathsf{MQ}(m \cdot v_1, v_2) = \top$. This concludes that $(x \cdot (v_1 \cdot v_2^k \cdot y)^h \cdot v_1, v_2)$ is a valid GfR prefix counterexample. To prove (2), we first have $m \cdot (v_1 \cdot y)^{\omega} \notin L$ and $m = \mathcal{M}_m(v_1 \cdot y)$ by assumption. By return condition of the outer loop, we have $m \cdot (v_1 \cdot v_2^{\omega} \cdot y)^{\omega} \in L$. Therefore, it follows that $\widetilde{v_1}$ and $v_1 \cdot v_2^k$ can be distinguished with y by Definition 5. Again, to obtain a valid GfR loop CEX, we return $(x, v_1 \cdot v_2^k \cdot y)$ since $m = \mathcal{M}(x)$. One can verify the returned CEX by setting $i = 0$ and $j = |v_1 \cdot v_2^k|$ in Definition 8. By Definition 8, we have that $y_i = \varepsilon$ and $y_j = \mathcal{N}^m((v_1 \cdot v_2^k \cdot y)[0 \cdots j] = \mathcal{N}^m(v_1 \cdot v_2^k) = \widetilde{v_1}$. For y_i, we have $m = \mathcal{M}_m(y_i \cdot v_1 \cdot v_2^k \cdot y) \implies m \cdot (y_i \cdot v_1 \cdot v_2^k \cdot y) \in L$ since $m \cdot (v_1 \cdot v_2^k \cdot y)^{\omega} \in L$ by return condition. For y_j, $m = \mathcal{M}_m(y_j \cdot (v_1 \cdot v_2^k \cdot y)[j \ldots]) \implies m \cdot (y_j \cdot (v_1 \cdot v_2^k \cdot y)[j \ldots]) \in L$ (equivalently $m = \mathcal{M}_m(\widetilde{v_1} \cdot y) \implies m \cdot (\widetilde{v_1} \cdot y) \in L$) does not hold since $(v_1 \cdot v_2^k \cdot y)[j \ldots] = y$ and $y_j = \widetilde{v_1}$. Therefore, the CEX $(x, v_1 \cdot v_2^k \cdot y)$ satisfies the loop requirement of Definition 8 and thus a valid loop GfR CEX.