

A Novel Learning Algorithm for Büchi Automata based on Family of DFAs and Classification Trees

Yong Li^{1,2}, Yu-Fang Chen³, Lijun Zhang^{1,2}, Depeng Liu^{1,2}

¹ State Key Laboratory of Computer Science, Institute of Software, CAS

² University of Chinese Academy of Sciences

³ Institute of Information Science, Academia Sinica

Abstract. In this paper, we propose a novel algorithm to learn a Büchi automaton from a teacher who knows an ω -regular language. The algorithm is based on learning a formalism named *family of DFAs* (FDFAs) recently proposed by Angluin and Fisman [10]. The main catch is that we use a *classification tree* structure instead of the standard *observation table* structure. The worst case storage space required by our algorithm is quadratically better than the table-based algorithm proposed in [10]. We implement the first publicly available library ROLL (Regular Omega Language Learning), which consists of all ω -regular learning algorithms available in the literature and the new algorithms proposed in this paper. Experimental results show that our tree-based algorithms have the best performance among others regarding the number of solved learning tasks.

1 Introduction

Since the last decade, learning-based automata inference techniques [7, 11, 30, 35] have received significant attention from the community of formal system analysis. In general, the primary applications of automata learning in the community can be categorized into two: *improving efficiency and scalability of verification* [6, 15, 17, 19, 21, 23, 25, 32] and *synthesizing abstract system model for further analysis* [1, 5, 16, 18, 22, 24, 26, 34, 36, 39].

The former usually is based on the so called *assume-guarantee* compositional verification approach, which divides a verification task into several subtasks via a composition rule. Learning algorithms are applied to construct environmental assumptions of components in the rule automatically. For the latter, automata learning has been used to automatically generate interface model of computer programs [5, 22, 26, 36, 40], a model of system error traces for diagnosis purpose [16], behavior model of programs for statistical program analysis [18], and model-based testing and verification [24, 34, 39].

Besides the classical finite automata learning algorithms, people also apply and develop learning algorithm for richer models for the above two applications. For example, learning algorithms for register automata [27, 28] have been developed and applied to synthesis system and program interface models. Learning algorithm for timed automata has been developed for automated compositional verification for timed systems [32]. However, all the results mentioned above are for checking *safety properties* or synthesizing *finite behavior models* of systems/programs. Büchi automaton is the standard model for describing liveness properties of distributed systems [4]. The model has been

applied in automata theoretical model checking [38] to describe the property to be verified. It is also often used in the synthesis of reactive systems. Moreover, Büchi automata have been used as a means to prove program termination [31]. However, unlike the case for finite automata learning, learning algorithms for Büchi automata are very rarely used in our community. We believe this is a potentially fertile area for further investigation.

The first learning algorithm for the full-class of ω -regular languages represented as Büchi automata was described in [20], based on the L^* algorithm [7] and the result of [14]. Recently, Angluin and Fisman propose a new learning algorithm for ω -regular languages [10] using a formalism called a *family of DFAs* (FDFAs), based on the results of [33]. The main problem of applying their algorithm in verification and synthesis is that their algorithm requires a teacher for FDFAs. In this paper, we show that their algorithm can be adapted to support Büchi automata teachers.

We propose a novel ω -regular learning algorithm based on FDFAs and a *classification tree* structure (inspired by the tree-based L^* algorithm in [30]). The worst case storage space required by our algorithm is quadratically better than the table-based algorithm proposed in [10]. Experimental results show that our tree-based algorithms have the best performance among others regarding the number of solved learning tasks.

For regular language learning, there are robust and publicly available libraries, e.g., libalf [12] and LearnLib [29]. A similar library is still lacking for Büchi automata learning. We implement the first publicly available Büchi automata learning library, named ROLL (Regular Omega Language Learning, <http://iscasmc.ios.ac.cn/roll>), which includes all Büchi automata learning algorithms of the full class of ω -regular languages available in the literature and the ones proposed in this paper. We compare the performance of those algorithms using a benchmark consists of 295 Büchi automata corresponding to all 295 LTL specifications available in BüchiStore [37].

To summarize, our contribution includes the following. (1) Adapting the algorithm of [10] to support Büchi automata teachers. (2) A novel learning algorithm for ω -regular language based on FDFAs and classification trees. (3) The publicly available library ROLL that includes all Büchi automata learning algorithms can be found in the literature. (4) A comprehensive empirical evaluation of Büchi automata learning algorithms.

2 Preliminaries

Let A and B be two sets. We use $A \oplus B$ to denote their *symmetric difference*, i.e., the set $(A \setminus B) \cup (B \setminus A)$. Let Σ be a finite set called *alphabet*. We use ϵ to represent an empty word. The set of all finite words is denoted by Σ^* , and the set of all infinite words, called ω -words, is denoted by Σ^ω . Moreover, we also denote by Σ^+ the set $\Sigma^* \setminus \{\epsilon\}$. We use $|u|$ to denote the length of the finite word u . We use $[i \cdots j]$ to denote the set $\{i, i+1, \dots, j\}$. We denote by $w[i]$ the i -th letter of a word w . We use $w[i..k]$ to denote the subword of w starting at the i -th letter and ending at the k -th letter, inclusive, when $i \leq k$ and the empty word ϵ when $i > k$. A *language* is a subset of Σ^* and an ω -*language* is a subset of Σ^ω . Words of the form uv^ω are called *ultimately periodic* words. We use a pair of finite words (u, v) to denote the ultimately periodic word $w = uv^\omega$. We also call (u, v) a *decomposition* of w . For an ω -language L , let $UP(L) = \{uv^\omega \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$, i.e., all ultimately periodic words in L .

A *finite automaton* (FA) is a tuple $A = (\Sigma, Q, q_0, F, \delta)$ consisting of a finite alphabet Σ , a finite set Q of states, an initial state q_0 , a set $F \subseteq Q$ of accepting states, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. For convenience, we also use $\delta(q, a)$ to denote the set $\{q' \mid (q, a, q') \in \delta\}$. A *run* of an FA on a finite word $v = a_1a_2a_3 \cdots a_n$ is a sequence of states q_0, q_1, \dots, q_n such that $(q_i, a_{i+1}, q_{i+1}) \in \delta$. The run v is *accepting* if $q_n \in F$. A word u is accepting if it has an accepting run. The language of A , denoted by $L(A)$, is the set $\{u \in \Sigma^* \mid u \text{ is accepted by } A\}$. Given two FAs A and B , one can construct a product FA $A \times B$ recognizing $L(A) \cap L(B)$ using a standard product construction.

A *deterministic finite automaton* (DFA) is an FA such that $\delta(q, a)$ is a singleton for any $q \in Q$ and $a \in \Sigma$. For DFA, we write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$. The transition can be lifted to words by defining $\delta(q, \epsilon) = q$ and $\delta(q, av) = \delta(\delta(q, a), v)$ for $q \in Q, a \in \Sigma$ and $v \in \Sigma^*$. We also use $A(v)$ as a shorthand for $\delta(q_0, v)$.

A *Büchi automaton* (BA) has the same structure as an FA, except that it accepts only infinite words. A run of an infinite word in a BA is an infinite sequence of states defined similarly to the case of a finite word in an FA. An infinite word w is accepted by a BA iff it has a run visiting at least one accepting state infinitely often. The language defined by a BA A , denoted by $L(A)$, is the set $\{w \in \Sigma^\omega \mid w \text{ is accepted by } A\}$. An ω -language $L \subseteq \Sigma^\omega$ is ω -regular iff there exists a BA A such that $L = L(A)$.

Theorem 1 (Ultimately Periodic Words of ω -Regular Languages [13]). *Let L, L' be two ω -regular languages. Then $L = L'$ if and only if $UP(L) = UP(L')$.*

Definition 1 (Family of DFAs (FDDFA) [10]). *A family of DFAs $\mathcal{F} = (M, \{A^q\})$ over an alphabet Σ consists of a leading automaton $M = (\Sigma, Q, q_0, \delta)$ and progress DFAs $A^q = (\Sigma, Q_q, s_q, \delta_q, F_q)$ for each $q \in Q$.*

Notice that the leading automaton M is a DFA without accepting states. Each FDDFA \mathcal{F} characterizes a set of ultimately periodic words $UP(\mathcal{F})$. Formally, an ultimately periodic word w is in $UP(\mathcal{F})$ iff it has a decomposition (u, v) *accepted* by \mathcal{F} . A decomposition (u, v) is accepted by \mathcal{F} iff $M(uv) = M(u)$ and $v \in L(A^{M(u)})$. An example of an FDDFA \mathcal{F} is depicted in Fig. 1. The leading automaton M has only one state ϵ . The progress automaton of ϵ is A^ϵ . The word $(ba)^\omega$ is in $UP(\mathcal{F})$ because it has a decomposition (ba, ba) such that $M(ba \cdot ba) = M(ba)$ and $ba \in L(A^{M(ba)}) = L(A^\epsilon)$. It is easy to see that the decomposition (bab, ab) is not accepted by \mathcal{F} since $ab \notin L(A^{M(bab)}) = L(A^\epsilon)$.

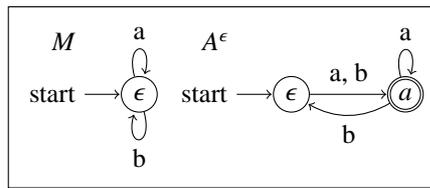


Fig. 1. An example of an FDDFA

and *recurrent* FDDFA. Their formal definitions are given in terms of *right congruence*.

An equivalence relation \sim on Σ^* is a right congruence if $x \sim y$ implies $xv \sim yv$ for every $x, y, v \in \Sigma^*$. The index of \sim , denoted by $|\sim|$, is the number of equivalence classes of \sim . We use Σ^*/\sim to denote the equivalence classes of the right congruence \sim . A *finite right congruence* is a right congruence with a finite index. For a word $v \in \Sigma^*$, we use

For any ω -regular language L , there exists an FDDFA \mathcal{F} such that $UP(L) = UP(\mathcal{F})$ [10]. We show in Sec. 6 that it is not the case for the reverse direction. More precisely, in [10], three kinds of FDDFAs are suggested as the canonical representation of ω -regular languages, namely *periodic* FDDFA, *syntactic* FDDFA

the notation $[v]_{\sim}$ to represent the class of \sim in which v resides and ignore the subscript \sim when the context is clear. The right congruence \sim_L of a given ω -regular language L is defined such that $x \sim_L y$ iff $\forall w \in \Sigma^\omega. xw \in L \iff yw \in L$. The index of \sim_L is finite because it is not larger than the number of states in a deterministic Muller automaton recognizing L [33].

Definition 2 (Canonical FDFFA [10]). Given an ω -regular language L , a periodic (respectively, syntactic and recurrent) FDFFA $\mathcal{F} = (M, \{A^u\})$ of L is defined as follows. The leading automaton M is the tuple $(\Sigma, \Sigma^* / \sim_L, [\epsilon]_{\sim_L}, \delta)$, where $\delta([u]_{\sim_L}, a) = [ua]_{\sim_L}$ for all $u \in \Sigma^*$ and $a \in \Sigma$.

We define the right congruences \approx_P^u, \approx_S^u , and \approx_R^u for progress automata A^u of periodic, syntactic, and recurrent FDFFA respectively as follows:

$$\begin{aligned} x \approx_P^u y \text{ iff} & \quad \forall v \in \Sigma^*, u(xv)^\omega \in L \iff u(yv)^\omega \in L, \\ x \approx_S^u y \text{ iff} & \quad ux \sim_L uy \text{ and } \forall v \in \Sigma^*, uxv \sim_L u \implies (u(xv)^\omega \in L \iff u(yv)^\omega \in L), \text{ and} \\ x \approx_R^u y \text{ iff} & \quad \forall v \in \Sigma^*, uxv \sim_L u \wedge u(xv)^\omega \in L \iff uyv \sim_L u \wedge u(yv)^\omega \in L. \end{aligned}$$

The progress automaton A^u is the tuple $(\Sigma, \Sigma^* / \approx_K^u, [\epsilon]_{\approx_K^u}, \delta_K, F_K)$, where $\delta_K([u]_{\approx_K^u}, a) = [ua]_{\approx_K^u}$ for all $u \in \Sigma^*$ and $a \in \Sigma$. The accepting states F_K is the set of equivalence classes $[v]_{\approx_K^u}$ for which $uv \sim_L u$ and $uv^\omega \in L$ when $K \in \{S, R\}$ and the set of equivalence classes $[v]_{\approx_K^u}$ for which $uv^\omega \in L$ when $K \in \{P\}$.

In this paper, by an abuse of notation, we use a finite word u to denote the state in a DFA in which the equivalence class $[u]$ resides.

Lemma 1 ([10]). Let \mathcal{F} be a periodic (syntactic, recurrent) FDFFA of an ω -regular language L . Then $UP(\mathcal{F}) = UP(L)$.

Lemma 2 ([9]). Let \mathcal{F} be a periodic (syntactic, recurrent) FDFFA of an ω -regular language L . One can construct a BA recognizing L from \mathcal{F} .

3 Büchi Automata Learning Framework based on FDFFA

We begin with an introduction of the framework of learning BA recognizing an unknown ω -regular language L .

Overview of the framework: First, we assume that we already have a BA teacher who knows the unknown ω -regular language L and answers *membership* and *equivalence* queries about L . More precisely, a membership query $\text{Mem}^{\text{BA}}(uv^\omega)$ asks if $uv^\omega \in L$. For an equivalence query $\text{Equ}^{\text{BA}}(B)$, the BA teacher answers “yes” when $L(B) = L$, otherwise it returns “no” as well as a counterexample $uv^\omega \in L \oplus L(B)$.

The framework depicted in Fig. 2 consists of two components, namely the FDFFA learner and the FDFFA teacher. Note that one can place any FDFFA learning algorithm to the FDFFA learner component. For instance, one can use the FDFFA learner from [10] which employs a table to store query results, or the FDFFA learner using a classification tree proposed in this paper. The FDFFA teacher can be any teacher who can answer membership and equivalence queries about an unknown FDFFA.

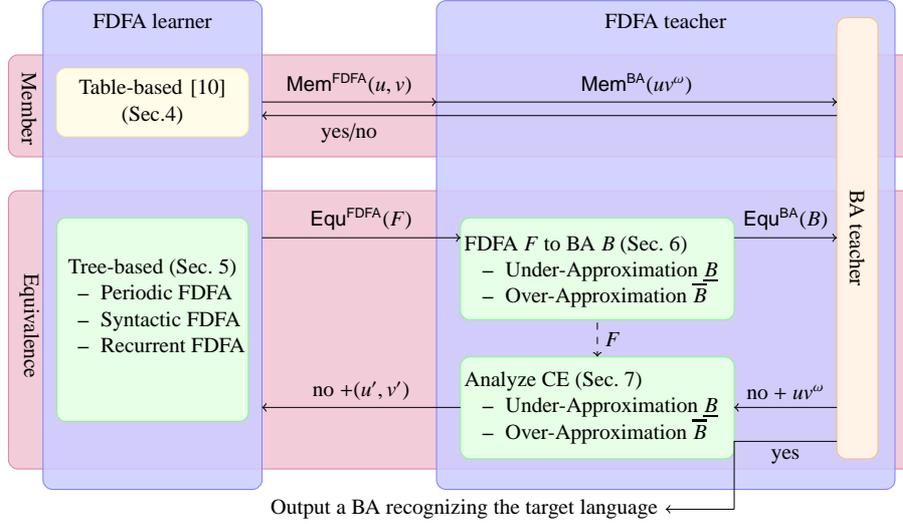


Fig. 2. Overview of the learning framework based on FDFA learning. The components in boxes are results from existing works. The components in boxes are our new contributions.

FDFA learners: The FDFA learners component will be introduced in Sec. 4 and 5. We first briefly review the table-based FDFA learning algorithms [10] in Sec. 4. Our tree-based learning algorithm for canonical FDFAs will be introduced in Sec. 5. The algorithm is inspired by the tree-based L^* learning algorithm [30]. Nevertheless, applying the tree structure to learn FDFAs is not a trivial task. For example, instead of a binary tree used in [30], we need to use a K -ary tree to learn syntactic FDFAs. The use of K -ary tree complicates the procedure of refining the classification tree and automaton construction. More details will be provided in Sec. 5.

FDFA teacher: The task of the FDFA teacher is to answer queries $\text{Mem}^{\text{FDFA}}(u, v)$ and $\text{Equ}^{\text{FDFA}}(F)$ posed by the FDFA learner. Answering $\text{Mem}^{\text{FDFA}}(u, v)$ is easy. The FDFA teacher just needs to redirect the result of $\text{Mem}^{\text{BA}}(uv^\omega)$ to the FDFA learner. Answering equivalence query $\text{Equ}^{\text{FDFA}}(F)$ is more tricky.

From an FDFA F to a BA B : The FDFA teacher needs to transform an FDFA F to a BA B to pose an equivalence query $\text{Equ}^{\text{BA}}(B)$. In Sec. 6, we show that, in general, it is impossible to build a BA B from an FDFA F such that $UP(L(B)) = UP(F)$. Therefore in Sec. 6, we propose two methods to approximate $UP(F)$, namely the *under-approximation* method and the *over-approximation* method. As the name indicates, the under-approximation (respectively, over-approximation) method constructs a BA B from F such that $UP(L(B)) \subseteq UP(F)$ (respectively, $UP(F) \subseteq UP(L(B))$). The under-approximation method is modified from the algorithm in [14]. Note that if the FDFAs are the canonical representations, the BAs built by the under-approximation method recognize the same ultimately periodic words as the FDFAs, which makes it a com-

plete method for BA learning (Lem. 1 and 2). As for the over-approximation method, we cannot guarantee to get a BA B such that $UP(L(B)) = UP(F)$ even if the F is a canonical representation, which thus makes it an incomplete method. However, in the worst case, the over-approximation method produces a BA whose number of states is only quadratic in the size of the FDFA. In contrast, the number of states in the BA constructed by the under-approximation method is cubic in the size of the FDFA.

Counterexample analysis: If the FDFA teacher receives “no” and a counterexample uv^ω from the BA teacher, the FDFA teacher has to return “no” as well as a valid decomposition (u', v') that can be used by the FDFA learner to refine F . In Sec. 7, we show how the FDFA teacher chooses a pair (u', v') from uv^ω that allows FDFA learner to refine current FDFA F . As the dashed line with a label F in Fig. 2 indicates, we use the current conjectured FDFA F to analyze the counterexample. The under-approximation method and the over-approximation method of FDFA to BA translation require different counterexample analysis procedures. More details will be provided in Sec. 7.

Once the BA teacher answers “yes” for the equivalence query $\text{Equ}^{\text{BA}}(B)$, the FDFA teacher will terminate the learning procedure and outputs a BA recognizing L .

4 Table-based Learning Algorithm for FDFAs

In this section, we briefly introduce the table-based learner for FDFAs [10]. It employs a structure called *observation table* [7] to organize the results obtained from queries and propose candidate FDFAs. The table-based FDFA learner simultaneously runs several instances of DFA learners. The DFA learners are very similar to the L^* algorithm [7], except that they use different conditions to decide if two strings belong to the same state (based on Def. 2). More precisely, the FDFA learner uses one DFA learner L_M^* for the leading automaton M , and for each state u in M , one DFA learner $L_{A^u}^*$ for each progress automaton A^u . The table-based learning procedure works as follows. The learner L_M^* first closes the observation table by posing membership queries and then constructs a candidate for leading automaton M . For every state u in M , the table-based algorithm runs an instance of DFA learner $L_{A^u}^*$ to find the progress automaton A^u . When all DFA learners propose candidate DFAs, the FDFA learner assembles them to an FDFA $\mathcal{F} = (M, \{A^u\})$ and then poses an equivalence query for it. The FDFA teacher will either return “yes” which means the learning algorithm succeeds or return “no” accompanying with a counterexample. Once receiving the counterexample, the table-based algorithm will decide which DFA learner should refine its candidate DFA. We refer interested readers to [10] for more details of the table-based algorithm.

5 Tree-based Learning Algorithm for FDFAs

In this section, we provide our tree-based learning algorithm for FDFAs. To that end, we first define the classification tree structure for FDFA learning in Sec. 5.1 and present the tree-based algorithm in Sec. 5.2.

5.1 Classification Tree Structure in Learning

Here we present our classification tree structure for FDFA learning. Compared to the classification tree defined in [30], ours is not restricted to be a binary tree. Formally, a classification tree is a tuple $\mathcal{T} = (N, r, L_n, L_e)$ where $N = I \cup T$ is a set of nodes consisting of the set I of *internal nodes* and the set T of *terminal nodes*, the node $r \in N$ is the root of the tree, $L_n : N \rightarrow \Sigma^* \cup (\Sigma^* \times \Sigma^*)$ labels an internal node with an *experiment* and a terminal node with a *state*, and $L_e : N \times D \rightarrow N$ maps a parent node and a label to its corresponding child node, where the set of labels D will be specified below.

During the learning procedure, we maintain a *leading tree* \mathcal{T} for the leading automaton M , and for every state u in M , we keep a *progress tree* \mathcal{T}_u for the progress automaton A^u . For every classification tree, we define a tree experiment function $\mathbf{TE} : \Sigma^* \times (\Sigma^* \cup (\Sigma^* \times \Sigma^*)) \rightarrow D$. Intuitively, $\mathbf{TE}(x, e)$ computes the entry value at row (state) x and column (experiment) e of an observation table in table-based learning algorithms. The labels of nodes in the classification tree \mathcal{T} satisfy the follow invariants: Let $t \in T$ be a terminal node labeled with a state $x = L_n(t)$. Let $t' \in I$ be an ancestor node of t labeled with an experiment $e = L_n(t')$. Then the child of t' following the label $\mathbf{TE}(x, e)$, i.e., $L_e(t', \mathbf{TE}(x, e))$, is either the node t or an ancestor node of t .

Leading tree \mathcal{T} : The leading tree \mathcal{T} for M is a binary tree with labels $D = \{F, T\}$. The tree experiment function $\mathbf{TE}(u, (x, y)) = T$ iff $uxy^\omega \in L$ (recall the definition of \sim_L in Sec. 2) where $u, x, y \in \Sigma^*$. Intuitively, each internal node n in \mathcal{T} is labeled by an experiment xy^ω represented as (x, y) . For any word $u \in \Sigma^*$, $uxy^\omega \in L$ (or $uxy^\omega \notin L$) implies that the equivalence class of u lies in the T-subtree (or F-subtree) of n .

Progress tree \mathcal{T}_u : The progress trees \mathcal{T}_u and the corresponding function $\mathbf{TE}(x, e)$ are defined based on the right congruences \approx_p^u , \approx_s^u , and \approx_R^u of canonical FDFAs in Def. 2.

Periodic FDFA: The progress tree for periodic FDFA is also a binary tree labeled with $D = \{F, T\}$. The experiment function $\mathbf{TE}(x, e) = T$ iff $u(xe)^\omega \in L$ where $x, e \in \Sigma^*$.

Syntactic FDFA: The progress tree for syntactic FDFA is a K -ary tree with labels $D = \overline{Q} \times \{A, B, C\}$ where Q is the set of states in the leading automaton M . For all $x, e \in \Sigma^*$, the experiment function $\mathbf{TE}(x, e) = (M(ux), t)$, where $t = A$ iff $u = M(uxe) \wedge u(xe)^\omega \in L$, $t = B$ iff $u = M(uxe) \wedge u(xe)^\omega \notin L$, and $t = C$ iff $u \neq M(uxe)$.

For example, assuming that M is constructed from the right congruence \sim_L , for any two states x and y such that $\mathbf{TE}(x, e) = \mathbf{TE}(y, e) = (z, A)$, it must be the case that $ux \sim_L uy$ because $M(ux) = z = M(uy)$. Moreover, the experiment e cannot distinguish x and y because $uxe \sim_L u \sim_L uye$ and both $u(xe)^\omega, u(ye)^\omega \in L$.

Recurrent FDFA: The progress tree for recurrent FDFA is a binary tree labeled with $D = \{F, T\}$. The function $\mathbf{TE}(x, e) = T$ iff $u(xe)^\omega \in L \wedge u = M(uxe)$ where $x, e \in \Sigma^*$.

5.2 Tree-based Learning Algorithm

The tree-based learning algorithm first initializes the leading tree \mathcal{T} and the progress tree \mathcal{T}_ϵ as a tree with only one terminal node r labeled by ϵ .

From a classification tree $\mathcal{T} = (N, r, L_n, L_e)$, the learner constructs a candidate of a leading automaton $M = (\Sigma, Q, \epsilon, \delta)$ or a progress automaton $A^u = (\Sigma, Q, \epsilon, \delta, F)$ as follow. The set of states is $Q = \{L_n(t) \mid t \in T\}$. Given $s \in Q$ and $a \in \Sigma$, the transition function $\delta(s, a)$ is constructed by the following procedure. Initially the current node $n := r$. If n is a terminal node, it returns $\delta(s, a) = L_n(n)$. Otherwise, it picks a unique child n' of n with $L_e(n, \mathbf{TE}(sa, L_n(n))) = n'$, updates the current node to n' , and repeats the procedure⁴. By Def. 2, the set of accepting states F of a progress automaton can be identified from the structure of M with the help of membership queries. For periodic FDFA, $F = \{v \mid uv^\omega \in L, v \in Q\}$ and for syntactic and recurrent FDFA, $F = \{v \mid uv \sim_M u, uv^\omega \in L, v \in Q\}$.

Whenever the learner has constructed an FDFA $\mathcal{F} = (M, \{A^u\})$, it will pose an equivalence query for \mathcal{F} . If the teacher returns “no” and a counterexample (u, v) , the learner has to refine the classification tree and propose another candidate of FDFA.

Definition 3 (Counterexample for FDFA Learner). *Given the conjectured FDFA \mathcal{F} and the target language L , we say that the counterexample*

- (u, v) is positive if $uv \sim_M u$, $uv^\omega \in UP(L)$, and (u, v) is not accepted by \mathcal{F} ,
- (u, v) is negative if $uv \sim_M u$, $uv^\omega \notin UP(L)$, and (u, v) is accepted by \mathcal{F} .

We remark that in our case all counterexamples (u, v) from the FDFA teacher satisfy the constraint $uv \sim_M u$, which corresponds to the *normalized factorization* form in [10].

Counterexample guided refinement of \mathcal{F} : Below we show how to refine the classification trees based on a negative counterexample (u, v) . The case of a positive counterexample is symmetric. By definition, we have $uv \sim_M u$, $uv^\omega \notin UP(L)$ and (u, v) is accepted by \mathcal{F} . Let $\tilde{u} = M(u)$, if $\tilde{u}v^\omega \in UP(L)$, the refinement of the leading tree is performed, otherwise $\tilde{u}v^\omega \notin UP(L)$, the refinement of the progress tree is performed.

Refinement for the leading tree: In the leading automaton M of the conjectured FDFA, if a state p has a transition to a state q via a letter a , i.e. $q = M(pa)$, then pa has been assigned to the terminal node labeled by q during the construction of M . If one also finds an experiment e such that $\mathbf{TE}(q, e) \neq \mathbf{TE}(pa, e)$, then we know that q and pa should not belong to the same state in a leading automaton. W.l.o.g., we assume $\mathbf{TE}(q, e) = F$. In such a case, the leading tree can be refined by replacing the terminal node labeled with q by a tree such that (i) its root is labeled by e , (ii) its left child is a terminal node labeled by q , and (iii) its right child is a terminal node labeled by pa .

Below we discuss how to extract the required states p, q and experiment e . Let $|u| = n$ and $s_0s_1 \cdots s_n$ be the run of M over u . Note that $s_0 = M(\epsilon) = \epsilon$ and $s_n = M(u) = \tilde{u}$. From the facts that (u, v) is a negative counterexample and $\tilde{u}v^\omega \in UP(L)$ (the condition to refine the leading tree), we have $\mathbf{TE}(s_0, (u[1 \cdots n], v)) = F \neq T = \mathbf{TE}(s_n, (\epsilon, v)) = \mathbf{TE}(s_n, (u[n+1 \cdots n], v))$ because $uv^\omega \notin UP(L)$ and $\tilde{u}v^\omega \in UP(L)$. Recall that we have $w[j \cdots k] = \epsilon$ when $j > k$. Therefore, there must exist a smallest $j \in [1 \cdots n]$ such

⁴ For syntactic FDFA, it can happen that $\delta(s, a)$ goes to a “new” terminal node. A new state for the FDFA is identified in such a case.

that $\mathbf{TE}(s_{j-1}, (u[j \cdots n], v)) \neq \mathbf{TE}(s_j, (u[j+1 \cdots n], v))$. It follows that we can use the experiment $e = (u[j+1 \cdots n], v)$ to distinguish $q = s_j$ and $pa = s_{j-1}u[j]$.

Example 1. Consider a conjectured FDFA \mathcal{F} in Fig. 1 produced during the process of learning $L = a^\omega + b^\omega$. The corresponding leading tree \mathcal{T} and the progress tree \mathcal{T}_ϵ are depicted on the left of Fig. 3. The dotted line is for the F label and the solid one is for the T label. Suppose the FDFA teacher returns a negative counterexample (ab, b) . The leading tree has to be refined since $M(ab)b^\omega = b^\omega \in L$. We find an experiment (b, b) to differentiate ϵ and a using the procedure above and update the leading tree \mathcal{T} to \mathcal{T}' . The leading automaton M constructed from \mathcal{T}' is depicted on the right of Fig. 3.

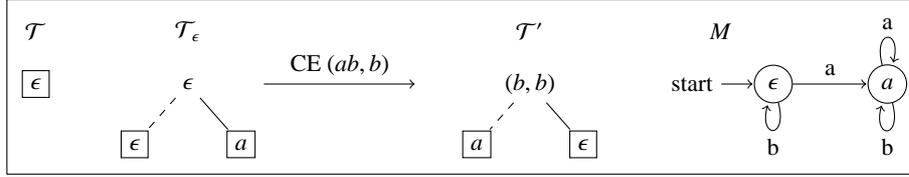


Fig. 3. Refinement of the leading tree and the corresponding leading automaton

Refinement for the progress tree: Here we explain the case of periodic FDFAs. The other cases are similar and we leave the details in Appendix B. Recall that $\tilde{u}v^\omega \notin UP(L)$ and thus the algorithm refines the progress tree $\mathcal{T}_{\tilde{u}}$. Let $|v| = n$ and $h = s_0s_1 \cdots s_n$ be the corresponding run of $A^{\tilde{u}}$ over v . Note that $s_0 = A^{\tilde{u}}(\epsilon) = \epsilon$ and $s_n = A^{\tilde{u}}(v) = \tilde{v}$. We have $\tilde{u}(\tilde{v})^\omega \in UP(L)$ because \tilde{v} is an accepting state. From the facts that $\tilde{u}v^\omega \notin UP(L)$ and $\tilde{u}(\tilde{v})^\omega \in UP(L)$, we have $\mathbf{TE}(s_0, v[1 \cdots n]) = \text{F} \neq \text{T} = \mathbf{TE}(s_n, \epsilon) = \mathbf{TE}(s_n, v[n+1 \cdots n])$. Therefore, there must exist a smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}, v[j \cdots n]) \neq \mathbf{TE}(s_j, v[j+1 \cdots n])$. It follows that we can use the experiment $e = v[j+1 \cdots n]$ to distinguish $q = s_j$, $pa = s_{j-1}v[j]$ and refine the progress tree $\mathcal{T}_{\tilde{u}}$.

Optimization: Example 1 also illustrates the fact that the counterexample (ab, b) may not be eliminated right away after the refinement. In this case, it is still a valid counterexample (assuming that the progress tree \mathcal{T}_ϵ remains unchanged). Thus as an optimization in our tool, one can repeatedly use the counterexample until it is eliminated.

6 From FDFA to Büchi Automata

Since the FDFA teacher exploits the BA teacher for answering equivalence queries, it needs first to convert the given FDFA into a BA. Unfortunately, with the following example, we show that in general, it is impossible to construct a *precise* BA B for an FDFA \mathcal{F} such that $UP(L(B)) = UP(\mathcal{F})$.

Example 2. Consider a non-canonical FDFA \mathcal{F} in Fig. 4, we have $UP(\mathcal{F}) = \bigcup_{n=0}^{\infty} \{a, b\}^*$.

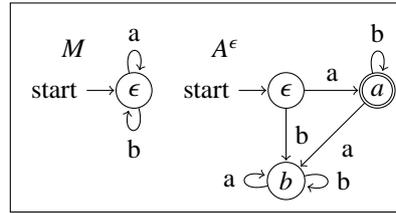


Fig. 4. An FDFA \mathcal{F} such that $UP(\mathcal{F})$ does not characterize an ω -regular language

$(ab^n)^\omega$. We assume that $UP(\mathcal{F})$ characterizes an ω -regular language L . It is known that the periodic FDFA recognizes exactly the ω -regular language and the index of each right congruence is finite [10]. However, we can show that the right congruence \approx_P^ϵ of a periodic FDFA of L is of infinite index. Observe that $ab^k \not\approx_P^\epsilon ab^j$ for any $k, j \geq 1$ and $k \neq j$, because $\epsilon \cdot (ab^k \cdot ab^k)^\omega \in UP(\mathcal{F})$ and $\epsilon \cdot (ab^j \cdot ab^k)^\omega \notin UP(\mathcal{F})$. It follows that \approx_P^ϵ is of infinite index. We conclude that $UP(\mathcal{F})$ cannot characterize an ω -regular language.

We circumvent the above problem by proposing two BAs $\underline{B}, \overline{B}$, which under- and over-approximate the ultimately periodic words of an FDFA. Given an FDFA $\mathcal{F} = (M, \{A^u\})$ with $M = (\Sigma, Q, q_0, \delta)$ and $A^u = (\Sigma, Q_u, s_u, \delta_u, F_u)$ for all $u \in Q$, we define $M_v^s = (\Sigma, Q, s, \delta, \{v\})$ and $(A^u)_v^s = (\Sigma, Q_u, s, \delta_u, \{v\})$, i.e., the DFA obtained from M and A^u by setting their initial and accepting states as s and $\{v\}$, respectively. Define $N_{(u,v)} = \{v^\omega \mid uv \sim_M u \wedge v \in L((A^u)_v^s)\}$. Then $UP(\mathcal{F}) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot N_{(u,v)}$.

We construct \overline{B} and \underline{B} by approximating the set $N_{(u,v)}$. For \overline{B} , we first define an FA $\overline{P}_{(u,v)} = (\Sigma, Q_{u,v}, s_{u,v}, \{f_{u,v}\}, \delta_{u,v}) = M_u^u \times (A^u)_v^{s_u}$ and let $\overline{N}_{(u,v)} = L(\overline{P}_{(u,v)})^\omega$. Then one can construct a BA $(\Sigma, Q_{u,v} \cup \{f\}, s_{u,v}, \{f\}, \delta_{u,v} \cup \delta_f)$ recognizing $\overline{N}_{(u,v)}$ where f is a ‘‘fresh’’ state and $\delta_f = \{(f, \epsilon, s_{u,v}), (f_{u,v}, \epsilon, f)\}$. For \underline{B} , we define an FA $\underline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ and let $\underline{N}_{(u,v)} = L(\underline{P}_{(u,v)})^\omega$. One can construct a BA recognizing $\underline{N}_{(u,v)}$ using a similar construction to the case of $\overline{N}_{(u,v)}$. In Def. 4 we show how to construct BAs \overline{B} and \underline{B} s.t. $UP(L(\overline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \overline{N}_{(u,v)}$ and $UP(L(\underline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \underline{N}_{(u,v)}$.

Definition 4. Let $\mathcal{F} = (M, \{A^u\})$ be an FDFA where $M = (\Sigma, Q, q_0, \delta)$ and $A^u = (\Sigma, Q_u, s_u, F_u, \delta_u)$ for every $u \in Q$. Let $(\Sigma, Q_{u,v}, s_{u,v}, \{f_{u,v}\}, \delta_{u,v})$ be a BA recognizing $\underline{N}_{(u,v)}$ (respectively $\overline{N}_{(u,v)}$). Then the BA \underline{B} (respectively \overline{B}) is defined as the tuple

$$\left(\Sigma, Q \cup \bigcup_{u \in Q, v \in F_u} Q_{u,v}, q_0, \bigcup_{u \in Q, v \in F_u} \{f_{u,v}\}, \delta \cup \bigcup_{u \in Q, v \in F_u} \delta_{u,v} \cup \bigcup_{u \in Q, v \in F_u} \{(u, \epsilon, s_{u,v})\} \right).$$

Lemma 3 (Sizes and Languages of \underline{B} and \overline{B}). Let \mathcal{F} be an FDFA and $\underline{B}, \overline{B}$ be the BAs constructed from \mathcal{F} by Def. 4. Let n and k be the numbers of states in the leading automaton and the largest progress automaton of \mathcal{F} . The number of states of \underline{B} and \overline{B} are in $O(n^2 k^3)$ and $O(n^2 k^2)$, respectively. Moreover, $UP(L(\underline{B})) \subseteq UP(\mathcal{F}) \subseteq UP(L(\overline{B}))$ and we have $UP(L(\underline{B})) = UP(\mathcal{F})$ when \mathcal{F} is a canonical FDFA.

The properties below will be used later in analyzing counterexamples.

Lemma 4. Given an FDFA $\mathcal{F} = (M, \{A^u\})$, and \underline{B} the BA constructed from \mathcal{F} by Def. 4. If (u, v^k) is accepted by \mathcal{F} for every $k \geq 1$, then $uv^\omega \in UP(L(\underline{B}))$.

Lemma 5. Given an ω -word $w \in UP(L(\overline{B}))$, there exists a decomposition (u, v) of w and $n \geq 1$ such that $v = v_1 \cdot v_2 \cdots v_n$ and for all $i \in [1 \cdots n]$, $v_i \in L(A^{M(i)})$ and $uv_i \sim_M u$.

Fig. 5 depicts the BAs \overline{B} and \underline{B} constructed from the FDFA \mathcal{F} in Fig. 1. In the example, we can see that the $b^\omega \in UP(\mathcal{F})$ while $b^\omega \notin UP(L(\underline{B}))$.

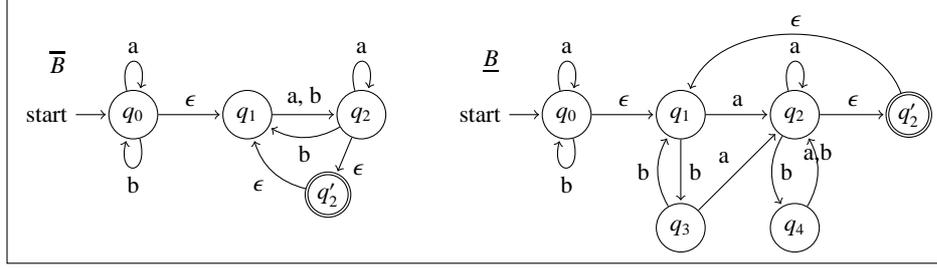


Fig. 5. NBA \overline{B} and \underline{B} for \mathcal{F} in Fig. 1

7 Counterexample Analysis for FDFFA Teacher

During the learning procedure, if we failed the equivalence query for the BA B , the BA teacher will return a counterexample uv^ω to the FDFFA teacher.

Definition 5 (Counterexample for the FDFFA Teacher). *Given the conjectured BA $B \in \{\underline{B}, \overline{B}\}$, the target language L , we say that*

- uv^ω is a positive counterexample if $uv^\omega \in UP(L)$ and $uv^\omega \notin UP(L(B))$,
- uv^ω is a negative counterexample if $uv^\omega \notin UP(L)$ and $uv^\omega \in UP(L(B))$.

Obviously, the above is different to the counterexample for the FDFFA learner in Def. 3. Below we illustrate the necessity of the counterexample analysis by an example.

Example 3. Again, consider the conjectured FDFFA \mathcal{F} depicted in Fig. 1 for $L = a^\omega + b^\omega$. Suppose the BA teacher returns a negative counterexample $(ba)^\omega$. In order to remove $(ba)^\omega \in UP(\mathcal{F})$, one has to find a decomposition of $(ba)^\omega$ that \mathcal{F} accepts, which is the goal of the counterexample analysis. Not all decompositions of $(ba)^\omega$ are accepted by \mathcal{F} . For instance, (ba, ba) is accepted while (bab, ab) is not.

A positive (respectively negative) counterexample uv^ω for the FDFFA teacher is *spurious* if $uv^\omega \in UP(\mathcal{F})$ (respectively $uv^\omega \notin UP(\mathcal{F})$). Suppose we use the under-approximation method to construct the BA \underline{B} from \mathcal{F} depicted in Fig. 5. The BA teacher returns a spurious positive counterexample b^ω , which is in $UP(\mathcal{F})$ but not in $UP(L(\underline{B}))$. We show later that in such a case, one can always find a decomposition, in this example (b, bb) , as the counterexample for the FDFFA learner.

Given FDFFA $\mathcal{F} = (M, \{A^u\})$, in order to analyze the counterexample uv^ω , we define:

- an FA $\mathcal{D}_{u\$v}$ with $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, uv^\omega = u'v'^\omega\}$,
- an FA \mathcal{D}_1 with $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \sim_M u, v \in L(A^{M(u)})\}$, and
- an FA \mathcal{D}_2 with $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \not\sim_M u, v \notin L(A^{M(u)})\}$.

Here $\$$ is a letter not in Σ . Intuitively, $\mathcal{D}_{u\$v}$ accepts every possible decomposition (u', v') of uv^ω , \mathcal{D}_1 recognizes every decomposition (u', v') which is accepted by \mathcal{F} and \mathcal{D}_2 accepts every decomposition (u', v') which is not accepted by \mathcal{F} yet $u'v' \sim_M u'$.

Given a BA \underline{B} constructed by the under-approximation method to construct a BA \underline{B} from \mathcal{F} , we have that $UP(L(\underline{B})) \subseteq UP(\mathcal{F})$. Fig. 6(a) depicts all possible cases of $uv^\omega \in UP(L(\underline{B})) \oplus UP(L)$.

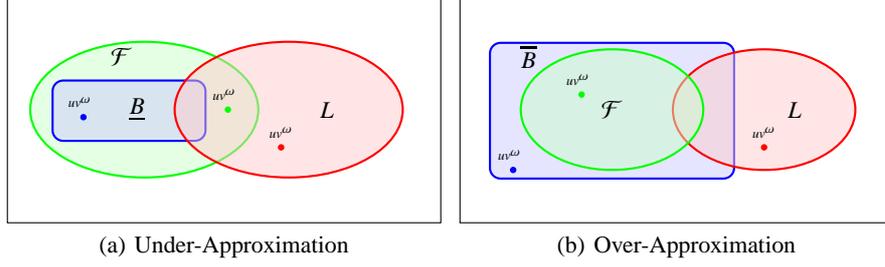


Fig. 6. The Case for Counterexample Analysis

- U1 : $uv^\omega \in UP(L) \wedge uv^\omega \notin UP(\mathcal{F})$ (Point in red). The word uv^ω is a positive counterexample, one has to find a decomposition (u', v') such that $u'v' \sim_M u'$ and $u'v'^\omega = uv^\omega$. This can be easily done by taking a word $u'Sv' \in L(\mathcal{D}_{u'Sv'}) \cap L(\mathcal{D}_2)$.
- U2 : $uv^\omega \notin UP(L) \wedge uv^\omega \in UP(\mathcal{F})$ (Point in blue). The word uv^ω is a negative counterexample, one needs to find a decomposition (u', v') of uv^ω that is accepted by \mathcal{F} . This can be done by taking a word $u'Sv' \in L(\mathcal{D}_{u'Sv'}) \cap L(\mathcal{D}_1)$.
- U3 : $uv^\omega \in UP(L) \wedge uv^\omega \in UP(\mathcal{F})$ (Point in green). The word uv^ω is a spurious positive counterexample. Suppose the decomposition (u, v) of uv^ω is accepted by \mathcal{F} , according to Lem. 4, there must exist some $k \geq 1$ such that (u, v^k) is not accepted by \mathcal{F} . Thus, we can also use the same method in U1 to get a counterexample (u', v') .

We can also use the over-approximation construction to get a BA \overline{B} from \mathcal{F} such that $UP(\mathcal{F}) \subseteq UP(L(\overline{B}))$, and all possible cases for a counterexample $uv^\omega \in UP(L(\overline{B})) \oplus UP(L)$ is depicted in Fig. 6(b).

- O1 : $uv^\omega \in UP(L) \wedge uv^\omega \notin UP(\mathcal{F})$ (Point in red). The word uv^ω is a positive counterexample that can be dealt with the same method for case U1.
- O2 : $uv^\omega \notin UP(L) \wedge uv^\omega \in UP(\mathcal{F})$ (Point in green). The word uv^ω is a negative counterexample that can be dealt with the same method for case U2.
- O3 : $uv^\omega \notin UP(L) \wedge uv^\omega \notin UP(\mathcal{F})$ (Point in blue). In this case, uv^ω is a spurious negative counterexample. In such a case it is possible that we cannot find a valid decomposition of uv^ω to refine \mathcal{F} . By Lem. 5, we can find a decomposition (u', v') of uv^ω such that $v' = v_1v_2 \cdots v_n$, $u'v_i \sim_M u'$, and $v_i \in L(A^{M(u')})$ for some $n \geq 1$. It follows that (u', v_i) is accepted by \mathcal{F} . If we find some $i \in [1 \cdots n]$ such that $u'v_i^\omega \notin UP(L)$, then we return (u', v_i) , otherwise, the algorithm aborts with an error.

Finally, we note that determining whether $uv^\omega \in UP(L)$ can be done by posing a membership query $\text{Mem}^{\text{BA}}(uv^\omega)$, and checking whether $uv^\omega \in UP(\mathcal{F})$ boils down to checking the emptiness of $L(\mathcal{D}_{u'Sv'}) \cap L(\mathcal{D}_1)$. The construction for $\mathcal{D}_{u'Sv'}$, \mathcal{D}_1 , and \mathcal{D}_2 , and the correctness proof of counterexample analysis are given in Appendix D.

8 Complexity

We discuss the complexity of tree-based FDFA learning algorithms in Sec. 5. Let $\mathcal{F} = (M, \{A^u\})$ be the corresponding periodic FDFA of the ω -regular language L , and let n

be the number of states in the leading automaton M and k be the number of states in the largest progress automaton A^u . We remark that \mathcal{F} is uniquely defined for L and the table-based algorithm needs the same amount of equivalence queries as the tree-based one in the worst case. Given a counterexample (u, v) returned from the FDFA teacher, we define its *length* as $|u| + |v|$.

Theorem 2 (Query Complexity). *Let (u, v) be the longest counterexample returned from the FDFA teacher. The number of equivalence queries needed for the tree-based FDFA learning algorithm to learn the periodic FDFA of L is in $O(n + nk)$, while the number of membership queries is in $O((n + nk) \cdot (|u| + |v| + (n + k) \cdot |\Sigma|))$.*

For the syntactic and recurrent FDFAs, the number of equivalence queries needed for the tree-based FDFA learning algorithm is in $O(n + n^3k)$, while the number of membership queries is in $O((n + n^3k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$.

The learning of syntactic and recurrent FDFAs requires more queries since once their leading automata have been modified, they need to redo the learning of all progress automata from scratch.

Theorem 3 (Space Complexity). *For all tree-based algorithms, the space required to learn the leading automaton is in $O(n)$. For learning periodic FDFA, the space required for each progress automaton is in $O(k)$, while for syntactic and recurrent FDFAs, the space required is in $O(nk)$. For all table-based algorithms, the space required to learn the leading automaton is in $O((n + n \cdot |\Sigma|) \cdot n)$. For learning periodic FDFA, the space required for each progress automaton is in $O((k + k \cdot |\Sigma|) \cdot k)$, while for syntactic and recurrent FDFAs, the space required is in $O((nk + nk \cdot |\Sigma|) \cdot nk)$.*

Theorem 4 (Correctness and Termination). *The BA learning algorithm based on the under-approximation method always terminates and returns a BA recognizing the unknown ω -regular language L in polynomial time. If the BA learning algorithm based on the over-approximation method terminates without reporting an error, it returns a BA recognizing L .*

Given a canonical FDFA \mathcal{F} , the under-approximation method produces a BA \underline{B} such that $UP(\mathcal{F}) = UP(L(\underline{B}))$, thus in the worst case, FDFA learner learns a canonical FDFA and terminates. In practice, the algorithm very often finds a BA recognizing L before converging to a canonical FDFA.

9 Experimental results

The ROLL library (<http://iscasmc.ios.ac.cn/roll>) is implemented in JAVA. The DFA operations in ROLL are delegated to the *dk.brics.automaton* package, and we use the RABIT tool [2, 3] to check the equivalence of two BAs. We evaluate the performance of ROLL using the smallest BAs corresponding to all the 295 LTL specifications available in BüchiStore [37], where the numbers of states in the BAs range over 1 to 17 and transitions range over 0 to 123. The machine we used for the experiments is a 2.5 GHz Intel Core i7-6500 with 4 GB RAM. We set the timeout period to 30 minutes.

Table 1. Overall experimental results. We show the results of 285 cases where all algorithms can finish the BA learning within the timeout period and list the number of cases cannot be solved (#Unsolved). The mark n^*/m denotes that there are n cases terminate with an error (in the over-approximation method) and it ran out of time for $m - n$ cases. The rows #St., #Tr., #MQ, and #EQ, are the numbers of states, transitions, membership queries, and equivalence queries. Time_{eq} is the time spent in answering equivalence queries and Time_{total} is the total execution time.

| Models | L^S | | L^{Periodic} | | | | $L^{\text{Syntactic}}$ | | | | $L^{\text{Recurrent}}$ | | | |
|--------------------------|-------------|--------------|-----------------------|-------|-------|------------|------------------------|-------|-------------|-------------|------------------------|-------|-------|-------|
| | Table | Tree | Table | | Tree | | Table | | Tree | | Table | | Tree | |
| | | | under | over | under | over | under | over | under | over | under | over | | |
| #Unsolved | 4 | 2 | 3 | 0/2 | 2 | 0/1 | 1 | 4*/5 | 0 | 3*/3 | 1 | 0/1 | 1 | 0/1 |
| #St. | 3078 | 3078 | 2481 | 2468 | 2526 | 2417 | 2591 | 2591 | 2274 | 2274 | 2382 | 2382 | 2400 | 2400 |
| #Tr. | 10.6k | 10.3k | 13.0k | 13.0k | 13.4k | 12.8k | 13.6k | 13.6k | 12.2k | 12.2k | 12.7k | 12.7k | 12.8k | 12.8k |
| #MQ | 105k | 114k | 86k | 85k | 69k | 67k | 236k | 238k | 139k | 139k | 124k | 124k | 126k | 126k |
| #EQ | 1281 | 2024 | 1382 | 1351 | 1950 | 1918 | 1399 | 1394 | 2805 | 2786 | 1430 | 1421 | 3037 | 3037 |
| $\text{Time}_{eq}(s)$ | 146 | 817 | 580 | 92 | 186 | 159 | 111 | 115 | 89 | 91 | 149 | 149 | 462 | 465 |
| $\text{Time}_{total}(s)$ | 183 | 861 | 610 | 114 | 213 | 186 | 140 | 144 | 118 | 120 | 175 | 176 | 499 | 501 |
| EQ(%) | 79.8 | 94.9 | 95.1 | 80.7 | 87.3 | 85.5 | 79.3 | 79.9 | 75.4 | 75.8 | 85.1 | 84.6 | 92.6 | 92.8 |

The overall experimental results are given in Tab. 1. In this section, we use L^S to denote the ω -regular learning algorithm in [20], and L^{Periodic} , $L^{\text{Syntactic}}$, and $L^{\text{Recurrent}}$ to represent the periodic, syntactic, and recurrent FDFA learning algorithm introduced in Sec. 4 and 5. From the table, we can find the following facts: (1) The BAs learned from L^S have more states but fewer transitions than their FDFA based counterpart. (2) L^{Periodic} uses fewer membership queries comparing to $L^{\text{Syntactic}}$ and $L^{\text{Recurrent}}$. The reason is that $L^{\text{Syntactic}}$ and $L^{\text{Recurrent}}$ need to restart the learning of all progress automata from scratch when the leading automaton has been modified. (3) Tree-based algorithms always solve more learning tasks than their table-based counterpart. In particular, the tree-based $L^{\text{Syntactic}}$ with the under-approximation method solves all 295 learning tasks.

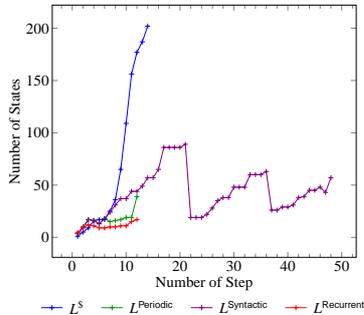


Fig. 7. Growth of state counts in BA

learning algorithm using one learning task that cannot be solved by L^S within the timeout period. The figures of the other three learning tasks show the same trend and

In the experiment, we observe that table-based L^S has 4 cases cannot be finished within the timeout period, which is the largest number amount all learning algorithms⁵. We found that for these 4 cases, the average time required for L^S to get an equivalence query result is much longer than the FDFA algorithms. Under scrutiny, we found that the growth rate of the size (number of states) of the conjectured BAs generated by table-based L^S is much faster than that of table-based FDFA learning algorithms. In Fig. 7, we illustrate the growth rate of the size (number of states) of the BAs generated by each table-based

⁵ Most of the unsolved tasks using the over-approximation method are caused by the situation that the FDFA teacher cannot find a valid counterexample for refinement.

hence are omitted. Another interesting observation is that the sizes of BAs generated by $L^{\text{Syntactic}}$ can decrease in some iteration because the leading automaton is refined and thus the algorithms have to redo the learning of all progress automata from scratch.

It is a bit surprise to us that, in our experiment, the size of BAs \overline{B} produced by the over-approximation method is not much smaller than the BAs \underline{B} produced by the under-approximation method. Recall that the progress automata of \overline{B} comes from the product of three DFAs $M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ while those for \underline{B} comes from the product of only two DFAs $M_u^u \times (A^u)_v^{s_u}$ (Sec. 6). We found the reason is that very often the language of the product of three DFAs is equivalent to the language of the product of two DFAs, thus we get the same DFA after applying DFA minimizations. Nevertheless, the over-approximation method is still helpful for L^{Periodic} and $L^{\text{Recurrent}}$. For L^{Periodic} , the over-approximation method solved more learning tasks than the under-approximation method. For $L^{\text{Recurrent}}$, the over-approximation method solved one tough learning task that is not solved by the under-approximation method.

As we mentioned at the end of Sec. 5.2, a possible optimization is to reuse the counterexample and to avoid equivalence query as much as possible. The optimization helps the learning algorithms to solve nine more cases that were not solved before.

10 Discussion and Future works

Regarding our experiments, the BAs from LTL specifications are in general simple; the average sizes of the learned BAs are around 10 states. From our experience of applying DFA learning algorithms, the performance of tree-based algorithm is significantly better than the table-based one when the number of states of the learned DFA is large, say more than 1000. We believe this will also apply to the case of BA learning. Nevertheless, in our current experiments, most of the time is spent in answering equivalence queries. One possible direction to improve the scale of the experiment is to use a PAC (probably approximately correct) BA teacher [8] instead of an exact one, so the equivalence queries can be answered faster because the BA equivalence testing will be replaced with a bunch of BA membership testings.

There are several avenues for future works. We believe the algorithm and library of learning BAs should be an interesting tool for the community because it enables the possibility of many applications. For the next step, we will investigate the possibility of applying BA learning to the problem of reactive system synthesis, which is known to be a very difficult problem and learning-based approach has not been tried yet.

There are learning algorithms for residual NFA [11], which is a more compact canonical representation of regular languages than DFA. We think maybe one can also generalize the learning algorithm for family of DFAs to family of residual NFAs (FRNFA). To do this, one needs to show FRNFAs also recognize ω -regular language and finds the corresponding right congruences.

References

1. F. Aarts, B. Jonsson, J. Uijen, and F. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *FMSD*, 46(1):1–41, 2015.

2. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *CAV*, pages 132–147, 2010.
3. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-Based Büchi Automata Inclusion Testing. In *CONCUR*, pages 187–202, 2011.
4. B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.
5. R. Alur, P. Černý, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. In *POPL*, pages 98–109, 2005.
6. R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *CAV*, pages 548–562, 2005.
7. D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
8. D. Angluin. Queries and Concept Learning. *Mach. Learn.*, 2(4):319–342, Apr. 1988.
9. D. Angluin, U. Boker, and D. Fisman. Families of DFAs as Acceptors of omega-Regular Languages. In *MFCS*, pages 11:1–11:14, 2016.
10. D. Angluin and D. Fisman. Learning Regular Omega Languages. In *ALT*, pages 125–139, 2014.
11. B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style Learning of NFA. In *IJCAI*, pages 1004–1009, 2009.
12. B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. libalf: The Automata Learning Framework. In *CAV*, pages 360–364, 2010.
13. J. R. Büchi. On a decision method in restricted second order arithmetic. volume 44, pages 1–11, 1966.
14. H. Calbrix, M. Nivat, and A. Podolski. Ultimately Periodic Words of Rational ω -Languages. In *MFPS*, pages 554–566, 1994.
15. S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *CAV*, pages 534–547, 2005.
16. M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the Language of Error. In *ATVA*, pages 114–130, 2015.
17. Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Learning minimal separating DFA’s for compositional verification. In *TACAS*, pages 31–45, 2009.
18. Y.-F. Chen, C. Hsieh, O. Lengál, T.-J. Lii, M.-H. Tsai, B.-Y. Wang, and F. Wang. PAC Learning-based Verification and Model Synthesis. In *ICSE*, pages 714–724, 2016.
19. J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.
20. A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In *TACAS*, pages 2–17, 2008.
21. L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *ICSE*, pages 2–17, 2011.
22. D. Giannakopoulou, Z. Rakamarić, and V. Raman. Symbolic learning of component interfaces. In *SAS*, pages 248–264, 2012.
23. O. Grumberg and Y. Meller. Learning-Based Compositional Model Checking of Behavioral UML Systems. *Dependable Software Systems Engineering*, 45:117, 2016.
24. A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In *FASE*, pages 80–95, 2002.
25. F. He, X. Gao, B. Wang, and L. Zhang. Leveraging Weighted Automata in Compositional Reasoning about Concurrent Probabilistic Systems. In *POPL*, pages 503–514, 2015.
26. F. Howar, D. Giannakopoulou, and Z. Rakamarić. Hybrid learning: interface generation through static, dynamic, and symbolic analysis. In *ISSTA*, pages 268–279, 2013.

27. F. Howar, B. Steffen, B. Jonsson, and S. Cassel. Inferring Canonical Register Automata. In *VMCAI*, pages 251–266, 2012.
28. M. Isberner, F. Howar, and B. Steffen. Learning register automata: from languages to program structures. *Machine Learning*, 96(1-2):65–98, 2014.
29. M. Isberner, F. Howar, and B. Steffen. The open-source LearnLib. In *CAV*, pages 487–495, 2015.
30. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.
31. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *POPL*, pages 81–92, 2001.
32. S.-W. Lin, E. André, Y. Liu, J. Sun, and J. S. Dong. Learning assumptions for compositional verification of timed systems. *IEEE Transactions on Software Engineering*, 40(2):137–153, 2014.
33. O. Maler and L. Staiger. On Syntactic Congruences for Omega-Languages. In *STACS*, pages 586–594, 1993.
34. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.
35. R. L. Rivest and R. E. Schapire. Inference of Finite Automata Using Homing Sequences. In *STOC*, pages 411–420, 1989.
36. J. Sun, H. Xiao, Y. Liu, S.-W. Lin, and S. Qin. TLV: abstraction through testing, learning, and validation. In *FSE*, pages 698–709, 2015.
37. Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, and Y.-W. Chang. Büchi Store: An Open Repository of Büchi Automata. In *TACAS*, pages 262–266, 2011.
38. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS*, pages 322–331, 1986.
39. F. Wang, J. Wu, C. Huang, and K. Chang. Evolving a Test Oracle in Black-Box Testing. In *FASE*, pages 310–325, 2011.
40. H. Xiao, J. Sun, Y. Liu, S.-W. Lin, and C. Sun. Tzuyu: Learning stateful tpestates. In *ASE*, pages 432–442, 2013.

Appendix

In this section, we first show that although our acceptance condition defined in Sec. 2 is different from the original one defined in [10], but the ultimately periodic words of the FDFAs will be preserved. Then, we give the refinement for the progress trees of syntactic and recurrent FDFAs in Sec. B. In Sec. C, we present the proofs of the lemmas given in Sec. 6. In Sec. D, we provide the constructions for the FAs $\mathcal{D}_{u\$v}$, \mathcal{D}_1 and \mathcal{D}_2 as well as the correctness proof of counterexample analysis. We also give the correctness proof and complexity of the tree-based learning algorithm in Sec. E.

A Language Preservation under Different Acceptance Conditions

Recall that the original acceptance condition for periodic FDFAs in [10] is that (u, v) is accepted by \mathcal{F} if $v \in L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. While the original acceptance conditions for syntactic and recurrent FDFAs in [10] are the same as the one defined in this paper. More specifically, (u, v) is accepted by \mathcal{F} if $M(uv) = M(u)$ and $v \in L(A^{M(u)})$. The set of ultimately periodic words of an FDFAs \mathcal{F} is defined as $UP(\mathcal{F}) = \{uv^\omega \mid (u, v) \text{ is accepted by } \mathcal{F}\}$. The acceptance condition for periodic FDFAs used in this paper is different from the original one in [10]. We prove that the acceptance condition does not change the ultimately periodic words of the periodic FDFAs.

Lemma 6. *Let \mathcal{F} be a periodic (syntactic, recurrent) FDFAs under the acceptance condition in [10], then $UP(\mathcal{F})$ is preserved under the acceptance condition defined in this paper.*

Proof. We only need to prove the preservation of ultimately periodic words for the periodic FDFAs. Given a periodic FDFAs \mathcal{F} , the original acceptance condition of periodic FDFAs requires that (u, v) is accepted by \mathcal{F} if $v \in L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. Clearly, the acceptance condition defined in this paper implies the original acceptance condition for the periodic FDFAs. Therefore, we only need prove that if (u, v) satisfies the original acceptance condition, then there exists some decomposition (x, y) of ω -word uv^ω which satisfies our acceptance condition. To achieve this, we first find a normalized formalization (x, y) of (u, v) such that $x = uv^i, y = v^j$ and $xy \sim_M x$ for some $i \geq 0, j \geq 1$ according to [10]. Further, it is known that periodic FDFAs is *saturated* in the sense that under the original acceptance condition, if (u, v) is accepted by \mathcal{F} , then every decomposition of uv^ω is accepted by \mathcal{F} . Therefore we have that (x, y) is accepted by \mathcal{F} , which means that $y \in L(A^{\tilde{x}})$ where $\tilde{x} = M(x)$. It follows that (x, y) is accepted by \mathcal{F} under our acceptance condition. ■

We remark that in [10], they also define an acceptance condition called *normalized acceptance condition*, which is able to make the syntactic and recurrent FDFAs saturated in the sense that if (u, v) is accepted by the FDFAs, then every decomposition of uv^ω is accepted by the FDFAs. Since our goal is to learn a BA in this paper, we do not require the saturation property for all decompositions of accepted ω -word. Thus, we do not use the normalized acceptance condition.

B Refinement of the Progress Trees

Suppose $\tilde{u} \cdot v^\omega \notin UP(L)$ for negative counterexample (u, v) , we thus need refine the progress tree $\mathcal{T}_{\tilde{u}}$. Let $|v| = n$ and $h = s_0 s_1 \cdots s_n$ be the corresponding run of v over $A^{\tilde{u}}$. At the beginning, we have $s_0 = \epsilon$ and $s_n = \tilde{v}$ where $\tilde{v} = A^{\tilde{u}}(v)$ and \tilde{v} is an accepting state in $A^{\tilde{u}}$, which implies that $\tilde{u}(\tilde{v})^\omega \in UP(L)$. Our job here is to find the smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}, v[j \cdots n]) \neq \mathbf{TE}(s_j, v[j+1 \cdots n])$ so that we can use the experiment $e = v[j+1 \cdots n]$ to differentiate $pa = s_{j-1}v[j]$ and $q = s_j$ since currently $s_j = \delta(s_{j-1}, v[j])$.

Afterwards, the progress tree $\mathcal{T}_{\tilde{u}}$ can be refined by replacing the terminal node labeled with s_j by a tree such that (i) its root is labeled by $e = v[j+1 \cdots n]$, (ii) its $\mathbf{TE}(s_j, v[j+1 \cdots n])$ -subtree is a terminal node labeled by s_j , and (iii) its $\mathbf{TE}(s_{j-1}v[j], v[j+1 \cdots n])$ -subtree is a terminal node labeled by $s_{j-1}v[j]$.

In order to establish above result, we have to prove that $\mathbf{TE}(s_0, v) \neq \mathbf{TE}(s_n, \epsilon)$ to ensure that there exists some $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}, v[j \cdots n]) \neq \mathbf{TE}(s_j, v[j+1 \cdots n])$. The proof is as follows.

- For periodic FDFA, we have $\mathbf{TE}(\epsilon, v) = \text{F}$ since $\tilde{u}(\epsilon \cdot v)^\omega \notin UP(L)$. Since \tilde{v} is an accepting state, we have $\mathbf{TE}(\tilde{v}, \epsilon) = \text{T}$.
- For syntactic FDFA, we notice that the counterexample requires $uv \sim_M u$, that is, $\tilde{u} = M(uv) = M(u) = M(\tilde{u}v)$. First, we have $\mathbf{TE}(\epsilon, v) = (M(\tilde{u} \cdot \epsilon), \text{B}) = (\tilde{u}, \text{B})$, where B is obtained here since $\tilde{u} = M(\tilde{u} \cdot \epsilon \cdot v)$ and $\tilde{u}(\epsilon \cdot v)^\omega \notin UP(L)$ according to the definition of \mathbf{TE} in syntactic FDFA. Since \tilde{v} is an accepting state in syntactic FDFA, it follows that $\tilde{u} = M(\tilde{u}\tilde{v})$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Def. 2. Thus, we have $\mathbf{TE}(\tilde{v}, \epsilon) = (M(\tilde{u}\tilde{v}), \text{A}) = (\tilde{u}, \text{A})$ where A is obtained since $\tilde{u} = M(\tilde{u} \cdot \tilde{v} \cdot \epsilon)$ and $\tilde{u}(\tilde{v} \cdot \epsilon)^\omega \in UP(L)$.
- For recurrent FDFA, similar as in syntactic FDFA, we have $\mathbf{TE}(\epsilon, v) = \text{F}$ and $\mathbf{TE}(\tilde{v}, \epsilon) = \text{T}$.

We remark that, if the target is syntactic or recurrent FDFA, as long as the leading automaton M changes, we need to initialize the classification tree \mathcal{T}_u again for every state u in leading automaton since the labels on the edges depend on current leading automaton M .

C Proofs of Lem. 3, Lem 4 and Lem 5

Lemma 4. *Given an FDFA $\mathcal{F} = (M, \{A^u\})$, and \underline{B} the BA constructed from \mathcal{F} by Def. 4. If (u, v^k) is accepted by \mathcal{F} for every $k \geq 1$, then $uv^\omega \in UP(L(\underline{B}))$.*

Proof. From the assumption, we have $uv^k \sim_M u$ and $v^k \in L(A^{\tilde{u}})$ for any $k \geq 1$ where $\tilde{u} = M(u)$. It must be the case that some accepting state, say f in $A^{\tilde{u}}$, will be visited twice after we read v^n from initial state for some $n > |A^{\tilde{u}}|$ with $f = A^{\tilde{u}}(v^n)$ since $A^{\tilde{u}}$ is a finite automaton. In other words, there is a loop in the run of v^n over $A^{\tilde{u}}$. Without loss of generality, suppose there exist $i, j \geq 1$ with $i + j = n$ such that $f = A^{\tilde{u}}(v^i) = A^{\tilde{u}}(v^{i+j})$.

In the following, our goal is to find some accepting state f' such that $f' = A^{\tilde{u}}(v^k) = A^{\tilde{u}}(v^{2k})$ for some $k \geq 1$. Fig. 8 depicts how to find the accepting state f' along the loop path in following two cases.

- $j \geq i$. Let $k = j$.
- $j < i$. Let $k = l \times j$ such that $k \geq i$ with the smallest $l \geq 1$.

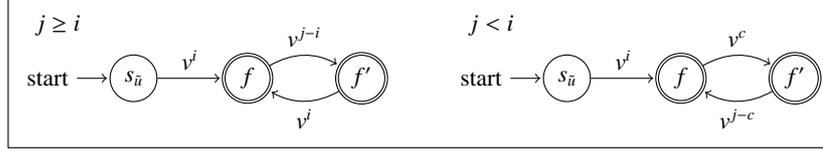


Fig. 8. Finding v^k . If $j \geq i$, we let $k = j$, otherwise let $c = (l \cdot j - i) \% j \geq 0$ where $k = l \cdot j \geq i$ for some $l \geq 1$

It is easy to check that $A^{\tilde{u}}(v^k) = A^{\tilde{u}}(v^{2k})$ since progress automaton $A^{\tilde{u}}$ is deterministic and the corresponding f' is an accepting state.

It follows that v^k is accepted by the product $\underline{P}_{(\tilde{u}, f')}$ of three automata $M_{\tilde{u}}^{\tilde{u}}$, $(A^{\tilde{u}})_{f'}^{s_{\tilde{u}}}$ and $(A^{\tilde{u}})_{f'}^{f'}$ where $s_{\tilde{u}}$ is the initial state of $A^{\tilde{u}}$. In other words, ω -word uv^ω will be accepted in \underline{B} since $u \cdot (v^k)^\omega \in L(M_{\tilde{u}}^{q_0}) \cdot (L(\underline{P}_{(\tilde{u}, f')}))^\omega$. ■

Lemma 5. Given an ω -word $w \in UP(L(\overline{B}))$, there exists a decomposition (u, v) of w and $n \geq 1$ such that $v = v_1 \cdot v_2 \cdots v_n$ and for all $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \sim_M u$.

Proof. Here we only consider ultimately periodic ω -words in \overline{B} , so every ω -word can be given by a decomposition.

Since $UP(L(\overline{B})) = \bigcup_{u \in Q, p \in F_u} L(M_u^{q_0}) \cdot (L(\overline{P}_{(u, p)}))^\omega$, suppose ω -word $w = uv^\omega \in UP(L(\overline{B}))$, then w can be given by a decomposition (u, v) such that $u \in L(M_u^{q_0})$ and $v \in (L(\overline{P}_{(\tilde{u}, p)}))^\omega$ for some $p \in F_{\tilde{u}}$ where $\tilde{u} = M(u)$. Thus, we have $v = v_1 \cdots v_n$ for some $n \geq 1$ such that $v_i \in L(\overline{P}_{(\tilde{u}, p)})$ for every $1 \leq i \leq n$. In addition, since $\overline{P}_{(\tilde{u}, p)} = M_{\tilde{u}}^{\tilde{u}} \times (A^{\tilde{u}})_p^{s_{\tilde{u}}}$, we conclude that $uv \sim_M u$ and $v_i \in L((A^{\tilde{u}})_p^{s_{\tilde{u}}})$ for every $1 \leq i \leq n$ where $s_{\tilde{u}}$ is the initial state in $A^{\tilde{u}}$.

Observe that p is the only accepting state of $(A^{\tilde{u}})_p^{s_{\tilde{u}}}$ and $(A^{\tilde{u}})_p^{s_{\tilde{u}}}$ is obtained from $A^{\tilde{u}}$ by setting $p \in F_{\tilde{u}}$ as its only accepting state, we have that $p = (A^{\tilde{u}})_p^{s_{\tilde{u}}}(v_i) = A^{\tilde{u}}(v_i)$ for every $1 \leq i \leq n$ and p is an accepting state in $A^{\tilde{u}}$.

The remaining job is how to find the accepting state p in $A^{\tilde{u}}$. Suppose we have the counterexample uv^ω given by the decomposition (u, v) , from which we construct the FA $\mathcal{D}_{u\$v}$ by the method in Sec. D.1. The number of states in $\mathcal{D}_{u\$v}$ is in $O(|v|(|v| + |u|))$. In addition, we can construct an FA \mathcal{A} such that $L(\mathcal{A}) = \bigcup_{u \in Q, p \in F_u} L(M_u^{q_0}) \cdot \$ \cdot (L(M_u^u \times (A^u)_p^{s_u}))^\omega$ where s_u is the initial state of A^u . By fixing u and p , we get $L(\mathcal{A}_{(u, p)}) = L(M_u^{q_0}) \cdot \$ \cdot (L(M_u^u \times (A^u)_p^{s_u}))^\omega = L(M_u^{q_0}) \cdot \$ \cdot (L(\overline{P}_{(u, p)}))^\omega$. We get the corresponding u and p such that $L(\mathcal{A}_{(u, p)}) \times \mathcal{D}_{u\$v} \neq \emptyset$. There must exist such u and p otherwise uv^ω will not be accepted by \overline{B} . To get all the fragment words v_i from v , one only needs to run the finite word v over $\overline{P}_{(u, p)}$. The time and space complexity of this procedure are in $O(nk(n + nk) \cdot (|v|(|v| + |u|)))$ and $O((n + nk) \cdot (|v|(|v| + |u|)))$ respectively where n is the number of states in the leading automaton and k the number of states in the largest progress automaton. Thus we complete the proof. ■

Lemma 3 (Sizes and Languages of \underline{B} and \overline{B}). Let \mathcal{F} be an FDFFA and \underline{B} , \overline{B} be the BAs constructed from \mathcal{F} by Def. 4. Let n and k be the numbers of states in the leading automaton and the largest progress automaton of \mathcal{F} . The number of states of \underline{B} and \overline{B} are in $O(n^2k^3)$ and $O(n^2k^2)$, respectively. Moreover, $UP(L(\underline{B})) \subseteq UP(\mathcal{F}) \subseteq UP(L(\overline{B}))$ and we have $UP(L(\underline{B})) = UP(\mathcal{F})$ when \mathcal{F} is a canonical FDFFA.

Proof. In the following, we prove the lemma by following cases.

- Sizes of \underline{B} and \overline{B} . In the under approximation construction, for every state u in M , there is a progress automaton A^u of size at most k . It is easy to conclude that the automaton $\underline{P}_{(u,v)}$ is of size nk^2 for every $v \in F_u$, so \underline{B} is of size $n + nk \cdot nk^2 \in O(n^2k^3)$. The over-approximation method differs in the construction of the automaton $\overline{P}_{(u,v)}$ from the under-approximation method. It is easy to conclude that the automaton $\overline{P}_{(u,v)}$ is of size nk for every $v \in F_u$, so \overline{B} is of size $n + nk \cdot nk \in O(n^2k^2)$.
- $UP(L(\underline{B})) \subseteq UP(\mathcal{F})$. Suppose ultimately periodic ω -word w is accepted by \underline{B} , there must be an accepting run in \underline{B} in form of $q_0 \xrightarrow{u} \tilde{u} \xrightarrow{\epsilon} s_{\tilde{u},v} \xrightarrow{v_1} f_v \xrightarrow{\epsilon} f'_v \xrightarrow{\epsilon} s_{\tilde{u},v} \cdots$. Then the ω -word w can be divided into the form of $u \cdot \epsilon \cdot v_1 \cdot \epsilon \cdot \epsilon \cdot v_2 \cdots$ by ϵ -transitions. According to the construction of \underline{B} , we have $u \in L(M_u^{q_0})$ and $v_i \in L(\overline{P}_{(\tilde{u},v)})$ for any $i \geq 1$. Moreover, since $\overline{P}_{(\tilde{u},v)}$ is the product of three automata $M_{\tilde{u}}^{\tilde{u}}$, $(A^{\tilde{u}})_v^{s_{\tilde{u},v}}$ and $(A^{\tilde{u}})_v^{v}$ where $s_{\tilde{u}}$ is the initial state in $A^{\tilde{u}}$. It follows that $L(M_u^{q_0}) \cdot (L(\overline{P}_{(\tilde{u},v)}))^* = L(M_u^{q_0})$ and $(L(\overline{P}_{(\tilde{u},v)}))^+ = L(\overline{P}_{(\tilde{u},v)})$. By Lem.5 in [14], there exist two words $x \in L(M_u^{q_0})$ and $y \in L(\overline{P}_{(\tilde{u},v)})$ such that $w = x \cdot y^\omega$. In other words, we have $\tilde{u} = M(x)$, $xy \sim_M x$ and $y \in L(A^{\tilde{u}})$, which implies that w is accepted by \mathcal{F} .
- $UP(\mathcal{F}) \subseteq UP(L(\overline{B}))$. Suppose an ω -word $w \in UP(\mathcal{F})$, then there exists a decomposition (u, v) of w such that $uv \sim_M u$ and \tilde{v} is an accepting state where $\tilde{u} = M(u)$ and $\tilde{v} = A^{\tilde{u}}(v)$. It follows that $v \in L(\overline{P}_{(\tilde{u},\tilde{v})})$ according to Def. 4. In addition, we have $u \in L(M_u^{q_0})$, which follows that $u \cdot v^\omega \in L(M_u^{q_0}) \cdot (L(\overline{P}_{(\tilde{u},\tilde{v})}))^\omega = UP(L(\overline{B}))$.
- $UP(L(\underline{B})) = UP(\mathcal{F})$ if \mathcal{F} is a canonical FDFFA. For any FDFFA \mathcal{F} , we have $UP(L(\underline{B})) \subseteq UP(\mathcal{F})$. Thus, the remaining job is to prove that $UP(\mathcal{F}) \subseteq UP(L(\underline{B}))$ if \mathcal{F} is a canonical FDFFA, which follows from Prop. 1 and Lem. 4. Thus, we complete the proof. ■

We present Prop. 1, which follows from Def. 2 of the canonical FDFAs.

Proposition 1. Let L be an ω -regular language, $\mathcal{F} = (M, \{A^u\})$ the corresponding periodic (syntactic, recurrent) FDFFA and $u, v \in \Sigma^*$. We have that if (u, v) is accepted by \mathcal{F} then (u, v^k) is also accepted by \mathcal{F} for any $k \geq 1$.

Proof. Let $\tilde{u} = M(u)$ and $\tilde{v}^k = A^{\tilde{u}}(v^k)$, then we have that $v^k \approx_K^{\tilde{u}} \tilde{v}^k$ for every $k \geq 1$ where $K \in \{P, S, R\}$. This is because $\tilde{v}^k = A^{\tilde{u}}(v^k) = A^{\tilde{u}}(v^k)$ which makes v^k in the equivalence class $[\tilde{v}^k]$. Our goal is to prove that (u, v^k) is also accepted by \mathcal{F} , that is, $uv^k \sim_M u$ and \tilde{v}^k is an accepting state for every $k \geq 1$. Since \sim_M and \sim_L is consistent in the three canonical FDFAs, so from the fact that (u, v) is accepted by \mathcal{F} , we have that $uv \sim_M u$, i.e., $uv \sim_L u$. It follows that $uv^k \sim_L u$ for every $k \geq 1$. Thus, the remaining proof is to prove that \tilde{v}^k is an accepting state for every $k \geq 1$ in the three canonical FDFAs.

- For periodic FDFA, since (u, v) is accepted by \mathcal{F} , i.e., \tilde{v} is an accepting state in $A^{\tilde{u}}$, then we have $\tilde{u}(\tilde{v})^\omega \in L$ according to Def. 2. By definition of $\approx_P^{\tilde{u}}$ and the fact that $\tilde{v} \approx_P^{\tilde{u}} v$, we have that $\tilde{u}(v)^\omega \in L$, i.e., $\tilde{u}(v^k)^\omega \in L$ for every $k \geq 1$. Similarly, since $\tilde{u}(v^k)^\omega \in L$ and $v^k \approx_P^{\tilde{u}} \tilde{v}^k$, we conclude that $\tilde{u}(\tilde{v}^k)^\omega \in L$, which means that the state \tilde{v}^k is an accepting state in $A^{\tilde{u}}$ for every $k \geq 1$.
- By the definition of $\approx_R^{\tilde{u}}$, if $x \approx_R^{\tilde{u}} y$, then we have $\tilde{u}x \sim_L \tilde{u} \wedge \tilde{u}x^\omega \in L \iff \tilde{u}y \sim_L \tilde{u} \wedge \tilde{u}y^\omega \in L$ for any $x, y \in \Sigma^*$. Since $x \approx_S^{\tilde{u}} y$ implies $x \approx_R^{\tilde{u}} y$, we also have above result if $x \approx_S^{\tilde{u}} y$. In the following, $\approx_K^{\tilde{u}}$ can be replaced by $\approx_S^{\tilde{u}}$ and $\approx_R^{\tilde{u}}$.
For syntactic FDFA and recurrent FDFA, if (u, v) is accepted by \mathcal{F} , then $\tilde{u}\tilde{v} \sim_L \tilde{u}$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Def. 2. By the fact that $v \approx_K^{\tilde{u}} \tilde{v}$, if we set $x = v$ and $y = \tilde{v}$, then we have that $\tilde{u}v \sim_L \tilde{u}$ and $\tilde{u}(v)^\omega \in L$, which implies that $\tilde{u}v^k \sim_L \tilde{u}$ and $\tilde{u}(v^k)^\omega \in L$ for every $k \geq 1$.
Similarly, as $v^k \approx_K^{\tilde{u}} \tilde{v}^k$, if we set $x = v^k$ and $y = \tilde{v}^k$, we have that $\tilde{u}v^k \sim_L \tilde{u}$ and $\tilde{u}(v^k)^\omega \in L$, which follows that \tilde{v}^k is an accepting state in $A^{\tilde{u}}$ for every $k \geq 1$. ■

D Finite Automaton Construction and Correctness for Counterexample Analysis

D.1 Construction for $\mathcal{D}_{u\$v}$

In [14], they presented a canonical representation $L_\$ = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$ for a regular ω -language L . Theoretically, we can apply their method to obtain the $\mathcal{D}_{u\$v}$ automaton for an ω -word uv^ω where the number of states in $\mathcal{D}_{u\$v}$ is in $O(2^{|u|+|v|})$. In this section, we introduce a more effective way to build an automaton $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega = w\}$ for a given ω -word w with the number of states in $O(|v|(|v| + |u|))$. A similar construction for $\mathcal{D}_{u\$v}$ has been proposed in [20], which first computes the regular expression to represent all possible decompositions of uv^ω and then constructs a DFA from the regular expression. In this section, we give a direct construction for $\mathcal{D}_{u\$v}$ of uv^ω as well as the complexity of the construction.

Fig. 9 depicts an example automaton $\mathcal{D}_{u\$v}$ for ω -word $(ab)^\omega$. From the example, we can find that both decompositions (aba, ba) and $(ababa, bababa)$ have the same periodic word $(ba)^\omega$, which means that the second finite word of a decomposition can be simplified as long as we do not change the periodic word.

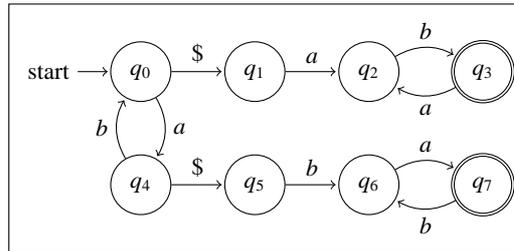


Fig. 9. $\mathcal{D}_{u\$v}$ for ω -word (aba, ba)

Formally, we give the definition of a *smallest period* in an ω -word w given by its decomposition (u, v) where $v \in \Sigma^+$. To that end, we need more notations. We use $u \preceq v$ to represent that there exists some $j \geq 1$ such that $u = v[1 \cdots j]$, and we say u is a prefix of v . We use $u \triangleleft v$ if $u \preceq v$ and $u \neq v$.

Definition 6 (Smallest period). For any ω -word w given by (u, v) , we say r is the *smallest period* of (u, v) if $r \preceq v$, $r^\omega = v^\omega$ and for any $t \triangleleft r$, we have $t^\omega \neq r^\omega$.

Take the ω -word $(ab)^\omega$ as an example, ab and ba are the smallest periods of decomposition (ab, ab) and (aba, ba) respectively. It is interesting to see that $|ab| = |ba|$ and ab can be transformed to ba by shift the first letter of ab to its tail. In general, given ω -word w , the length of the smallest period is fixed no matter how w is decomposed which is justified by Lem. 7.

Lemma 7. Given an ω -word w , (u, v) and (x, y) are different decompositions of w and their corresponding smallest periods are r and t , respectively. Then $|r| = |t| = n$ and either there exists $j \geq 2$ such that $r = t[j \cdots n] \cdot t[1 \cdots j - 1]$ or $r = t$.

Proof. According to Def. 6, $w = uv^\omega = ur^\omega = xy^\omega = xt^\omega$. We prove it by contradiction. Without loss of generality, suppose $|r| > |t|$. If $|u| = |x|$, then $r^\omega = t^\omega$, we then conclude that r is not a smallest period of (u, v) since $t \triangleleft r$. Otherwise if $|u| \neq |x|$, we can either prove that $r = t$ or find some $j \geq 2$ such that $z = t[j \cdots n] \cdot t[1 \cdots j - 1] \triangleleft r$ and $z^\omega = r^\omega$ in following cases.

- $|u| > |x|$. Let $k = (|u| - |x|) \% |t| + 1$. If $k = 1$, then $z = t$, otherwise $j = k$;
- $|x| > |u|$. Let $k = (|r| - (|x| - |u|) \% |r|) \% |t| + 1$. If $k = 1$, then $z = t$, otherwise $j = k$;

We depict the situation where $|u| > |x|$ in the following.

$$\begin{array}{ll} (u, r) & u[1]u[2] \cdots u[k]u[k+1] \cdots u[m] \cdot r \cdot r \cdot r \cdots \\ (x, t) & x[1]x[2] \cdots x[k]t[1] \cdots t[j-1] \cdot z \cdot z \cdot z \cdots \end{array}$$

From the assumption $|t| < |r|$, we have that $z \triangleleft r$. However, since $z^\omega = r^\omega$, we conclude that r is not the smallest period of (u, v) . Contradiction. Thus we complete the proof. ■

Lem. 7 shows that if the size of the smallest period of an ω -word w is n , then there are exactly n different smallest periods for w . In the following, we define the shortest form for a decomposition of an ω -word.

Lemma 8. For any decomposition (u, v) of an ω -word w , and y is its corresponding smallest period, then we can rewrite $u = xy^i$ and $v = y^j$ for some $i \geq 0$, $j \geq 1$ such that for any $x' \preceq u$ with $u = x'y^k$ for some $0 \leq k \leq i$, we have $x' = xy^{i-k}$. We say such (x, y) is the shortest form for (u, v) .

Proof. This can be proved by Def. 6 and the fact that $y^\omega = v^\omega$, which can be further illustrated by the procedure of constructing (x, y) . To find the shortest form of (u, v) , we need to first find the smallest period y of (u, v) , which is illustrated by following procedure. At first we initialize $k = 1$.

- Step 1. Let $y = v[1 \cdots k]$, we recursively check whether there exists some $j \geq 1$ such that $v = y^j$. If there exists such j , we return y as the smallest period. Otherwise we go to Step 2.
- Step 2. We increase k by 1 and go to Step 1.

Since k starts at 1, then y must be the smallest period of (u, v) such that $v^\omega = y^\omega$.

We find the above x of the shortest form in the following procedure.

- Step 1. Let $x = u$. If $x = \epsilon$, or $x = y$ then we return ϵ . Otherwise we check whether there exists some $k \geq 1$ such that $x = x[1 \cdots k] \cdot x[k+1 \cdots |x|]$ and $y = x[k+1 \cdots |x|]$. If there is no such k , we return x as the final result. Otherwise we go to Step 2.
- Step 2. We set $u = x[1 \cdots k]$.

One can easily conclude that x is the shortest prefix of u such that $u = xy^i$ for some $i \geq 0$. ■

Following corollary is straightforward.

Corollary 1. *Given two decompositions (u_1, v_1) and (u_2, v_2) of uv^ω . If (u_1, v_1) and (u_2, v_2) share the smallest period y , then they also have the same shortest form (x, y) where $u_1 = xy^i, u_2 = xy^j$ for some $i, j \geq 0$.*

Proof (Sketch). If we assume they have different shortest forms, they should not be two decompositions of the same ω -word. ■

By Coro. 1, we can represent all decompositions of an ω -word w which share the same smallest period y with (xy^i, y^j) with some $i \geq 0, j \geq 1$. In addition, since the number of different smallest periods is $|y|$, we can thus denote all the decompositions of w by the set $\bigcup_{k=1}^{|y|} \{(x_k y_k^i, y_k^j) \mid i \geq 0, j \geq 1\}$ where (x_k, y_k) is the k -th shortest form of w . Therefore, we provide the construction of $\mathcal{D}_{u\$v}$ as follows.

Construction of $\mathcal{D}_{u\$v}$ Now we are ready to give the construction of $\mathcal{D}_{u\$v}$ for a single ω -word w given by (u, v) . Suppose (x, y) is the shortest form of (u, v) , then we construct $\mathcal{D}_{u\$v}$ as follows. Let $k = 1, n = |y|$, and we first construct an automaton D_1 such that $L(D_1) = xy^* \$y^+$.

- Step 1. If $k = n$, then we construct the $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \bigcup_{i=1}^n L(D_i)$, otherwise, we go to Step 2.
- Step 2. We first increase k by 1. Let $u' = x \cdot y[1]$ and $y' = y[2 \cdots n] \cdot y[1]$. We then get the shortest form (x', y') of (u', y') where the second element is y' since y' is the smallest period of (u', y') according to Lem. 7. We then construct an automaton D_k such that $L(D_k) = x' y'^* \$y'^+$ and let $x = x', y = y'$ and go to Step 1.

Suppose $|x| = m$ and $|y| = n$, the DFA A that accepts $xy^* \$y^+$ can be constructed as follows.

- If $m = 0$, then we construct a DFA $A = (\Sigma, \{q_0, \dots, q_{2n}\}, q_0, \{q_{2n}\}, \delta)$ where we have that $\delta(q_{k-1}, y[k]) = q_k$ when $1 \leq k \leq n-1$, $\delta(q_{n-1}, y[n]) = q_0$, $\delta(q_0, \$) = q_n$, $\delta(q_{n-1+k}, y[k]) = q_{n+k}$ when $1 \leq k \leq n$, and $\delta(q_{2n}, y[1]) = q_{n+1}$.

- Otherwise $m \geq 1$, then we construct a DFA $A = (\Sigma, \{q_0, \dots, q_{2n+m}\}, q_0, \{q_{m+2n}\}, \delta)$ where we have that $\delta(q_{k-1}, x[k]) = q_k$ when $1 \leq k \leq m$, $\delta(q_{m-1+k}, y[k]) = q_{m+k}$ when $1 \leq k \leq n-1$, $\delta(q_{m+n-1}, y[n]) = q_m$, $\delta(q_m, \$) = q_{m+n}$, $\delta(q_{m+n+k-1}, y[k]) = q_{m+n+k}$ when $1 \leq k \leq n$, and $\delta(q_{m+2n}, y[1]) = q_{m+n+1}$.

One can validate that $L(A) = xy^*\$y^+$ and the number of states in A is at most $|x|+2|y|+1$.

Proposition 2. *Let $\mathcal{D}_{u\$v}$ be the DFA constructed from the decomposition (u, v) of ω -word uv^ω , then $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, u'v'^\omega = uv^\omega\}$.*

Proof.

\subseteq . This direction is easy since $L(\mathcal{D}_{u\$v}) = \bigcup_{i=1}^n L(D_i)$, we only need to prove that for any $1 \leq i \leq n$, if $u'\$v' \in \bigcup_{i=1}^n L(D_i)$, then $u'v'^\omega = uv^\omega$. Suppose $L(D_i) = x_i y_i^* \$y_i^+$, thus for any $u'\$v' \in L(D_i)$, we have $u' = x_i y_i^j$ and $v' = y_i^k$ for some $j \geq 0, k \geq 1$. It follows that $u'v'^\omega = uv^\omega$ since $x_i y_i^\omega = uv^\omega$.

\supseteq . For any decomposition (u', v') of uv^ω , we can get its shortest form (x', y') where y' is the smallest period of (u', v') according to Lem. 8. Suppose (x, y) is the first shortest form used in the $\mathcal{D}_{u\$v}$ construction. By Lem. 7, we prove $u'\$v'$ is accepted by $\mathcal{D}_{u\$v}$ as follows.

- $y = y'$. We have that $u' = xy^i$ and $v' = y^j$ for some $i \geq 0, j \geq 1$, thus $u'\$v' \in L(D_1) \subseteq L(\mathcal{D}_{u\$v})$.
- $y' = y[j \cdots n]y[1 \cdots j-1]$ for some $j \geq 2$. We conclude that $L(D_j) = x'y'^*\$y'^+$ since the shortest form is unique if we fix the smallest period by Coro. 1, which follows that $u'\$v' \in L(D_j) \subseteq L(\mathcal{D}_{u\$v})$.

Therefore, we complete the proof. \blacksquare

Proposition 3. *Given an ω -word w given by (u, v) , then the automaton $\mathcal{D}_{u\$v}$ has at most $O(|v|(|u| + |v|))$ of states.*

For every automaton D_i such that $L(D_i) = xy^*\$y^+$, the number of states in D_i is at most $|u| + 2|r| + 2$ where r is the smallest period of (u, v) , thus the number of states in $\mathcal{D}_{u\$v}$ is in $O(|r| \times (|r| + |u|)) \in O(|v|(|u| + |v|))$.

D.2 Construction of \mathcal{D}_1 and \mathcal{D}_2

In this section, given an FDFA $\mathcal{F} = (M, \{A^u\})$, we provide the constructions for \mathcal{D}_1 and \mathcal{D}_2 . To ease the construction, we define two automata N_u and \overline{N}_u which will be used in the construction for every state u in the leading automaton M . Assume that we have M_u^u , the corresponding progress automaton $A^u = (\Sigma, Q^u, s^u, F^u, \delta^u)$ and a DFA $\overline{A}^u = (\Sigma, Q^u, s^u, Q^u \setminus F^u, \delta^u)$ built from A^u such that $L(\overline{A}^u) = \Sigma^* \setminus L(A^u)$. Note that the transition δ^u is complete in the sense that $\delta^u(s, a)$ is defined for every $s \in Q^u, a \in \Sigma$.

- For \mathcal{D}_1 , we have $N_u = M_u^u \times A^u$. Intuitively, we only keep the finite words which start at u and can go back to u in the leading automaton. In other words, $L(N_u) = \{v \in \Sigma^* \mid uv \sim_M u, v \in L(A^u)\}$.
- For \mathcal{D}_2 , we have $\overline{N}_u = M_u^u \times \overline{A}^u$. Similarly, we have $L(\overline{N}_u) = \{v \in \Sigma^* \mid uv \sim_M u, v \notin L(A^u)\}$.

More precisely, The construction is as follows.

Definition 7. Let $\mathcal{F} = \{M, \{A^u\}\}$ be an FDFA where we have $M = (\Sigma, Q, q_0, \delta)$ and for every $u \in Q$, the corresponding progress automaton $A^u = (\Sigma, Q^u, s^u, F^u, \delta^u)$. Let N_u (and \bar{N}_u) be given by $(\Sigma, Q_u, s_u, F_u, \delta_u)$. The DFA \mathcal{D}_1 (and \mathcal{D}_2) is defined as the tuple $(\Sigma \cup \{\$, Q \cup Q_{Acc}, q_0, F, \delta \cup \delta_{Acc} \cup \delta_{\$})$ where

$$Q_{Acc} = \bigcup_{u \in Q} Q_u \text{ and } F = \bigcup_{u \in Q} F_u \text{ and } \delta_{Acc} = \bigcup_{u \in Q} \delta_u$$

$$\delta_{\$} = \{(u, \$, s_u) \mid u \in Q\}$$

where $\$$ is a fresh symbol.

In Fig. 10, we depict the DFA \mathcal{D}_1 and \mathcal{D}_2 constructed from \mathcal{F} in Fig. 1.

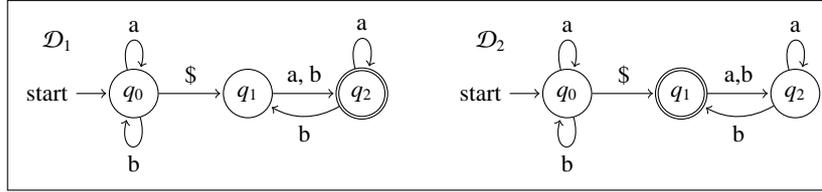


Fig. 10. \mathcal{D}_1 and \mathcal{D}_2 for \mathcal{F} in Fig. 1

Proposition 4. Given an FDFA $\mathcal{F} = (M, \{A^u\})$ and \mathcal{D}_1 defined in Def. 7, then $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \sim_M u, \tilde{u} = M(u), v \in L(A^{\tilde{u}})\}$.

Proof. By Def. 7, it is easy to conclude that for any $u \in \Sigma^*$, then we have $\tilde{u} = M(u) = \mathcal{D}_1(u)$. For any $u, v \in \Sigma^*$, we have that $N_{\tilde{u}}(v) = \mathcal{D}_1(u\$v)$ where $\tilde{u} = M(u)$ since \mathcal{D}_1 is a DFA. By acceptance condition, (u, v) is accepted by \mathcal{F} iff we have $uv \sim_M u$ and $v \in L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. Thus we just need to prove that (u, v) is accepted by \mathcal{F} iff $u\$v$ is accepted by \mathcal{D}_1 .

\supseteq . (u, v) is accepted by \mathcal{F} , then $u\$v \in L(\mathcal{D}_1)$. By $uv \sim_M u$ and $v \in L(A^{\tilde{u}})$, we have that $v \in L(N_{\tilde{u}})$, which follows that $N_{\tilde{u}}(v)$ is an accepting state. Since $N_{\tilde{u}}(v) = \mathcal{D}_1(u\$v)$, we have that $\mathcal{D}_1(u\$v)$ is an accepting state. Therefore, $u\$v \in L(\mathcal{D}_1)$.

\subseteq . First, we have that $L(\mathcal{D}_1) \subseteq \Sigma^*\Σ^* by Def. 7. For any $u, v \in \Sigma^*$, if $u\$v \in L(\mathcal{D}_1)$, then $\mathcal{D}_1(u\$v)$ is an accepting state. It follows that $v \in L(N_{\tilde{u}})$ with $\tilde{u} = M(u)$. Since $N_{\tilde{u}} = M_{\tilde{u}}^{\tilde{u}} \times A^{\tilde{u}}$, we have that $v \in L(M_{\tilde{u}}^{\tilde{u}})$ and $v \in L(A^{\tilde{u}})$, which implies that $uv \sim_M u$ and $v \in L(A^{\tilde{u}})$. Thus, we conclude that (u, v) is accepted by \mathcal{F} . ■

Proposition 5. Given an FDFA \mathcal{F} and \mathcal{D}_2 the corresponding deterministic automaton, then $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \sim_M u, \tilde{u} = M(u), v \notin L(A^{\tilde{u}})\}$.

Proof. By Def. 7, it is easy to conclude that for any $u \in \Sigma^*$, then we have $\tilde{u} = M(u) = \mathcal{D}_2(u)$. For any $u, v \in \Sigma^*$, we have that $\bar{N}_{\tilde{u}}(v) = \mathcal{D}_2(u\$v)$ where $\tilde{u} = M(u)$ since \mathcal{D}_2 is a DFA.

⊇. Assume that we have $uv \sim_M u$ and $v \notin L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. By $uv \sim_M u$, we have that $v \in L(M_{\tilde{u}}^{\tilde{u}})$. Further, from $v \notin L(A^{\tilde{u}})$, we have that $v \in L(\overline{A^{\tilde{u}}})$. It follows that $N_{\tilde{u}}(v)$ is an accepting state. Since $\overline{N_{\tilde{u}}}(v) = \mathcal{D}_2(u\$v)$, then $\mathcal{D}_2(u\$v)$ is an accepting state. Therefore, $u\$v \in L(\mathcal{D}_2)$.

⊆. First, we have that $L(\mathcal{D}_2) \subseteq \Sigma^*\Sigma^*$ by Def. 7. For any $u, v \in \Sigma^*$, if $u\$v \in L(\mathcal{D}_2)$, then $\mathcal{D}_2(u\$v)$ is an accepting state. It follows that $v \in L(\overline{N_{\tilde{u}}})$ with $\tilde{u} = M(u)$. Since $\overline{N_{\tilde{u}}} = M_{\tilde{u}}^{\tilde{u}} \times \overline{A^{\tilde{u}}}$, we have that $v \in L(M_{\tilde{u}}^{\tilde{u}})$ and $v \in L(\overline{A^{\tilde{u}}})$, which implies that $uv \sim_M u$ and $v \notin L(A^{\tilde{u}})$. ■

Proposition 6. *The numbers of states in \mathcal{D}_1 and \mathcal{D}_2 are both in $\mathcal{O}(n + n^2k)$.*

Suppose n is the number of states in M and k is the number of states in the largest progress automaton, then the number of states in \mathcal{D}_1 (\mathcal{D}_2) is in $\mathcal{O}(n + n^2k)$.

D.3 Correctness of Counterexample Analysis for FDFA Teacher

Given the counterexample uv^ω for the FDFA teacher, we prove the decomposition (u', v') is a counterexample for FDFA learner defined in Def. 3 by following cases:

- $uv^\omega \in UP(L) \wedge uv^\omega \notin UP(\mathcal{F})$. By Def. 3, we know that uv^ω is a positive counterexample and we return a counterexample (u', v') such that $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$. We first need to prove that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty. Since $uv^\omega \notin UP(\mathcal{F})$, then any decomposition of uv^ω , say (u, v) , is not accepted by \mathcal{F} . Since M is a DFA, we can always find a decomposition $x = uv^i$ and $y = v^j$ from some $i \geq 0, j \geq 1$ such that $xy \sim_M x$ according to [10]. Therefore (x, y) is also a decomposition of uv^ω and it is not accepted by \mathcal{F} , that is, $y \notin L(A^{\tilde{x}})$ where $\tilde{x} = M(x)$. It follows that $x\$y \in L(\mathcal{D}_2)$ according to Thm. 5. Thus, we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty. We let $u' = x$ and $v' = y$, and it is easy to validate that (u', v') is a positive counterexample for FDFA learner. This case is applied for case U1 and O1.
- $uv^\omega \in UP(L) \wedge uv^\omega \in UP(\mathcal{F})$. In this case, uv^ω is a spurious positive counterexample, which happens when we use the under-approximation method to construct the Büchi automaton. Here we also return a counterexample (u', v') such that $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$. Since $uv^\omega \in UP(\mathcal{F})$, then there exists some decomposition of uv^ω , say (u, v) , is accepted by \mathcal{F} . We observe that $uv^\omega \notin UP(L(\underline{B}))$, which follows that there exists some $k \geq 1$ such that (u, v^k) is not accepted by \mathcal{F} by Lem. 4. By $uv \sim_M u$, we also have that $uv^k \sim_M u$ since M is a DFA. It follows that $u\$v^k \in L(\mathcal{D}_2)$. Therefore, we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$ is not empty and for every finite word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$, we have (u', v') is a positive counterexample for FDFA learner. This case is applied for U3.
- $uv^\omega \notin UP(L) \wedge uv^\omega \in UP(\mathcal{F})$. In this case, uv^ω is a negative counterexample, one has to return a counterexample (u', v') such that $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$. We first need to prove that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$ is not empty. Since $uv^\omega \in UP(\mathcal{F})$, then there exists some decomposition (u', v') of uv^ω is accepted by \mathcal{F} . It follows that $u'\$v' \in L(\mathcal{D}_1)$ by Thm. 4. Thus we conclude that $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$ is not empty. Moreover, it is easy to validate that (u', v') is a negative counterexample for FDFA learner. This case is applied for U2 and O2.

- $uv^\omega \notin UP(L) \wedge uv^\omega \notin UP(\mathcal{F})$. In this case, uv^ω is a spurious negative counterexample, which happens when we use the over-approximation method to construct the Büchi automaton. It is possible that we cannot find a valid decomposition (u', v') to refine \mathcal{F} . According to the proof of Lem. 5, one can construct a decomposition (u, v) of uv^ω and $n \geq 1$ such that $v = v_1 \cdot v_2 \cdots v_n$ and for all $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \sim_M u$. If we find some $i \geq 1$ such that $uv_i^\omega \notin UP(L)$, then we let $u' = u$ and $v' = v_i$. Clearly, (u', v') is a negative counterexample for FDFA learner. This case is applied for O3.

E Correctness and Termination of Tree-based Algorithm

In the following, we need the notion called normalized factorization introduced in [10]. Recall that given a decomposition (u, v) of ω -word uv^ω and the leading automaton M , we can get its normalized factorization (x, y) with respect to M such that $x = uv^i$, $y = v^j$ and $M(xy) = M(x)$ for some smallest $i \geq 0$, $j \geq 1$ since M is finite.

E.1 Correctness of Tree-based Algorithm for FDFA

Lem. 9 establishes the correctness of our tree-based algorithm for periodic FDFA.

Lemma 9. *For the leading tree in all three FDFAs and the progress trees in the periodic FDFA, the tree-based algorithm will never classify two finite words which are in the same equivalence class into two different terminal nodes.*

Proof. We prove by contradiction. Suppose there are two finite word $x_1, x_2 \in \Sigma^*$ which are in the same equivalence class but they are currently classified into different terminal nodes in classification tree \mathcal{T} .

- \mathcal{T} is the leading tree. We assume that $x_1 \sim_L x_2$. Suppose x_1 and x_2 have been assigned to terminal nodes t_1 and t_2 respectively with $t_1 \neq t_2$. Therefore, we can find the least common ancestor n from \mathcal{T} , where $L_n(n) = (y, v)$ is supposed to be an experiment to differentiate x_1 and x_2 . Without loss of generality, we assume that t_1 and t_2 are in the left and right subtrees of n respectively. Therefore, we have $\mathbf{TE}(x_1, (y, v)) = \mathbf{F}$ and $\mathbf{TE}(x_2, (y, v)) = \mathbf{T}$. It follows that $x_1(yv)^\omega \notin UP(L)$ and $x_2(yv)^\omega \in UP(L)$, which implies that $x_1 \not\sim_L x_2$. Contradiction.
- $\mathcal{T} = \mathcal{T}_u$ is a progress tree in periodic FDFA. We assume that $x_1 \approx_P^u x_2$. Similarly, suppose x_1 and x_2 have been assigned to terminal nodes t_1 and t_2 of \mathcal{T}_u respectively with $t_1 \neq t_2$. Therefore, we can find the least common ancestor n from \mathcal{T}_u , where $L_n(n) = v$ is supposed to be an experiment to differentiate x_1 and x_2 . Without loss of generality, we assume that t_1 and t_2 are in the left and right subtrees of n respectively. Therefore, we have $\mathbf{TE}(x_1, v) = \mathbf{F}$ and $\mathbf{TE}(x_2, v) = \mathbf{T}$. It follows that $u(x_1v)^\omega \notin UP(L)$ and $u(x_2v)^\omega \in UP(L)$, which implies that $x_1 \not\approx_P^u x_2$. Contradiction.

■

Lem. 9 cannot apply to the progress trees in syntactic and recurrent FDFAs as the progress trees heavily rely on the current leading automaton. In the following, we prove the correctness of syntactic and recurrent FDFA. We say the leading automaton M is consistent with \sim_L iff for any $x_1, x_2 \in \Sigma^*$, we have $M(x_1) = M(x_2) \iff x_1 \sim_L x_2$.

Lemma 10. *For the progress trees in the syntactic and recurrent FDFA, the tree-based algorithm will never classify two finite words which are in the same equivalence class into two different terminal nodes if the leading automaton M is consistent with \sim_L .*

If the tree-based algorithm classifies two finite words which are in the same equivalence class into two different terminal nodes, then M is not consistent with \sim_L currently.

Proof. Intuitively, the progress trees \mathcal{T}_u in syntactic and recurrent FDFAs are constructed with respect to the current leading automaton. We prove the lemma in following cases.

- \mathcal{T}_u is a progress tree in syntactic FDFA. We assume that $x_1 \approx_S^u x_2$. Suppose x_1 and x_2 have been assigned to terminal node t_1 and t_2 of \mathcal{T}_u respectively. Therefore, we can find the least common ancestor n from \mathcal{T}_u , where $L_n(n) = v$ is supposed to be an experiment to differentiate x_1 and x_2 . Thus, by the definition of **TE** in syntactic FDFA, we can assume that $d_1 := \mathbf{TE}(x_1, v) = (M(ux_1), m_1)$ and $d_2 := \mathbf{TE}(x_2, v) = (M(ux_2), m_2)$ where $m_1, m_2 \in \{A, B, C\}$. Since t_1 and t_2 are in different subtrees of n , we thus have $d_1 \neq d_2$, that is, $M(ux_1) \neq M(ux_2)$ or $m_1 \neq m_2$.

1) First we assume that M is consistent with \sim_L .

- $M(ux_1) \neq M(ux_2)$. Since $x_1 \approx_S^u x_2$, we have $ux_1 \sim_L ux_2$, which implies that $M(ux_1) = M(ux_2)$. Contradiction.
- $m_1 \neq m_2$. Since $x_1 \approx_S^u x_2$, we have $ux_1 \sim_L ux_2$, which follows that $M(ux_1) = M(ux_2)$ since M is consistent with \sim_L . Moreover, we have that $M(ux_1v) = M(ux_2v)$ since M is deterministic. We discuss the values of m_1 and m_2 in the following.
 - * $u = M(ux_1v)$. It follows that $ux_1v \sim_L u$ since M is consistent with \sim_L , which implies that $u(x_1v)^\omega \in UP(L) \iff u(x_2v)^\omega \in UP(L)$. Moreover, we have $u = M(ux_2v)$ since $ux_1 \sim_L ux_2$. Therefore, we conclude that $m_1, m_2 \in \{A, B\}$ by the definition of **TE**. Without loss of generality, we let $m_1 = A$ and $m_2 = B$, which implies that $u(x_1v)^\omega \in UP(L)$ while $u(x_2v)^\omega \notin UP(L)$. Contradiction.
 - * $u \neq M(ux_1v)$. Thus, we have $m_1 = m_2 = C$, which follows that $d_1 = d_2$ since $M(ux_1) = M(ux_2)$. Contradiction.

Therefore, t_1 and t_2 cannot be different terminal nodes.

2) In this case, M is not necessarily consistent with \sim_L .

- $M(ux_1) \neq M(ux_2)$. Let $s_1 = M(ux_1)$ and $s_2 = M(ux_2)$. We have that s_1 and s_2 are classified into different terminal nodes in the leading tree \mathcal{T} since $s_1 \neq s_2$ and they are two labels of the terminal nodes. It follows that $s_1 \not\sim_L s_2$ by Lem. 9. By $x_1 \approx_S^u x_2$, we have $ux_1 \sim_L ux_2$, which implies that $s_1 \not\sim_L ux_1$ or $s_2 \not\sim_L ux_2$, otherwise we get $s_1 \sim_L s_2$. Without loss of generality, suppose $s_1 \not\sim_L ux_1$, then there exists some experiment (y, v) to differentiate them. However, ux_1 is currently assigned into the equivalence class of s_1 since $s_1 = M(ux_1)$. It follows that M is not consistent with \sim_L .

- $m_1 \neq m_2$.
 - 1) We assume that $ux_1v \sim_L u$, then we have $ux_2v \sim_L u$ since $ux_1 \sim_L ux_2$ by $x_1 \approx_S^u x_2$, which implies that $u(x_1v)^\omega \in UP(L) \iff u(x_2v)^\omega \in UP(L)$. If M is consistent with \sim_L , we conclude that $m_1 = m_2 = A$ or $m_1 = m_2 = B$. Contradiction. Therefore, M is not consistent with \sim_L .
 - 2) We assume that $ux_1v \not\sim_L u$, then we can find some experiment (y, z) to differentiate them. It follows that $ux_2v \not\sim_L u$ since $x_1 \approx_S^u x_2$ and $ux_1 \sim_L ux_2$. Assume that M is consistent with \sim_L , then we have that $u \neq M(ux_1v)$ and $u \neq M(ux_2v)$, which implies that $m_1 = m_2 = C$. Contradiction. Thus, M is not consistent with \sim_L .
- \mathcal{T}_u is a progress tree in recurrent FDFA. The analysis is similar as the syntactic FDFA. We assume that $x_1 \approx_R^u x_2$. Suppose x_1 and x_2 have been assigned to terminal node t_1 and t_2 of \mathcal{T}_u respectively. Therefore, we can find the least common ancestor n from \mathcal{T}_u , where $L_n(n) = v$ is supposed to be an experiment to differentiate x_1 and x_2 . Thus, we can assume that $d_1 := \mathbf{TE}(x_1, v)$ and $d_2 := \mathbf{TE}(x_2, v)$ where $d_1, d_2 \in \{F, T\}$. Since t_1 and t_2 are in different subtrees of n , we thus have $d_1 \neq d_2$.
 - 1) We assume that M is consistent with \sim_L . Without loss of generality, suppose $d_1 = F$ and $d_2 = T$. Since $d_2 = T$, we have that $u = M(ux_2v)$ and $u(x_2v)^\omega \in UP(L)$. It follows that $ux_2v \sim_L u$ since M is consistent with \sim_L . Moreover, we conclude that $u = M(ux_1v)$ and $u(x_1v)^\omega \in UP(L)$ by the fact that $x_1 \approx_R^u x_2$. By the definition of \mathbf{TE} , we have $d_1 = T$. Contradiction. Therefore, t_1 and t_2 cannot be different terminal nodes.
 - 2) M is not necessarily consistent with \sim_L . Without loss of generality, suppose $d_1 = F$ and $d_2 = T$. Since $d_2 = T$, we have that $u = M(ux_2v)$ and $u(x_2v)^\omega \in UP(L)$. Assume that M is consistent with \sim_L , it follows that $ux_2v \sim_L u$. Moreover, we conclude that $u = M(ux_1v)$ and $u(x_1v)^\omega \in UP(L)$ by the fact that $x_1 \approx_R^u x_2$. By the definition of \mathbf{TE} , we have $d_1 = T$. Contradiction. Therefore, M is not consistent with \sim_L .

■

Once two finite words which are in the same equivalence class have been classified into two terminal nodes in the progress tree, we can always prove that the leading automaton is not consistent with \sim_L . Therefore, the FDFA teacher is able to return some counterexample to refine the leading automaton. If the leading automaton changes, the FDFA learner should learn all progress automata from scratch with respect to current leading automaton. At a certain point, the leading automaton M will be consistent with \sim_L since it will be added a new state after every refinement. Thus, we conclude that the equivalence classes in the progress trees will finally be classified correctly.

Proposition 7. *Given the FDFA teacher that is able to answer membership and equivalence queries for FDFA, the tree-based FDFA learning algorithm can correctly classify all finite words.*

E.2 Complexity for Tree-based FDFA Learning Algorithm

The counterexample guided refinement for \mathcal{F} shows that:

Corollary 2. *Given a counterexample (u, v) for FDFA learner, the tree-based FDFA learner will either add a new state to the leading automaton M or the corresponding progress automaton $A^{\bar{u}}$.*

Corollary. 2 is a critical property for the termination of the tree-based FDFA learning algorithm since each time we either make progress for the leading automaton or the corresponding progress automaton.

In Lem. 10, we encounter a situation where the progress tree may classify two finite words which are in the same equivalence class into two terminal nodes if M is not consistent with \sim_L . One may worry that if the FDFA teacher chooses to refine the progress automaton continually, the refinement may not terminate. Lem. 11 shows that it will terminate since the number of equivalence classes of the progress automata with respect to M is finite. More precisely, if we fix the leading automaton M , we are actually learning a DFA induced by the right congruence $x \approx_S^u, y$ iff $M(ux) = M(uy)$ and for every $v \in \Sigma^*$, if $M(uxv) = u$, then $u(xv)^\omega \in L \iff u(yv)^\omega \in L$. One can easily verify that $x \approx_S^u, y$ is a right congruence. We remark that if M is consistent with \sim_L , then $x \approx_S^u, y$ is equivalent to $x \approx_S^u y$.

Lemma 11. *Given then leading automaton M , then for every state u in M , the index of \approx_S^u is bounded by $|Q| \cdot |\approx_P^u|$ where Q is the state set of M .*

Proof. We prove the lemma by giving the upper bound $|Q| \cdot |\approx_P^u|$ of the index of \approx_S^u . We use q_i to denote the state which can be reached by u for $1 \leq i \leq n$ where n is the number of states reachable by u . We classify any $x \in \Sigma^*$ into a equivalence class of \approx_S^u , as follows.

We first find $q_i = M(ux)$. Since for every $y \in \Sigma^*$ with $q_i = M(uy)$, we have $M(uxv) = M(uyv)$, thus those experiments $v \in \Sigma^*$ with $M(uxv) \neq u$ are not able to differentiate x and y . In other words, the value of $M(uxv) = u$ is not necessary here. Therefore, if we only consider $x, y \in \Sigma^*$ with $q_i = M(ux) = M(uy)$, the criterion to decide whether x and y are in the same equivalence class is to judge whether for any $v \in \Sigma^*$, $u(xv)^\omega \in L \iff u(yv)^\omega \in L$, which is exactly the same definition for \approx_P^u . Thus, we can find the notation $(q_i, [x]_{\approx_P^u})$ to uniquely represent the equivalence class $[x]_{\approx_S^u}$. Therefore, the index of the right congruence \approx_S^u is $n \cdot |\approx_P^u| \leq |Q| \cdot |\approx_P^u|$. ■

Similarly, if we fix the leading automaton M and learn recurrent FDFA, we are actually learning DFA induced by the right congruence $x \approx_{R'}^u, y$ iff for every $v \in \Sigma^*$, $M(uxv) = u \wedge u(xv)^\omega \in L \iff M(uyv) = u \wedge u(yv)^\omega \in L$. Since $x \approx_S^u, y$ implies $x \approx_{R'}^u, y$, it follows that $|\approx_{R'}^u|$ is smaller than $|\approx_S^u|$.

The implication from $x \approx_S^u, y$ to $x \approx_{R'}^u, y$ can be easily established by assuming $x \approx_S^u, y$ and then for any $v \in \Sigma^*$, we have that $uyv \sim_M u \wedge u(yv)^\omega \in L$ if $uxv \sim_M u \wedge u(xv)^\omega \in L$. First, assuming that $uxv \sim_M u \wedge u(xv)^\omega \in L$ and $x \approx_S^u, y$, one can easily conclude that $u(yv)^\omega \in L$. In addition, one can combine the result $ux \sim_M uy$ from $x \approx_S^u, y$ and assumption $uxv \sim_M u$ together to prove $uyv \sim_M u$ since M is deterministic and \sim_M is an equivalence relation.

Lemma 12. *Given the leading automaton M , then for every state u in M , the index of $\approx_{R'}^u$ is bounded by $|Q| \cdot |\approx_P^u|$ where Q is the state set of M .*

Assume that $\mathcal{F} = (M, \{A^u\})$ is the corresponding periodic FDFFA recognizing L . Let n be the number of states in M of \mathcal{F} and k be the number of states in the largest progress automaton of \mathcal{F} .

Theorem 2 (Query Complexity). *Let (u, v) be the longest counterexample returned from the FDFFA teacher. The number of equivalence queries needed for the tree-based FDFFA learning algorithm to learn the periodic FDFFA of L is in $O(n + nk)$, while the number of membership queries is in $O((n + nk) \cdot (|u| + |v| + (n + k) \cdot |\Sigma|))$.*

For the syntactic and recurrent FDFFAs, the number of equivalence queries needed for the tree-based FDFFA learning algorithm is in $O(n + n^3k)$, while the number of membership queries is in $O((n + n^3k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$.

Proof. Thm. 2 can be concluded from Lem. 9, Coro. 2, Lem. 11, and Lem. 12. Suppose $\mathcal{F} = (M, \{A^u\})$ is the corresponding periodic FDFFA recognizing L . The number of states in M is n and k is the number of the largest progress automaton in \mathcal{F} .

Given a counterexample (u, v) , the number of membership queries is at most $|u|$ if we refine the leading automaton and is at most $|v|$ if we refine the progress automaton. Therefore, the number of membership queries used in analyzing counterexample is bounded by $|u| + |v|$. One can also use binary search to reduce the number of membership queries used by counterexample analysis to $\log(|u| + |v|)$. Moreover, when the classification tree has been refined, we need to construct the corresponding M or $A^{M(u)}$ again. Suppose the new added terminal node is labeled by p , the terminal node which needs to be refined is labeled by q and the experiment is e . We only need to compute the successors of p and update the successors of the predecessors of q .

- Computing the successors of p is to calculate $\delta(p, a)$ for every $a \in \Sigma$, which requires $|\Sigma| \cdot h$ membership queries where h is the height of the classification tree.
- Updating the successors of the predecessors of q is to calculate $\mathbf{TE}(s, e)$ for every state label s and $a \in \Sigma$ such that currently we have $\delta(s, a) = q$, which requires at most $|\Sigma| \cdot m$ membership queries where m is the number of states in current M or $A^{M(u)}$.

Since the height of the classification tree is at most m , thus the number of membership queries needed for constructing the conjectured DFA is at most $2 \cdot m \cdot |\Sigma|$. It follows that for the tree-based algorithm, the number of membership queries used in the counterexample guided refinement is bounded by $|u| + |v| + 2m \cdot |\Sigma|$. We remark that in the table-based algorithm, the number of membership queries used in the counterexample guided refinement is bounded by $|u| + |v| + m + |\Sigma| \cdot m + |\Sigma|$.

We give the complexity of the tree-based algorithm as follows.

- For periodic FDFFA. During the learning procedure, when receiving a counterexample for FDFFA learner, the tree-based algorithm either adds a new state into the leading automaton or into the corresponding progress automaton. Thus, the number of the equivalence queries is bounded by $n + nk$ since the number of states in the target periodic FDFFA is bounded by $n + nk$. In periodic FDFFA, we have $m \leq n + k$ since every time we either refine the leading automaton or a progress automaton. Therefore, the number of membership queries needed for the algorithm is bounded by $(n + nk) \cdot (|u| + |v| + 2(n + k) \cdot |\Sigma|) \in O((n + nk) \cdot (|u| + |v| + (n + k) \cdot |\Sigma|))$ in the worst case.

- For syntactic and recurrent FDFAs, when receiving a counterexample for FDFAs learner, the tree-based algorithm will first decide whether to refine the leading automaton or the progress automaton. If it decides to refine the leading automaton, we need to initialize all progress trees with a single node labelled by ϵ again, so the number of states in the progress automata of the FDFAs may decrease at that point, otherwise it refines the progress automaton and the number of states in FDFAs will increase by one.

In the worst case, the learner will try to learn the progress automata as much as possible. In other words, if current leading automaton has m states, the number of states in every progress automaton is at most $m \cdot k$ according to Lem. 11 and Lem. 12. When all progress trees cannot be refined any more, either the learning task finishes or the FDFAs teacher returns a counterexample to refine current leading automaton. For the latter case, the number of states in the leading automaton will increase by one, that is, $m+1$, and we need to redo the learning work for all progress trees. The number of states in all progress automata in the new FDFAs is bounded by $(m+1)^2 \cdot k$. Therefore, the number of equivalence queries needed for tree-based algorithm is bounded by $(1 + 1 \cdot 1 \cdot k) + (1 + 2 \cdot 2 \cdot k) + \dots + (1 + (n-1) \cdot (n-1) \cdot k) + (1 + n \cdot n \cdot k) \in O(n + n^3k)$. Similarly, in syntactic and recurrent FDFAs, we have that $m \leq n + nk$ since the number of states in a progress automaton is bounded by nk . It follows that the number of membership queries needed for the algorithm is in $O((n + n^3k) \cdot (|u| + |v| + 2(n + nk) \cdot |\Sigma|)) \in O((n + n^3k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$ in the worst case.

■

Theorem 3 (Space Complexity). *For all tree-based algorithms, the space required to learn the leading automaton is in $O(n)$. For learning periodic FDFAs, the space required for each progress automaton is in $O(k)$, while for syntactic and recurrent FDFAs, the space required is in $O(nk)$. For all table-based algorithms, the space required to learn the leading automaton is in $O((n + n \cdot |\Sigma|) \cdot n)$. For learning periodic FDFAs, the space required for each progress automaton is in $O((k + k \cdot |\Sigma|) \cdot k)$, while for syntactic and recurrent FDFAs, the space required is in $O((nk + nk \cdot |\Sigma|) \cdot nk)$.*

Proof. As we mentioned in Sec. 4, the FDFAs learner can be viewed as a learner consisting of many component DFA learners. For a component DFA learner, suppose the number of the states in the target DFA is m , for table-based component DFA learner, the size of the observation table is in $O((m + m \cdot |\Sigma|) \cdot m)$ since there are $m + m \cdot |\Sigma|$ rows and at most m columns in the observation table in the worst case. In contrast, for the tree-based component DFA learner, the number of nodes in the classification tree is in $O(m)$ since the number of terminal nodes in the classification tree is m and the number of internal nodes is at most $m - 1$.

- For the periodic FDFAs, the number of states in the FDFAs will increase after each refinement step. Thus, it is easy to conclude that the space required for the leading automaton is in $O(n)$ if we use tree-based learning algorithm and the space required by the table-based algorithm is in $O((n + n \cdot |\Sigma|) \cdot n)$. Similarly, the space required by tree-based learning algorithm to learn each progress automaton is in $O(k)$, while for table-based algorithm, the space required is in $O((k + k \cdot |\Sigma|) \cdot k)$.

- For the syntactic and recurrent FDFA. The learning procedure for the leading automaton is the same as periodic automaton. Thus the space required by table-based and tree-based algorithm remain the same.

For learning progress automaton, the number of states in each progress automaton is at most nk according to Lem. 11 and Lem. 12. Therefore, for table-based algorithm, the space required is in $O((nk + nk \cdot |\Sigma|) \cdot nk)$. While for tree-based algorithm, the space required to learn each progress automaton is in $O(nk)$.

■

Proposition 8. *In FDFA teacher, suppose n is the number of states in the leading automaton and k is the number of states in the largest progress automaton in the input FDFA \mathcal{F} and the returned counterexample uv^ω has a decomposition (u, v) . Then*

- *the time and space complexity for building the BAs \underline{B} and \overline{B} are in $O(n^2k^3)$ and $O(n^2k^2)$ respectively, and*
- *for the under approximation method, the time and space complexity for analyzing the counterexample uv^ω are in $O(n^2k \cdot (|v|(|v|+|u|)))$, while for the over approximation method, the time and space complexity for analyzing the counterexample uv^ω are in $O(n^2k^2 \cdot (|v|(|v|+|u|)))$ and in $O(n^2k(|v|(|v|+|u|)))$ respectively.*

Proof. Suppose the FDFA teacher currently needs to answer the equivalence query for FDFA $\mathcal{F} = (M, \{A^u\})$. Then the number of states in \underline{B} (\overline{B}) is in $O(n + n^2k^3)$ (respectively, $O(n + n^2k^2)$). In addition, the number of states in FA \mathcal{D}_1 and \mathcal{D}_2 are both in $O(n + n^2k)$ and the number of states in $\mathcal{D}_{u\&v}$ is at most $|v|(|v| + |u|)$ given that (u, v) is a decomposition of the returned counterexample uv^ω , which can be applied to the under and the over approximation except for case O3 in the over approximation. When we analyze the spurious negative counterexample, the time and space complexity are in $O(nk(n + nk) \cdot (|v|(|v| + |u|)))$ and $O((n + nk) \cdot (|v|(|v| + |u|)))$ according to Lem. 5. Therefore, we complete the proof. ■

Theorem 4 (Correctness and Termination). *The BA learning algorithm based on the under-approximation method always terminates and returns a BA recognizing the unknown ω -regular language L in polynomial time. If the BA learning algorithm based on the over-approximation method terminates without reporting an error, it returns a BA recognizing L .*

Proof. If we use the under-approximation method to construct the Büchi automaton, then the BA learning algorithm will need to first learn a canonical FDFA to get the final Büchi automaton in the worst case. This theorem is justified by Lem. 2 and Lem. 3. ■