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

Yong Li[1,2], Yu-Fang Chen[3], Lijun Zhang[1,2], Depeng Liu[1,2]

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
[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, 36] 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, 33] and *synthesizing abstract system model for further analysis* [1,5,16,18,22,24,26,35,37,40].

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, 37, 41], 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, 35, 40].

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 [33]. 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 [39] 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 [34]. 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 consisting of 295 Büchi automata corresponding to all 295 LTL specifications available in BüchiStore [38].

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 $\{i, i+1, \cdots, j\}$. 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 $\mathrm{UP}(L) = \{uv^\omega \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$, 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 $v = a_1 a_2 a_3 \cdots a_n$ is a sequence of states $q_0, q_1, \cdots, q_n$ 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 (FDFA) [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 FDFA $\mathcal{F}$ characterizes a set of ultimately periodic words $\mathrm{UP}(\mathcal{F})$. Formally, an ultimately periodic word w is in $\mathrm{UP}(\mathcal{F})$ 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 FDFA $\mathcal{F}$ 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 $ab \notin L(A^{M(bab)}) = L(A^\epsilon)$.

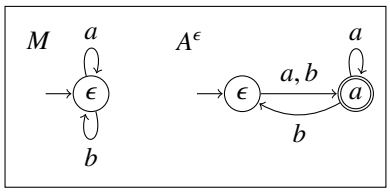


Fig. 1. An example of an FDFA

For any ω-regular language L, there exists an FDFA $\mathcal{F}$ such that $\mathrm{UP}(L) = \mathrm{UP}(\mathcal{F})$ [10]. We show in Sec. 6 that it is not the case for the reverse direction. More precisely, in [10], three kinds of FDFAs are suggested as the canonical representations of ω-regular languages, namely *periodic* FDFA, *syntactic* FDFA and *recurrent* FDFA. Their formal definitions are given in terms of *right congruence*.

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

the notation $[v]_\backsim$ to represent the class of $\backsim$ in which v resides and ignore the subscript $\backsim$ when the context is clear. The right congruence $\backsim_L$ of a given ω-regular language L is defined such that $x \backsim_L y$ iff $\forall w \in \Sigma^\omega . xw \in L \iff yw \in L$. The index of $\backsim_L$ is finite because it is not larger than the number of states in a deterministic Muller automaton recognizing L [34].

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

We define the right congruences $\approx^u_P, \approx^u_S$, and $\approx^u_R$ for progress automata A^u of periodic, syntactic, and recurrent FDFA *respectively as follows:*

$$
\begin{array}{ll}
x \approx^u_P y \text{ iff} & \forall v \in \Sigma^*, u(xv)^\omega \in L \iff u(yv)^\omega \in L, \\
x \approx^u_S y \text{ iff } ux \backsim_L uy \text{ and} & \forall v \in \Sigma^*, uxv \backsim_L u \implies (u(xv)^\omega \in L \iff u(yv)^\omega \in L), \text{ and} \\
x \approx^u_R y \text{ iff} & \forall v \in \Sigma^*, uxv \backsim_L u \wedge u(xv)^\omega \in L \iff uyv \backsim_L u \wedge u(yv)^\omega \in L.
\end{array}
$$

The progress automaton A^u is the tuple $(\Sigma, \Sigma^*/_{\approx^u_K}, [\epsilon]_{\approx^u_K}, \delta_K, F_K)$, where $\delta_K([u]_{\approx^u_K}, a) = [ua]_{\approx^u_K}$ for all $u \in \Sigma^*$ and $a \in \Sigma$. The accepting states F_K is the set of equivalence classes $[v]_{\approx^u_K}$ for which $uv \backsim_L u$ and $uv^\omega \in L$ when $K \in \{S, R\}$ and the set of equivalence classes $[v]_{\approx^u_K}$ 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)* FDFA *of an ω-regular language L. Then $UP(\mathcal{F}) = UP(L)$.*

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

3 Büchi Automata Learning Framework based on FDFA

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 $\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$ asks if $uv^\omega \in L$. For an equivalence query $\mathsf{Equ}^{\mathsf{BA}}(B)$, 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 FDFA *learner* and the FDFA *teacher*. Note that one can place any FDFA learning algorithm to the FDFA learner component. For instance, one can use the FDFA learner from [10] which employs a table to store query results, or the FDFA learner using a classification tree proposed in this paper. The FDFA teacher can be any teacher who can answer membership and equivalence queries about an unknown FDFA.

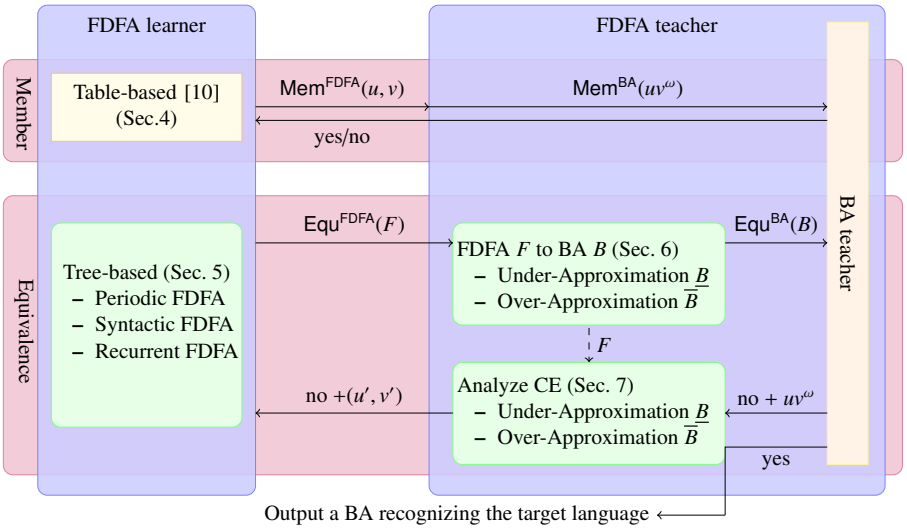


Fig. 2. Overview of the learning framework based on FDFA learning. The components in rectangle boxes are results from existing works. The components in rounded boxes are our new contributions.

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 $\mathsf{Mem}^{\mathsf{FDFA}}(u, v)$ and $\mathsf{Equ}^{\mathsf{FDFA}}(F)$ posed by the FDFA learner. Answering $\mathsf{Mem}^{\mathsf{FDFA}}(u, v)$ is easy. The FDFA teacher just needs to redirect the result of $\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$ to the FDFA learner. Answering equivalence query $\mathsf{Equ}^{\mathsf{FDFA}}(F)$ 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 $\mathsf{Equ}^{\mathsf{BA}}(B)$. In Sec. 6, we show that, in general, it is impossible to build a BA B from an FDFA F such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$. Therefore in Sec. 6, we propose two methods to approximate $\mathrm{UP}(F)$, 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 $\mathrm{UP}(L(B)) \subseteq \mathrm{UP}(F)$ (respectively, $\mathrm{UP}(F) \subseteq \mathrm{UP}(L(B))$). 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 complete

method for BA learning (Lem. 1 and 2). As for the over-approximation method, we cannot guarantee to get a BA B such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$ 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 $\mathsf{Equ}^{\mathsf{BA}}(B)$, the FDFA teacher will terminate the learning procedure and outputs a BA recognizing L.

Due to the lack of space, all missing proofs and details for our Büchi learning algorithm are provided in [32].

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, $L_n : N \to \Sigma^* \cup (\Sigma^* \times \Sigma^*)$ labels an internal node with an *experiment* and a terminal node with a *state*, and $L_e : N \times D \to N$ 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 $\mathbf{TE} : \Sigma^* \times (\Sigma^* \cup (\Sigma^* \times \Sigma^*)) \to D$. 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 $D = \{\mathrm{F}, \mathrm{T}\}$. The tree experiment function $\mathbf{TE}(u, (x, y)) = \mathrm{T}$ iff $uxy^\omega \in L$ (recall the definition of $\backsim_L$ 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 $D = \{\mathrm{F}, \mathrm{T}\}$. The experiment function $\mathbf{TE}(x, e) = \mathrm{T}$ 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 $D = Q \times \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ 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 $t = \mathrm{A}$ iff $u = M(uxe) \wedge u(xe)^\omega \in L$, $t = \mathrm{B}$ iff $u = M(uxe) \wedge u(xe)^\omega \notin L$, and $t = \mathrm{C}$ iff $u \neq M(uxe)$.

For example, assuming that M is constructed from the right congruence $\backsim_L$, 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 $ux \backsim_L uy$ because $M(ux) = z = M(uy)$. Moreover, the experiment e cannot distinguish x and y because $uxe \backsim_L u \backsim_L uye$ and both $u(xe)^\omega, u(ye)^\omega \in L$.

Recurrent FDFA: The progress tree for recurrent FDFA is a binary tree labeled with $D = \{\mathrm{F}, \mathrm{T}\}$. The function $\mathbf{TE}(x, e) = \mathrm{T}$ 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 $\mathcal{T}_\epsilon$ 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 procedure[4]. 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, $F = \{v \mid uv \backsim_M u, uv^\omega \in L, v \in Q\}$.

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* $uv \backsim_M u$, $uv^\omega \in UP(L)$, *and* (u, v) *is not accepted by* $\mathcal{F}$,
- (u, v) *is* negative *if* $uv \backsim_M u$, $uv^\omega \notin UP(L)$, *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 $uv \backsim_M u$, 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 $uv \sim_M u$, $uv^\omega \notin \mathrm{UP}(L)$ and (u, v) is accepted by $\mathcal{F}$. Let $\tilde{u} = M(u)$, if $\tilde{u}v^\omega \in \mathrm{UP}(L)$, the refinement of the leading tree is performed, otherwise $\tilde{u}v^\omega \notin \mathrm{UP}(L)$, 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 $\mathbf{TE}(q, e) = \mathrm{F}$. 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 $s_0 s_1 \cdots s_n$ 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 $\tilde{u}v^\omega \in \mathrm{UP}(L)$ (the condition to refine the leading tree), we have $\mathbf{TE}(s_0, (u[1 \cdots n], v)) = \mathrm{F} \neq \mathrm{T} = \mathbf{TE}(s_n, (\epsilon, v)) = \mathbf{TE}(s_n, (u[n+1 \cdots n], v))$ because $uv^\omega \notin \mathrm{UP}(L)$ and $\tilde{u}v^\omega \in \mathrm{UP}(L)$. 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 $\mathcal{T}_\epsilon$ are depicted on the left of Fig. 3. The dashed 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.

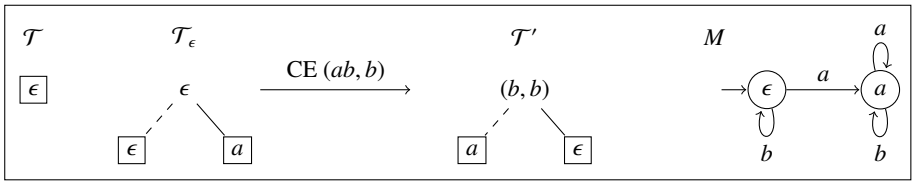


Fig. 3. Refinement of the leading tree and the corresponding leading automaton

Refinement for the progress tree: Here we explain the case of periodic FDFAs. The other cases are similar and we leave the details in [32]. Recall that $\tilde{u}v^\omega \notin \mathrm{UP}(L)$ and thus the algorithm refines the progress tree $\mathcal{T}_{\tilde{u}}$. Let $|v| = n$ and $h = s_0 s_1 \cdots s_n$ 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 $\tilde{u}(\tilde{v})^\omega \in \mathrm{UP}(L)$ because $\tilde{v}$ is an accepting state. From the facts that $\tilde{u}v^\omega \notin \mathrm{UP}(L)$ and $\tilde{u}(\tilde{v})^\omega \in \mathrm{UP}(L)$, we have $\mathbf{TE}(s_0, v[1 \cdots n]) = \mathrm{F} \neq \mathrm{T} = \mathbf{TE}(s_n, \epsilon) = \mathbf{TE}(s_n, v[n+1 \cdots n])$. 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 $\mathcal{T}_\epsilon$ 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 $\mathrm{UP}(L(B)) = \mathrm{UP}(\mathcal{F})$.

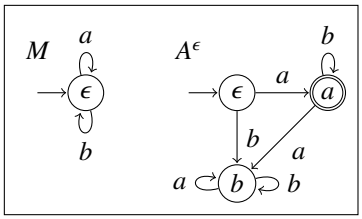


Fig. 4. An FDFA $\mathcal{F}$ such that $\mathrm{UP}(\mathcal{F})$ does not characterize an ω-regular language

Example 2. Consider a non-canonical FDFA $\mathcal{F}$ in Fig. 4, we have $\mathrm{UP}(\mathcal{F}) = \bigcup_{n=0}^{\infty}\{a, b\}^* \cdot (ab^n)^\omega$. We assume that $\mathrm{UP}(\mathcal{F})$ 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 $\approx_P^\epsilon$ 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 $\epsilon \cdot (ab^k \cdot ab^k)^\omega \in \mathrm{UP}(\mathcal{F})$ and $\epsilon \cdot (ab^j \cdot ab^k)^\omega \notin \mathrm{UP}(\mathcal{F})$. It follows that $\approx_P^\epsilon$ is of infinite index. We conclude that $\mathrm{UP}(\mathcal{F})$ 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 $N_{(u,v)} = \{v^\omega \mid uv \backsim_M u \wedge v \in L((A^u)_v^{s_u})\}$. Then $\mathrm{UP}(\mathcal{F}) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot N_{(u,v)}$.

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. $\mathrm{UP}(L(\overline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \overline{N}_{(u,v)}$ and $\mathrm{UP}(L(\underline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot \underline{N}_{(u,v)}$.

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 $(\Sigma, Q_{u,v}, s_{u,v}, \{\underline{f}_{u,v}\}, \delta_{u,v})$ 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 $O(n^2k^3)$ 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* 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 $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \backsim_M u$.*

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 $b^\omega \in \mathrm{UP}(\mathcal{F})$ while $b^\omega \notin \mathrm{UP}(L(\underline{B}))$.

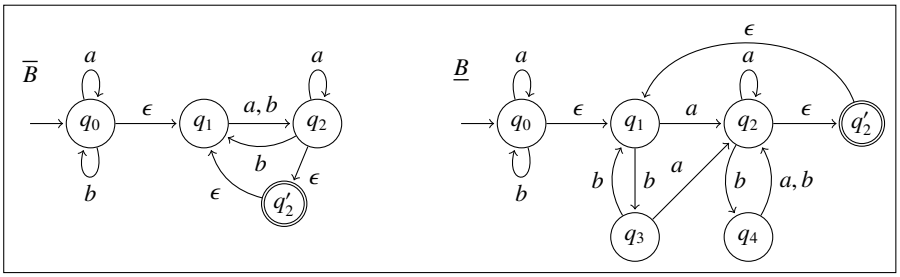


Fig. 5. NBA $\overline{B}$ and $\underline{B}$ for $\mathcal{F}$ in Fig. 1

7 Counterexample Analysis for FDFA 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 FDFA teacher.

Definition 5 (Counterexample for the FDFA 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 FDFA learner in Def. 3. Below we illustrate the necessity of the counterexample analysis by an example.

Example 3. Again, consider the conjectured FDFA $\mathcal{F}$ depicted in Fig. 1 for $L = a^\omega + b^\omega$. Suppose the BA teacher returns a negative counterexample $(ba)^\omega$. In order to remove $(ba)^\omega \in \mathrm{UP}(\mathcal{F})$, 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.

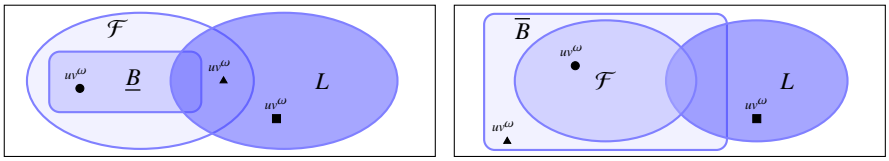
A positive (respectively negative) counterexample uv^ω for the FDFA teacher is *spurious* if $uv^\omega \in \mathrm{UP}(\mathcal{F})$ (respectively $uv^\omega \notin \mathrm{UP}(\mathcal{F})$). Suppose we use the under-approximation method to construct the BA $\underline{B}$ from $\mathcal{F}$ depicted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

Given FDFA $\mathcal{F} = (M, \{A^u\})$, 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 $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \in L(A^{M(u)})\}$, and
- an FA $\mathcal{D}_2$ with $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, v \notin L(A^{M(u)})\}$.

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 $u'v' \backsim_M u'$.

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



(a) Under-Approximation (b) Over-Approximation

Fig. 6. The Case for Counterexample Analysis

U1 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (square). The word uv^ω is a positive counterexample, one has to find a decomposition (u', v') such that $u'v' \backsim_M u'$ and $u'v'^\omega = uv^\omega$. This can be easily done by taking a word $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$.

U2 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (circle). 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 $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$.

U3 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (triangle). 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 $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\overline{B}))$, and all possible cases for a counterexample $uv^\omega \in \mathrm{UP}(L(\overline{B})) \oplus \mathrm{UP}(L)$ is depicted in Fig. 6(b).

O1 : $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (square). The word uv^ω is a positive counterexample that can be dealt with the same method for case U1.

O2 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$ (circle). The word uv^ω is a negative counterexample that can be dealt with the same method for case U2.

O3 : $uv^\omega \notin \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$ (triangle). 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 $v' = v_1 v_2 \cdots v_n$, $u'v_i \backsim_M u'$, 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 $u'v_i^\omega \notin \mathrm{UP}(L)$, then we return (u', v_i), otherwise, the algorithm aborts with an error.

Finally, we note that determining whether $uv^\omega \in \mathrm{UP}(L)$ can be done by posing a membership query $\mathsf{Mem}^{\mathsf{BA}}(uv^\omega)$, and checking whether $uv^\omega \in \mathrm{UP}(\mathcal{F})$ boils down to checking the emptiness of $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$. The construction for $\mathcal{D}_{u\$v}$, $\mathcal{D}_1$, and $\mathcal{D}_2$, and the correctness proof of counterexample analysis are given in [32].

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 FDFA*s, 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* FDFA*s, 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* FDFA*s, 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 $\mathrm{UP}(\mathcal{F}) = \mathrm{UP}(L(\underline{B}))$, 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 [38], 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.

The overall experimental results are given in Tab. 1. In this section, we use $L^{\$}$ to denote the ω-regular learning algorithm in [20], and L^{Periodic}, $L^{\mathsf{Syntactic}}$, and $L^{\mathsf{Recurrent}}$

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	$L^\$$		L^{Periodic}				$L^{\text{Syntactic}}$				$L^{\text{Recurrent}}$			
Struct.& Approxi.	Table	Tree	Table		Tree		Table		Tree		Table		Tree	
			under	over	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
Time_{eq}(s)	146	817	580	92	186	159	111	115	**89**	91	149	149	462	465
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

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 $L^\$$ 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.

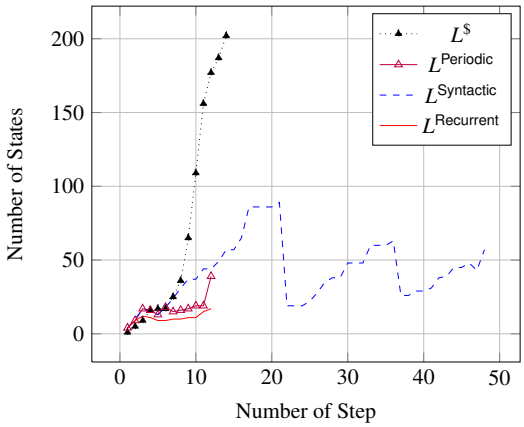


Fig. 7. Growth of state counts in BA

In the experiment, we observe that table-based $L^\$$ has 4 cases cannot be finished within the timeout period, which is the largest number amount all learning algorithms[5]. We found that for these 4 cases, the average time required for $L^\$$ 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 $L^\$$ 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 learning algorithm using one learning task that can-

[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.

not be solved by $L^{\$}$ within the timeout period. The figures of the other three learning tasks show the same trend and hence are omitted. Another interesting observation is that the sizes of BAs generated by $L^{\mathsf{Syntactic}}$ 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 $L^{\mathsf{Recurrent}}$. For L^{Periodic}, the over-approximation method solved more learning tasks than the under-approximation method. For $L^{\mathsf{Recurrent}}$, 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.

Acknowledgement This work has been supported by by the National Basic Research (973) Program of China under Grant No. 2014CB340701, the CAS Fellowship for Visiting Scientists from Taiwan under Grant No. 2015TW2GA0001, the National Natural Science Foundation of China (Grants 61532019, 61472473), the CAS/SAFEA International Partnership Program for Creative Research Teams, the Sino-German CDZ project CAP (GZ 1023), and the MOST project No. 103-2221-E-001-019-MY3.

References

1. F. Aarts, B. Jonsson, J. Uijen, and F. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods in System Design*, 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. Černỳ, 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. In *1960 International Congress on Logic, Methodology and Philosophy of Science*, volume 44, pages 1–11, 1966.
14. H. Calbrix, M. Nivat, and A. Podelski. 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. Y. Li, Y. Chen, L. Zhang, and D. Liu. A Novel Learning Algorithm for Büchi Automata based on Family of DFAs and Classification Trees. *CoRR*, abs/1610.07380, 2016.
33. 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.
34. O. Maler and L. Staiger. On Syntactic Congruences for Omega-Languages. In *STACS*, pages 586–594, 1993.
35. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.
36. R. L. Rivest and R. E. Schapire. Inference of Finite Automata Using Homing Sequences. In *STOC*, pages 411–420, 1989.
37. 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.
38. 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.
39. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS*, pages 322–331, 1986.
40. F. Wang, J. Wu, C. Huang, and K. Chang. Evolving a Test Oracle in Black-Box Testing. In *FASE*, pages 310–325, 2011.
41. H. Xiao, J. Sun, Y. Liu, S.-W. Lin, and C. Sun. Tzuyu: Learning stateful typestates. In *ASE*, pages 432–442, 2013.