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

Yong Li[a,b], Yu-Fang Chen[c], Lijun Zhang[a,b], Depeng Liu[a,b]

[a] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190*
[b] *University of Chinese Academy of Sciences, Beijing 100049*
[c] *Institute of Information Science, Academia Sinica, Taipei 11529*

Abstract

In this paper, we propose a novel algorithm to learn a Büchi automaton from a teacher who knows an ω-regular language. The learned Büchi automaton can be a nondeterministic one or a limit deterministic Büchi automaton which can be used in the verification of probabilistic systems and the program termination analysis. The algorithm is based on learning a formalism named *family of DFAs* (FDFAs) recently proposed by Angluin and Fisman [1]. 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 [1]. We implement the first publicly available library ROLL (Regular Omega Language Learning), which consists of all full 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.

Keywords: Büchi Automata, Learning Algorithm, Observation Table, Family of DFAs, Classification Tree, L^*

Email addresses: liyong@ios.ac.cn (Yong Li), yfc@iis.sinica.edu.tw (Yu-Fang Chen), zhanglj@ios.ac.cn (Lijun Zhang), liudp@ios.ac.cn (Depeng Liu)

1. Introduction

Since the last decade, learning-based automata inference techniques [2, 3, 4, 5] 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, 7, 8, 9, 10, 11, 12, 13] and *synthesizing abstract system model for further analysis* [14, 15, 16, 17, 18, 19, 20, 21, 22, 23].

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 [17, 18, 19, 24, 20], a model of system error traces for diagnosis purpose [22], behavior model of programs for statistical program analysis [23], and model-based testing and verification [14, 15, 16]. Specially, a recent position paper by Vaandrager [25] explains the concept of *model learning*, which infers an automata representation of a hardware or software system using active automata learning algorithms. The inferred automata models are used to help people better understand and diagnose a system.

Besides the classical finite automata learning algorithms, people also develop and apply learning algorithms for richer models for the above two applications. For example, learning algorithms for register automata [26, 27] have been developed and applied to synthesis systems and program interface models. A learning algorithm for timed automata has been developed for automated compositional verification for timed systems [10]. However, all the results mentioned above are for checking *safety properties* or synthesizing *finite behavior models* of systems/programs. Büchi automata are a standard model for describing liveness properties of distributed systems [28]. The model has been applied in automata theoretical model checking [29] to describe the property to be verified. They are also often used in the synthesis of reactive systems. Moreover, Büchi automata have been used as a means to prove program termination [30]. 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 promising area for further investigation.

The first learning algorithm for the full-class of ω-regular languages represented as Büchi automata was described in [31], based on the L^* algorithm [4] and the result of [32]. Recently, Angluin and Fisman propose a new learning algorithm for ω-regular languages [1] 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. To the best of our knowledge, the FDFAs have not yet been applied in the verification while the Büchi automata have already been used in several areas such as program termination [34] and probabilistic verification [35]. Nonetheless, in this paper, we show that their FDFA learning 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 [3]). The worst case storage space required by our algorithm is quadratically better than the table-based algorithm proposed in [1]. 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[36] and LearnLib[37]. 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 [1] 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.

A previous version of our learning algorithm appeared in [39]. Compared to the

previous version, we have added more examples and intuitions about the proposed learning algorithms in this version. For instance, we have added Fig. 2 in order to give the readers an idea of three different canonical FDFAs. We have provided detailed proofs and complexity analysis. Many proofs given here are not trivial so we added them in the hope that the readers may benefit from those ideas in their own works.

The major breakthrough made in this paper is that we extended the learning algorithm for Büchi automata proposed in [39] to a learning algorithm for *limit deterministic* Büchi automaton. Limit deterministic Büchi automaton is a kind of Büchi automaton introduced in [35] for qualitative verification of Markov Decision Processes (MDPs). More precisely, our learned limit deterministic Büchi automaton is very special and has two components, namely the *initial* component and the *accepting* component where two components are deterministic and all accepting states are contained in the accepting component. The nondeterminism only occurs on the transitions from the initial component to the accepting component. We are aware of that the same kind of Büchi automata is also defined in [40]. Moreover, limit deterministic Büchi automaton is also used in the program termination analysis according to [34, 41]. Therefore, we also provide the possibility to apply our learning algorithm in probabilistic verification and program analysis.

2. Preliminaries

Let $\oplus$ be the standard modular arithmetic operator. Let A and B be two sets. We use $A \ominus B$ 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 $i > k$. Given a finite word $u = u[1] \cdots u[k]$ and a word w, we denote by $u \cdot w$ the *concatenation* of u and w, i.e., the finite or infinite word $u \cdot w = u[1] \dots u[k]w[1] \dots$,

respectively. We may just write uw instead of $u \cdot w$.

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 $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for every $0 \leq i \leq n-1$ where $n \geq 1$. The run on v is *accepting* if $q_n \in F$. A word u is accepting in the FA if it has an accepting run. We also say the word u is accepted by the FA when it is accepting in the FA. A *language* is a subset of Σ^* and the language of A, denoted by $L(A)$, is the set $\{u \in \Sigma^* \mid u \text{ is accepting in } A\}$. 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 $\delta(q_0, v)$.

We call the language of a DFA or an FA a *regular* language 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. In the paper, we are particularly interested in a special ω-language called *ω-regular* language. In the following, we will introduce the representations used in this paper to recognize ω-regular languages and discuss their roles in the ω-regular language learning.

3. Representations of ω-Regular Languages

The first representation of ω-regular languages introduced here is the *Büchi automaton*, which is invented by Julius Richard Büchi and it is later widely used in model checking field. A Büchi automaton (BA) has the same structure as an FA, except that it accepts only infinite words. A run of a BA on an infinite word is an infinite sequence of states defined similarly to the case of an FA on a finite word. An infinite word w is ac-

cepted 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 $L = L(A)$. In this paper, we also consider some other kinds of BAs. A BA is a *deterministic Büchi automaton* (DBA) if $|\delta(q, a)| = 1$ for each $q \in Q$ and $a \in \Sigma$. A BA is a *limit deterministic Büchi automaton* (LDBA) if its states set Q can be partitioned into two disjoint sets $Q_N \cup Q_D$, such that 1) $\delta(q, a) \subseteq Q_D$ and $|\delta(q, a)| = 1$ if $q \in Q_D$ and $a \in \Sigma$ and 2) $F \subseteq Q_D$. Note that limit deterministic Büchi automata can also recognize ω-regular languages [35, 40] and have the same expressive power as the BAs while DBAs are strictly less expressive than BAs. For every ω-regular language L, the set of ultimately periodic words of L, i.e., UP(L), is unique.

Theorem 1 (Ultimately Periodic Words of ω-Regular Languages [42]). *Let L, L' be two ω-regular languages. Then $L = L'$ if and only if $UP(L) = UP(L')$.*

An immediate consequence of Theorem 1 is that, for any two ω-regular languages L_1 and L_2, if $L_1 \neq L_2$ then there must exist some ultimately periodic word $xy^\omega \in \text{UP}(L_1) \ominus \text{UP}(L_2)$. Therefore, Calbrix *et al.* proposed a special DFA as another representation of the ω-regular languages in [32]. More precisely, they construct a DFA $D_\$$ from a BA A to represent $L = L(A)$ such that $L(D_\$) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in \text{UP}(L)\}$ where $\$ \notin \Sigma$ is a fresh letter. Intuitively, the DFA $D_\$$ accepts every ultimately periodic word uv^ω of UP(L) in the form of a finite word $u\$v$.

Our goal in this paper is to learn the ω-regular languages via Büchi automata and our idea of learning goes back to the learning algorithm L^* by Angluin in [4]. In [4], Angluin proposed to learn the regular languages via DFAs and the *right congruence* is the theoretical foundation for it to discover states in a regular language. A right congruence is an equivalence relation $\backsim$ on Σ^* such that $x \backsim y$ implies $xv \backsim yv$ for every $x, y, v \in \Sigma^*$. We denote by $|\backsim|$ the index of $\backsim$, i.e., 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 $u \in \Sigma^*$, we denote by $[u]_\backsim$ the class of $\backsim$ in which u resides.

The main obstacle to learn ω-regular languages via Büchi automata is that there

is a lack of right congruence for Büchi automata. The first learning algorithm for the full-class of ω-regular languages represented as Büchi automata proposed in [31] circumvents this problem by using L^* algorithm to learn the DFA $D_\$$ in [32] and transform the learned $D_\$$ to a BA. Another way to bypass the obstacle is to propose right congruences for the ω-regular languages. Inspired by the work of Arnold [43], Maler and Stager [44] proposed the notion of *family of right-congruences*. Based on this, Angluin and Fisman [1] further proposed to learn ω-regular languages via a formalism called *family of DFAs*, in which every DFA corresponds to a right congruence with a finite index. Our idea to learn a BA is by learning a family of DFAs.

Definition 1 (Family of DFAs (FDFA) [1]). 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 UP($\mathcal{F}$).

Definition 2 (Acceptance condition of FDFA). 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 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 $ab \notin L(A^{M(bab)}) = L(A^\epsilon)$.

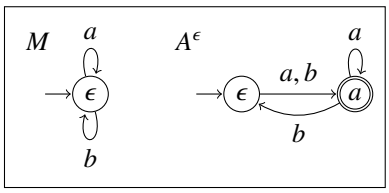


Figure 1: An example of an FDFA

For any ω-regular language L, there exists an FDFA $\mathcal{F}$ such that UP(L) = UP($\mathcal{F}$) [1]. We show in Sect. 7 that it is not the case for the reverse direction. More precisely, in [1], 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*.

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 [33]. Given a deterministic automaton $M = (\Sigma, Q, q_0, \delta)$ by ignoring its accepting states, we can define a right congruence $\backsim_M$ as follows: $x \backsim_M y$ iff $\delta(q_0, x) = \delta(q_0, y)$ for any $x, y \in \Sigma^*$.

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. The three canonical FDFAs introduced in [1] also follow the idea to recognize the ω-regular language L by the set of ultimately periodic words $\mathrm{UP}(L)$ as the DFA $D_\$$ in [32]. We now are ready to introduce the right congruences of the three canonical FDFAs defined in [1].

We first introduce the periodic FDFA, which Angluin and Fisman called in [1] the "FDFA version" of the language $L(D_\$)$ defined in [32].

Definition 3 (Periodic FDFA [1]). Given an ω-regular language L, the periodic 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 $\approx_P^u$ for progress automata A^u of the periodic FDFA as follows: $x \approx_P^u y$ iff $\forall v \in \Sigma^*. u(xv)^\omega \in L \iff u(yv)^\omega \in L$.

The progress DFA A^u is the tuple $(\Sigma, \Sigma^*/_{\approx_P^u}, [\epsilon]_{\approx_P^u}, \delta_P, F_P)$, where we have that $\delta_P([v]_{\approx_P^u}, a) = [va]_{\approx_P^u}$ for all $v \in \Sigma^*$ and $a \in \Sigma$. The accepting states F_P is the set of equivalence classes $[v]_{\approx_P^u}$ for which $uv^\omega \in L$.

It has been shown that for any $u, x, y, v \in \Sigma^*$, $xv \approx_P^u yv$ if $x \approx_P^u y$ and $\approx_P^u$ is of finite index for the given ω-regular language L [1]. Therefore $\approx_P^u$ are right congruences of finite index so one can build a transition system from each of them. Note that the syntactic right congruences and the recurrent right congruences introduced later are also of finite index.

To further explain the canonical FDFAs, we introduce another notion for the FDFAs. We say a decomposition (u, v) is *captured* by an FDFA $\mathcal{F} = (M, \{A^q\})$ if $A^{M(u)}(v)$

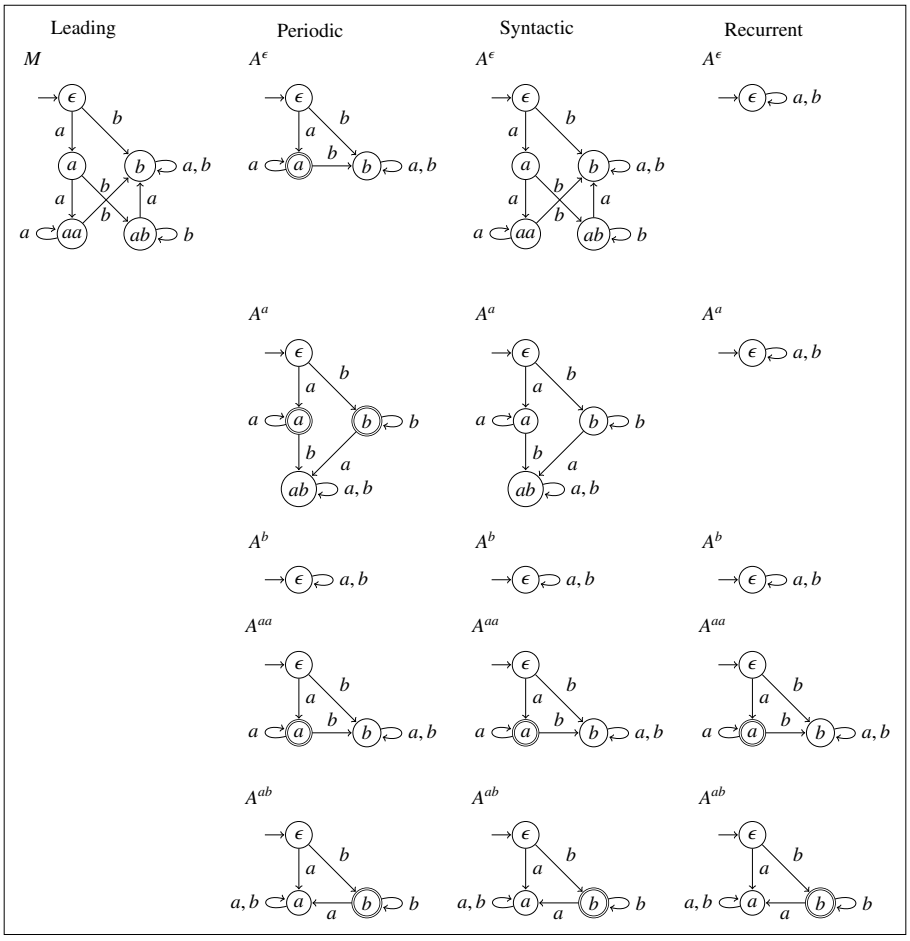


Figure 2: An example of three canonical FDFAs $\mathcal{F} = (M, \{A^q\})$ of $L = a^\omega + ab^\omega$

is an accepting state in $A^{M(u)}$ and we also say $\mathcal{F}$ *captures* the decomposition (u, v). Intuitively, the corresponding periodic FDFA of L captures every decomposition (u, v) of $uv^\omega \in \mathrm{UP}(L)$ just like the DFA $D_\$$. It follows that the language of the progress DFA A^u in the periodic FDFA is the set $\{v \in \Sigma^+ \mid uv^\omega \in L\}$. Take the periodic FDFA of $L = a^\omega + ab^\omega$ in Fig. 2 as an example where the leading automaton M is given in the column labeled "Leading" and the progress DFAs are in the column labeled "Periodic". There are three equivalent classes in the periodic right congruence $\approx_P^{aa}$, namely $[\epsilon]_{\approx_P^{aa}}$, $[a]_{\approx_P^{aa}}$ and $[b]_{\approx_P^{aa}}$. We can check that a is not in the equivalent classes $[\epsilon]_{\approx_P^{aa}}$ and $[b]_{\approx_P^{aa}}$ since there exists a finite word ϵ such that $aa(a\epsilon)^\omega \in L$ while $aa(\epsilon\epsilon)^\omega \notin L$ and $aa(b\epsilon)^\omega \notin L$. The word ϵ is not in the equivalent class $[b]_{\approx_P^{aa}}$ since there exists a such that $aa(ba)^\omega \notin L$ while $aa(\epsilon a)^\omega \in L$. There is a transition from the state a to the state b via letter b in the progress DFA A^{aa} since the word ab is in the equivalent class $[b]_{\approx_P^{aa}}$. The state a is an accepting state in A^{aa} since $aa(a)^\omega \in L$ according to Definition 3. One can easily verify that the periodic FDFA indeed captures all the decompositions of the ultimately periodic words a^ω and ab^ω and they are in the form (ϵ, a^+) (by A^ϵ), (a, a^+) (by A^a), (a, b^+) (by A^a), (aa^+, a^+) (by A^{aa}) and (ab^+, b^+) (by A^{ab}).

The second canonical FDFA is the syntactic FDFA which Angluin and Fisman construct from the family of right congruences defined by Maler and Staiger in [33]. The leading automaton in the syntactic FDFA is the same one in the periodic FDFA and they are different from each other by their definitions for the progress DFAs. Basically, given a state u in the leading automaton, the progress DFA A_P^u in the periodic FDFA of L accepts the regular language $\{v \in \Sigma^+ \mid uv^\omega \in L\}$ while the progress DFA A_S^u in the syntactic FDFA accepts the regular language $\{v \in \Sigma^+ \mid u \backsim_L uv \wedge uv^\omega \in L\}$. If we construct the leading automaton M from the right congruence $\backsim_L$, we have that $M(u) = M(uv)$ for every $v \in L(A_S^u)$. That is, the syntactic FDFA only captures the decomposition (u, v) of the ultimately periodic word $uv^\omega \in L$ such that after we reach the state $M(u)$ in M and keep reading the periodic part v, we will go back to the same state $M(u)$. This minor change of the ultimately periodic words captured in the syntactic FDFA can make a big difference since it has been shown in [1] that there exists some ω-regular language L for which the number of states in the syntactic FDFA is exponentially smaller than the number of states in the periodic FDFA.

Definition 4 (Syntactic FDFA [1]). Given an ω-regular language L, the syntactic FDFA $\mathcal{F} = (M, \{A^q\})$ of L is defined as follows.
The leading automaton M is defined the same as in Definition 3.

We define the right congruences $\approx_S^u$ for progress automata A^u of the syntactic FDFA as follows:

$$x \approx_S^u y \text{ iff } ux \backsim_L uy \text{ and } \forall v \in \Sigma^*. uxv \backsim_L u \Longrightarrow (u(xv)^\omega \in L \Longleftrightarrow u(yv)^\omega \in L).$$

The progress DFA A^u is defined similarly as in Definition 3 except that we use the equivalence relation $\approx_S^u$ for the DFA construction and the accepting states F_S is the set of equivalence classes $[v]_{\approx_S^u}$ for which $uv \backsim_L u$ and $uv^\omega \in L$.

An example of the syntactic FDFA for $L = a^\omega + ab^\omega$ is given in Fig. 2, which is also considered in [1]. In [1], the progress automaton for the state a in the syntactic FDFA is not correct since there is a transition from ab to b via letter a. However, by the definition of $\approx_S^a$ in Definition 4, aba is not in the equivalent class $[b]_{\approx_S^a}$ since $a \cdot aba \not\backsim_L a \cdot b$. That is, if aba and b have to be in the same equivalence class of $\approx_S^a$, then $a \cdot aba$ and $a \cdot b$ have to be in the same equivalence class of $\backsim_L$ first. It is easy to see that the decomposition (a, a) of a^ω is captured by the periodic FDFA and it is not captured by the syntactic FDFA in Fig. 2 since $M(a) = a \neq M(aa) = aa$.

The syntactic FDFA constructed from Definition 4 may have redundant states for some ω-regular languages. Take the DFAs A^ϵ and A^a of the syntactic FDFA in Fig. 2 as examples, we can see that they both accept nothing but $\approx_S^\epsilon$ and $\approx_S^a$ have 5 and 4 equivalent classes respectively. Therefore, Angluin and Fisman propose the recurrent FDFA [1]. The progress DFA A_R^u in the recurrent FDFA of an ω-regular language L also accepts the regular language $R^u = \{v \in \Sigma^+ \mid u \backsim_L uv \wedge uv^\omega \in L\}$ just the same as the progress DFA A_S^u of the syntactic FDFA. The difference is that A_R^u is the minimal DFA recognizing the regular language R^u.

Definition 5 (Recurrent FDFA [1]). Given an ω-regular language L, the recurrent FDFA $\mathcal{F} = (M, \{A^q\})$ of L is defined as follows.
The leading automaton M is defined the same as in Definition 3.

We define the right congruences $\approx_R^u$ for progress automata A^u of the recurrent FDFA respectively as follows:

$$x \approx_R^u y \text{ iff } \forall v \in \Sigma^*.(uxv \backsim_L u \wedge u(xv)^\omega \in L) \iff (uyv \backsim_L u \wedge u(yv)^\omega \in L).$$

The progress DFA A^u is defined the similarly as in Definition 4 except that we use the equivalence relation $\approx_R^u$ for the DFA construction.

Different from the syntactic FDFA which is associated to a Muller automaton [33], the recurrent right congruence $\approx_R^u$ of the recurrent FDFA focuses on the regular language $R^u = \{v \in \Sigma^+ \mid u \backsim_L uv \wedge uv^\omega \in L\}$ the progress DFA A^u should accept. The definition of $\approx_R^u$ is an instantiation of the right congruence $\backsim_{R^u}$ for the regular language R^u where $x \backsim_{R^u} y$ iff $\forall v \in \Sigma^*.xv \in R^u \iff yv \in R^u$ for $x, y \in \Sigma^*$. Indeed, for any $x_1, x_2 \in \Sigma^*$, we have that $x_1 \approx_R^u x_2$ iff $x_1 \backsim_{R^u} x_2$. Therefore, we can see that the right congruences $\approx_R^\epsilon$ and $\approx_R^a$ of $L = a^\omega + ab^\omega$ in Fig. 2 both have only one equivalent class, which is the only equivalent class needed for the empty language.

As mentioned before, we learn a BA by learning an FDFA since there is no corresponding right congruences for BAs. As you will see in Sect. 7, learning a small canonical FDFA often means the learned BA will be small. The reason why we keep both the periodic FDFA and the recurrent FDFA for the BA learning algorithm in the paper is due to the following facts in [1] when we fix an ω-regular language L and compare the number of states in each canonical FDFA.

- We mentioned before that the periodic FDFA can be exponentially larger than the syntactic FDFA for some ω-regular language.
- The corresponding recurrent FDFA is at least not larger than the corresponding syntactic FDFA.
- There exists some ω-regular language L such that the corresponding recurrent FDFA is larger than the corresponding periodic FDFA.

Thus, we consider both the periodic FDFA and the recurrent FDFA in the BA learning algorithm since the recurrent FDFA and the periodic FDFA are incomparable regarding the number of states. The reason why we keep the syntactic FDFA in the paper is that

according to our experiments, learning Büchi automata by the syntactic FDFAs performs quite well in practice and we observe that the progress right congruences of the syntactic FDFA have stronger ability to discover new states in the learning procedure than its other counterparts, see Fig. 6 in Sect. 6.1.

In the following, we present Proposition 1 to show all the three canonical FDFAs accept a kind of ultimately periodic words of L.

Proposition 1. *Let L be an ω-regular language, $\mathcal{F} = (M, \{A^u\})$ the corresponding periodic (syntactic, recurrent) FDFA 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 $\widetilde{v^k} = A^{\tilde{u}}(v^k)$, then we have that $v^k \approx_K^{\tilde{u}} \widetilde{v^k}$ for every $k \geq 1$ where $K \in \{P, S, R\}$. This is because $\widetilde{v^k} = A^{\tilde{u}}(\widetilde{v^k}) = A^{\tilde{u}}(v^k)$ which makes v^k in the equivalence class $[\widetilde{v^k}]$. Our goal is to prove that (u, v^k) is also accepted by $\mathcal{F}$, that is, $uv^k \backsim_M u$ and $\widetilde{v^k}$ is an accepting state for every $k \geq 1$. Given a canonical FDFA $\mathcal{F} = (M, \{A^u\})$ recognizing ω-regular language L,we say $\backsim_M$ and $\backsim_L$ are *consistent* iff for any $x, y \in \Sigma^*$, $x \backsim_M y \Leftrightarrow x \backsim_L y$. Since $\backsim_M$ and $\backsim_L$ are consistent in the three canonical FDFAs, so from the fact that (u, v) is accepted by $\mathcal{F}$, we have that $uv \backsim_M u$, i.e., $uv \backsim_L u$. It follows that $uv^k \backsim_L u$ for every $k \geq 1$. Thus, the remaining proof is to show that $\widetilde{v^k}$ 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., $\tilde{v} = A^{\tilde{u}}(v)$ is an accepting state in $A^{\tilde{u}}$, then we have $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 3. 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 $v^k \approx_P^{\tilde{u}} \widetilde{v^k}$, we conclude that $\tilde{u}(\widetilde{v^k})^\omega \in L$, which means that the state $\widetilde{v^k}$ 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 $\tilde{u}x \backsim_L \tilde{u} \wedge \tilde{u}x^\omega \in L \Longleftrightarrow \tilde{u}y \backsim_L \tilde{u} \wedge \tilde{u}y^\omega \in L$ 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 $\tilde{u}\tilde{v} \backsim_L \tilde{u}$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 4 and Definition 5. By the fact that $v \approx_K^{\tilde{u}} \tilde{v}$,

if we set $x = v$ and $y = \tilde{v}$, then we have that $\tilde{u}v \backsim_L \tilde{u}$ and $\tilde{u}(v)^\omega \in L$, which implies that $\tilde{u}v^k \backsim_L \tilde{u}$ and $\tilde{u}(v^k)^\omega \in L$ for every $k \geq 1$.

Similarly, as $v^k \approx_K^{\tilde{u}} \widetilde{v^k}$, if we set $x = v^k$ and $y = \widetilde{v^k}$, we have that $\tilde{u}\widetilde{v^k} \backsim_L \tilde{u}$ and $\tilde{u}(\widetilde{v^k})^\omega \in L$, which follows that $\widetilde{v^k}$ is an accepting state in $A^{\tilde{u}}$ for every $k \geq 1$.

■

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

Lemma 2 ([45]). *Let $\mathcal{F}$ be a periodic (syntactic, recurrent) FDFA of an ω-regular language L. One can construct a BA recognizing L from $\mathcal{F}$.*

4. Büchi Automata Learning Framework based on FDFA

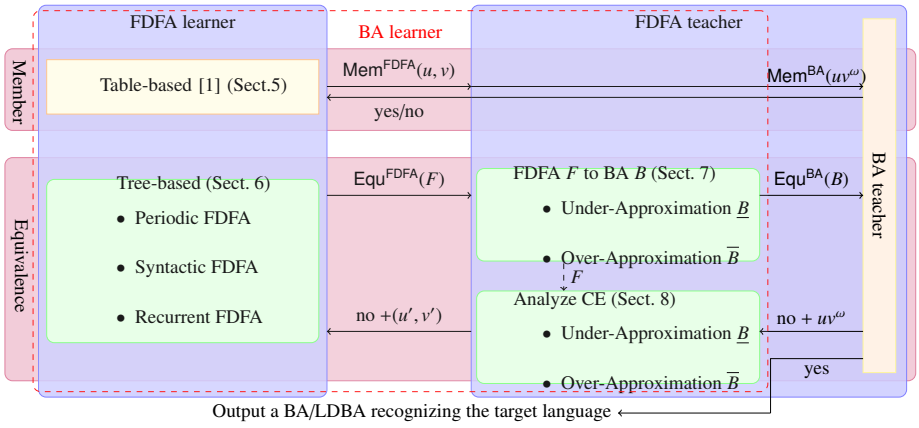


Figure 3: Overview of the learning framework based on FDFA learning. The components in boxes are results from existing works. The components in boxes are our new contributions.

We begin with an introduction of the framework of learning BA (respectively, LDBA) 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 $uv^\omega \in L \ominus L(B)$, which is possible due to Theorem 1.

The framework depicted in Fig. 3 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 [1] 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. Note that the red dashed rounded box is the BA learner we proposed in this paper.

FDFA learners: The FDFA learners component will be introduced in Sect. 5 and Sect. 6. We first briefly review the table-based FDFA learning algorithms [1] in Sect. 5. Our tree-based learning algorithm for canonical FDFAs will be introduced in Sect. 6. The algorithm is inspired by the tree-based L^* learning algorithm [3]. Nevertheless, applying the tree structure to learn FDFAs is not a trivial task. For example, instead of a binary tree used in [3], 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 Sect. 6.

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 Sect. 7, 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 Sect. 7, 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 [32]. 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 (Lemma 1 and Lemma 2). As for the over-approximation method, we only guarantee to get a BA B such that $\mathrm{UP}(L(B)) = \mathrm{UP}(F)$ if the F is a special kind of canonical FDFAs, which thus makes our learning algorithm with over-approximation method an incomplete algorithm. Nevertheless, 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" and a valid decomposition (u', v') that can be used by the FDFA learner to refine F. In Sect. 8, we show how the FDFA teacher chooses a pair (u', v') from uv^ω that allows FDFA learner to refine the current FDFA F. As the dashed line with a label F in Fig. 3 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 Sect. 8.

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. We remark that the output Büchi automaton can be a nondeterministic Büchi automaton or a limit deterministic Büchi automaton.

5. Table-based Learning Algorithm for FDFAs

In this section, we briefly introduce the table-based learner for FDFAs [1] under the assumption that we have an FDFA teacher who knows the target FDFA. It employs a structure called *observation table* [4] 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 [4], except that they use different conditions to decide if two strings belong to the same state (based on Definition 3, 4 and 5). 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 [1] for more details of the table-based algorithm.

6. Tree-based Learning Algorithm for FDFAs

In this section, we provide our tree-based learning algorithm for FDFAs and we also assume that we have an FDFA teacher knowing the target FDFA. To that end, we first define the classification tree structure for FDFA learning in Sect. 6.1 and present the tree-based algorithm in Sect. 6.2.

6.1. Classification Tree Structure in Learning

Here we present our classification tree structure for FDFA learning. Compared to the classification tree defined in [3], 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*. Intuitively, on learning a target automaton, a state $u \in \Sigma^*$ is the representative of a unique state in the target automaton we have discovered so far and if there are two words $u, u' \in \Sigma^*$ such that $u \neq u'$ and they are representatives of different states in the target automaton, then we can always find an experiment to distinguish u and u' according to the right congruence of the target automaton. For instance, we are learning the periodic FDFA of the language $L = a^\omega + ab^\omega$ depicted in Fig. 2, then in the leading automaton M, finite words a and b will be the states in

the classification tree for the leading automaton M and we have that a and b can be distinguished by an experiment (a, a) since $a \cdot aa^\omega \in L$ while $b \cdot aa^\omega \notin L$. We notice that in the classification tree for the leading automaton M, the experiments are ultimately periodic words represented by a decomposition, i.e., a pair of finite words, while the experiments in the classification trees for the progress automata are finite words. The function $L_e : I \times D \to N$ 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 and note $\mathbf{TE}(x, e)$ takes all possible inputs from $\Sigma^* \times (\Sigma^* \cup (\Sigma^* \times \Sigma^*))$. Note that since the experiments for the leading tree and the progress trees are different, we actually have $\mathbf{TE} : \Sigma^* \times (\Sigma^* \times \Sigma^*) \to D$ for the leading tree and $\mathbf{TE} : \Sigma^* \times \Sigma^* \to D$ for the progress trees. The labels of nodes in the classification tree $\mathcal{T}$ satisfy the following 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. Figure 4 depicts a leading tree $\mathcal{T}$ of the leading automaton M in Fig. 2 for $L = a^\omega + ab^\omega$. The dashed line is for the F label and the solid one is for the T label. The tree experiment function $\mathbf{TE} : \Sigma^* \times (\Sigma^* \times \Sigma^*) \to \{\mathrm{F}, \mathrm{T}\}$ is defined as $\mathbf{TE}(u, (x, y)) = \mathrm{T}$ iff $uxy^\omega \in L$ where $u, x, y \in \Sigma^*$. There are four internal nodes, namely i_1, i_2, i_3 and i_4, and five terminal nodes, namely t_1, t_2, t_3, t_4, and t_5. One can check that all nodes of $\mathcal{T}$ indeed satisfy aforementioned invariants. For instance, let $t = t_2$ be the terminal node and we have $x = L_n(t) = ab$. t_2 has two ancestors, namely i_1 and i_2. Let $t' = i_1$ and we have $e = L_n(t') = (\epsilon, a)$. The child of t' following the label $\mathbf{TE}(ab, (\epsilon, a)) = \mathrm{F}$ is i_2, which is an ancestor of t_2.

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 Sect. 2) where $u, x, y \in \Sigma^*$. Intuitively, each internal node n in $\mathcal{T}$ is labeled by an

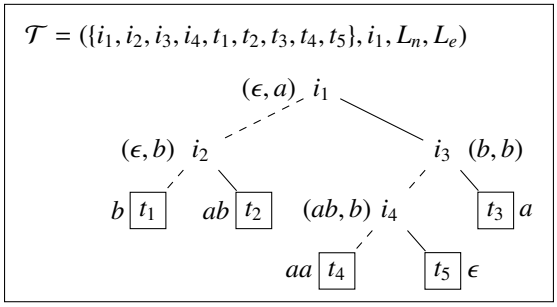


Figure 4: An example of leading classification tree $\mathcal{T}$ for $L = a^\omega + ab^\omega$.

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. One example of the leading tree $\mathcal{T}$ for the leading automaton M from Fig. 2 is depicted in Fig. 4. One can see that every label of the terminal nodes corresponds to a state in M.

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 Definition 3, 4 and 5.

Periodic FDFA: The progress tree for the 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 $x, e \in \Sigma^*$. Intuitively, for any finite words $u, u' \in \Sigma^*$ such that $u \not\approx_P^u u'$, there must exist some experiment $e \in \Sigma^*$ such that $\mathbf{TE}(u, e) \neq \mathbf{TE}(u', e)$. For instance, the progress tree $\mathcal{T}_{aa}$ for the progress DFA A^{aa} of the periodic FDFA from Fig. 2 is depicted in Fig. 5. We can see that $\mathbf{TE}(\epsilon, a) = \mathrm{T}$ since $aa(\epsilon a)^\omega \in L$ while $\mathbf{TE}(b, a) = \mathrm{F}$ since $aa(ba)^\omega \notin L$, thus in $\mathcal{T}_{aa}$, the experiment a of the internal node i_2 can distinguish states ϵ and b.

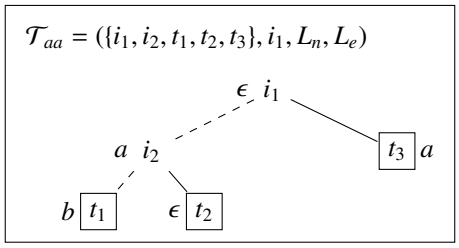


Figure 5: An example of progress classification tree $\mathcal{T}_{aa}$ for $L = a^\omega + ab^\omega$.

Syntactic FDFA: The progress tree for the 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 current leading automaton M and $K = 3|Q|$. Note that when the current leading tree $\mathcal{T}$ is fixed, we can immediately construct the corresponding leading automaton M by Definition 6 which will be given in Sect. 6.2, thus we fix the current leading automaton M in the definition of **TE** function. 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, assume 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 $u(xe)^\omega, u(ye)^\omega \in L$.

For instance, the progress tree $\mathcal{T}_{aa}$ for the progress DFA A^{aa} of the syntactic FDFA from Fig. 2 is depicted in Fig. 6. In Fig. 6, the dashed line, the dotted line and the solid line are labeled by L_e with $\mathbf{TE}(a, \epsilon) = (M(aaa), A) = (aa, A)$, $\mathbf{TE}(\epsilon, \epsilon) = (M(aa\epsilon), B) = (aa, B)$ and $\mathbf{TE}(b, \epsilon) = (M(aab), C) = (ab, C)$ respectively. Therefore, the experiment ϵ of the internal node i_1 is able to distinguish the states ϵ, a and b from each other, while for the periodic and recurrent progress trees, at least two experiments are needed to distinguish states ϵ, a and b.

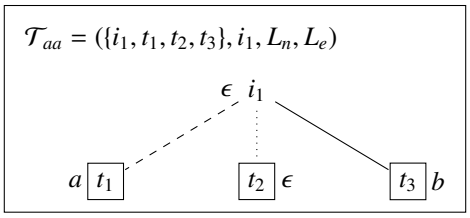


Figure 6: The progress classification tree $\mathcal{T}_{aa}$ for the syntactic FDFA of $L = a^\omega + ab^\omega$.

Recurrent FDFA: The progress tree for the 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 $x, e \in \Sigma^*$. The progress tree $\mathcal{T}_{aa}$ for the progress DFA A^{aa} in the recurrent FDFA from Fig. 2 is the same as the one depicted in Fig. 5.

6.2. 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.

Definition 6. 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 follows. 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[1]. By Definition 3, 4 and 5, the set of accepting states F of a progress automaton A^u can be identified from the structure of M with the help of membership queries where u is a state in the leading automaton M. For the periodic FDFA, $F = \{v \mid uv^\omega \in L, v \in Q\}$ and for the syntactic and the recurrent FDFA, $F = \{v \mid uv \backsim_M u, uv^\omega \in L, v \in Q\}$ where here Q is the state set of the corresponding progress DFA A^u.

Figure 7 depicts a periodic classification tree $\mathcal{T}_a$ and its corresponding progress automaton A^a for $L = a^\omega + ab^\omega$. The dashed line is for the F label and the solid one is for the T label. The tree experiment function is defined as $\mathbf{TE}(x, y) = \text{T}$ iff $a(xy)^\omega \in L$. To construct $A^a = (\Sigma, Q, \epsilon, \delta, F)$ from $\mathcal{T}_a$, one first has to construct the state set $Q = \{\epsilon, a, b, ab\}$ by collecting all terminal labels in $\mathcal{T}_a$. As for the transition function, we give an example to further illustrate it. For instance, $\delta(b, a)$ is decided by classifying the word ba to one of the terminal nodes in $\mathcal{T}_a$. Starting with the root i_1, we have $\mathbf{TE}(ba, L_n(i_1)) = \mathbf{TE}(ba, \epsilon) = \text{F}$ since $a \cdot (ba \cdot \epsilon)^\omega \notin L$. Therefore, we go to the F-child of i_1, namely i_2. Since i_2 is not a terminal node and we have that $\mathbf{TE}(ba, L_n(i_2)) = \mathbf{TE}(ba, b) = \text{F}$, we go further to its F-child and finally reach the terminal node ab. Thus, we conclude that $\delta(b, a) = ab$. We collect the accepting states

[1]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.

from Q by identifying the state $v \in Q$ such that $av^\omega \in L$ according to the construction. Therefore, we have $F = \{a, b\}$ since $aa^\omega \in L$ and $ab^\omega \in L$.

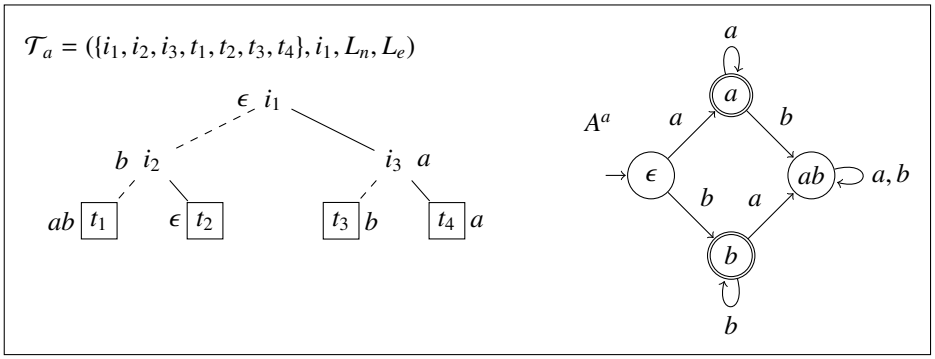


Figure 7: An example of periodic progress classification tree $\mathcal{T}_a$ and periodic progress automaton A^a for $L = a^\omega + ab^\omega$.

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 7 (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 $uv \backsim_M u$, $uv^\omega \in \mathrm{UP}(L)$, and (u, v) is not accepted by $\mathcal{F}$,
- (u, v) is *negative* if $uv \backsim_M u$, $uv^\omega \notin \mathrm{UP}(L)$, 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 [1]. One can check that our returned counterexample for FDFA learner constructed in Sect. 8 respects Definition 7. The way we analyze the counterexamples is very similar to the one used in the table-based FDFA learning algorithm [1] so we also follow their way to present the counterexample analysis for the FDFA learner.

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 $j > k$ defined in Sect. 2. Therefore, there must exist a smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1}u[j], (u[j+1 \cdots n], v)) \neq \mathbf{TE}(s_j, (u[j+1 \cdots n], v))$. 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. 8. 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 distinguish 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. 8.*

Refinement for the progress tree: Recall that $\tilde{u} \cdot v^\omega \notin \mathrm{UP}(L)$ and thus the algorithm needs to 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 $A^{\tilde{u}}$ 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. Assume that we have

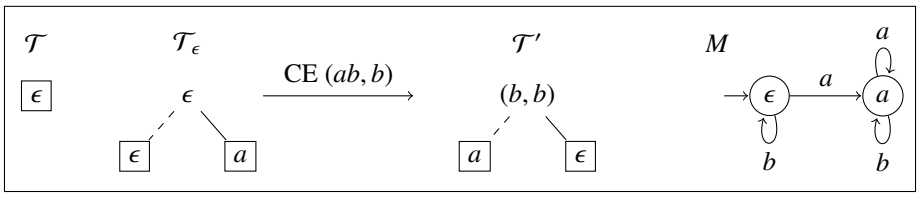


Figure 8: Refinement of the leading tree and the corresponding leading automaton

$\mathbf{TE}(s_0 v[1 \cdots n], \epsilon) = \mathbf{TE}(v[1 \cdots n], \epsilon) \neq \mathbf{TE}(s_n, \epsilon) = \mathbf{TE}(s_n, v[n+1 \cdots n])$, there must exist a smallest $j \in [1 \cdots n]$ such that $\mathbf{TE}(s_{j-1} v[j], v[j+1 \cdots n]) \neq \mathbf{TE}(s_j, v[j+1 \cdots n])$. 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 $\mathcal{T}_{\tilde{u}}$.

Therefore, 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 show that $\mathbf{TE}(s_0 v, \epsilon) \neq \mathbf{TE}(s_n, \epsilon)$ indeed holds.

- For the periodic FDFA, we have $\mathbf{TE}(v, \epsilon) = \mathrm{F}$ since $\tilde{u}(v \cdot \epsilon)^\omega \notin \mathrm{UP}(L)$. Since $\tilde{v}$ is an accepting state, we have $\mathbf{TE}(\tilde{v}, \epsilon) = \mathrm{T}$.

- For the syntactic FDFA, we have that $uv \backsim_M u$ according to Definition 7, that is, $\tilde{u} = M(uv) = M(u) = M(\tilde{u}v)$.

  First, we have $\mathbf{TE}(v, \epsilon) = (M(\tilde{u} \cdot v), \mathrm{B}) = (\tilde{u}, \mathrm{B})$, where B is obtained here since $\tilde{u} = M(\tilde{u} \cdot v \cdot \epsilon)$ and $\tilde{u}(v \cdot \epsilon)^\omega \notin \mathrm{UP}(L)$ according to the definition of $\mathbf{TE}$ in the syntactic FDFA.

  Since $\tilde{v}$ is an accepting state in the current syntactic FDFA, it follows that $\tilde{u} = M(\tilde{u}\tilde{v})$ and $\tilde{u}(\tilde{v})^\omega \in L$ according to Definition 4. Thus, we have $\mathbf{TE}(\tilde{v}, \epsilon) = (M(\tilde{u}\tilde{v}), \mathrm{A}) = (\tilde{u}, \mathrm{A})$ where A is obtained since $\tilde{u} = M(\tilde{u} \cdot \tilde{v} \cdot \epsilon)$ and $\tilde{u}(\tilde{v} \cdot \epsilon)^\omega \in \mathrm{UP}(L)$.

- For the recurrent FDFA, similar as in the syntactic FDFA, we have $\mathbf{TE}(v, \epsilon) = \mathrm{F}$ and $\mathbf{TE}(\tilde{v}, \epsilon) = \mathrm{T}$.

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, one can repeatedly use the counterexample until it is eliminated.

We introduce an immediate result of the counterexample guided refinement for $\mathcal{F}$ as Lemma 3. It shows that the tree-based learning algorithm will make progress upon receiving a counterexample, which is an important property for the termination of the learning algorithm.

Lemma 3. *During the learning procedure, if the tree-based FDFA learner receives a counterexample (u, v), then there will be a new state added to the leading automaton M or the corresponding progress automaton $A^{\tilde{u}}$ where $\tilde{u} = M(u)$.*

7. 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 $\mathrm{UP}(L(B)) = \mathrm{UP}(\mathcal{F})$.

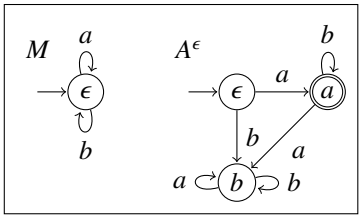


Figure 9: An FDFA $\mathcal{F}$ such that $\mathrm{UP}(\mathcal{F})$ does not characterize an ω-regular language

Example 2. *Consider a non-canonical FDFA $\mathcal{F}$ in Fig. 9, we have $UP(\mathcal{F}) = \bigcup_{n=0}^{\infty}\{a, b\}^* \cdot (ab^n)^\omega$. 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 [1]. However, we can show that the right congruence $\approx_P^\epsilon$ of a periodic FDFA of L, if exists, has to be 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 $\approx_P^\epsilon$ is of infinite index. We conclude that $UP(\mathcal{F})$ cannot characterize an ω-regular language.*

Therefore, in general, we cannot construct a BA B from an FDFA $\mathcal{F}$ such that $\mathrm{UP}(L(B)) = \mathrm{UP}(\mathcal{F})$. We propose a BA $\underline{B}$, which underapproximates the ultimately periodic words of an FDFA. For the under-approximation method, upon receiving a counterexample from the BA teacher each time, our FDFA teacher can always find a valid counterexample for FDFA learner defined in Definition 7 to refine the current FDFA. Moreover, if $\mathcal{F}$ is a canonical FDFA, the under-approximation method guarantees to construct a BA $\underline{B}$ such that $\mathrm{UP}(L(\underline{B})) = \mathrm{UP}(\mathcal{F})$, which makes it a complete method for BA learning. Another method is to construct a BA $\overline{B}$ which overapproximates the ultimately periodic words of the FDFA $\mathcal{F}$. For the over-approximation method, given a canonical FDFA $\mathcal{F}$, that whether $\mathrm{UP}(L(\overline{B})) = \mathrm{UP}(\mathcal{F})$ is still unknown and it can be incomplete in the sense that it may not be able to find the valid counterexamples for FDFA learner when dealing with counterexamples returned from the BA teacher. Nevertheless, we show that the over-approximation method guarantees to construct a BA $\overline{B}$ such that $\mathrm{UP}(L(\overline{B})) = \mathrm{UP}(\mathcal{F})$ for a special kind of canonical FDFA $\mathcal{F}$. We keep the over-approximation method since the size of the corresponding $\overline{B}$ is quadratic in the size of the given FDFA while the size of $\underline{B}$ is cubic according to Lemma 5.

We first give the main idea behind the two approximation methods and then give the formal definition of the methods in the following. 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 $M^s_v = (\Sigma, Q, s, \delta, \{v\})$ and $(A^u)^s_v = (\Sigma, Q_u, s, \delta_u, \{v\})$, i.e., the DFA obtained from M and A^u by setting their initial state and accepting states as s and $\{v\}$, respectively. We define $N_{(u,v)} = \{v^\omega \mid uv \backsim_M u \wedge v \in L((A^u)^{s_u}_v)\}$, which includes only the word $v \in L((A^u)^{s_u}_v)$ such that $u = M(u) = M(uv)$. Therefore, according to Definition 2, we have that $\mathrm{UP}(\mathcal{F}) = \bigcup_{u \in Q, v \in F_u} L(M^{q_0}_u) \cdot N_{(u,v)}$ where $L(M^{q_0}_u)$ contains the finite prefixes and $N_{(u,v)}$ contains the periodic finite words for every state pair (u, v).

We construct $\overline{B}$ and $\underline{B}$ by approximating the set $N_{(u,v)}$. For $\overline{B}$, we first define an FA $\overline{P}_{(u,v)} = (\Sigma, Q_{(u,v)}, s_{(u,v)}, \{f_{(u,v)}\}, \delta_{(u,v)}) = M^u_u \times (A^u)^{s_u}_v$ 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 $\delta_f = \{(f, \epsilon, s_{(u,v)}), (f_{(u,v)}, \epsilon, f)\}$. Note that ϵ transitions can be taken without consuming any letters and can be removed by standard methods in automata theory. Intuitively, we overapproximate the set $N_{(u,v)}$ as $\overline{N}_{(u,v)}$ by adding

$(v_1 \cdot v_2)^\omega$ into $N_{(u,v)}$ if $(v_1)^\omega \in N_{(u,v)}$ and $(v_2)^\omega \in N_{(u,v)}$ where $v_1, v_2 \in \Sigma^+$. 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 $\overline{N}_{(u,v)}$. Intuitively, we underapproximate the set $N_{(u,v)}$ as $\underline{N}_{(u,v)}$ by only keeping $v^\omega \in N_{(u,v)}$ if $A^u(v) = A^u(v \cdot v)$ where $v \in \Sigma^+$. In Definition 8 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 8. 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

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

Intuitively, we connect the leading automaton M to the BA recognizing $\underline{N}_{(u,v)}$ (respectively $\overline{N}_{(u,v)}$) by linking the state u of M and the initial state $s_{(u,v)}$ of the BA with an ϵ-transition for every state pair (u, v) where $v \in F_u$.

Figure 10 depicts the BAs $\overline{B}$ and $\underline{B}$ constructed from the FDFA $\mathcal{F}$ in Fig. 1. In the example, we can see that $b^\omega \in \mathrm{UP}(\mathcal{F})$ while $b^\omega \notin \mathrm{UP}(L(\underline{B}))$.

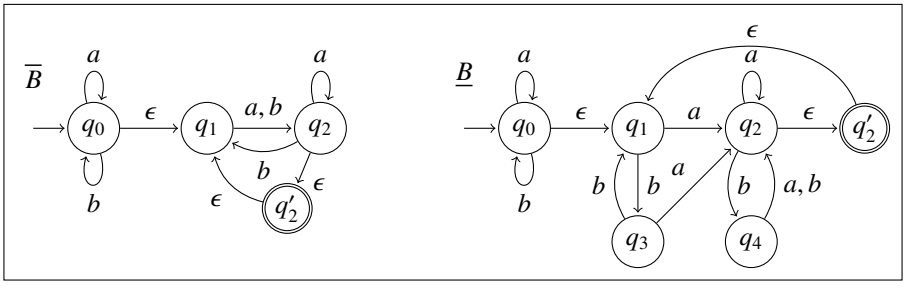


Figure 10: NBA $\overline{B}$ and $\underline{B}$ for $\mathcal{F}$ in Fig. 1

In the following, we introduce Lemma 4, which will be used to prove Lemma 5.

Lemma 4. *Given an FDFA $\mathcal{F} = (M, \{A^u\})$, and $\underline{B}$ the BA constructed from $\mathcal{F}$ by Definition 8. If (u, v^k) is accepted by $\mathcal{F}$ for every $k \geq 1$, then $uv^\omega \in UP(L(\underline{B}))$.*

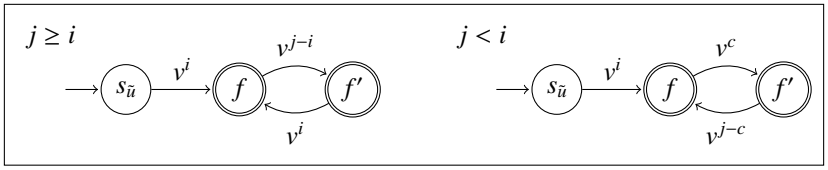


Figure 11: Finding v^k. If $j \geq i$, we let $k = j$, otherwise let $c = (l \cdot j - i) \oplus j \geq 0$ where $k = l \cdot j \geq i$ for some $l \geq 1$ and recall that $\oplus$ is the standard modular arithmetic operator.

Proof. From the assumption that (u, v^k) is accepted by $\mathcal{F}$ for every $k \geq 1$, we have $uv^k \backsim_M u$ and $v^k \in L(A^{\tilde{u}})$ for any $k \geq 1$ where $\tilde{u} = M(u)$ by Definition 2. Recall that $M^{\tilde{u}}_{\tilde{u}}$ is obtained from M by setting its initial and final state to $\tilde{u}$. It follows that

$$v^k \in L(M^{\tilde{u}}_{\tilde{u}}) \tag{1}$$

for every $k \geq 1$ since $uv^k \backsim_M u$. 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 DFA. In other words, there is a loop in the run of $A^{\tilde{u}}$ over v^n. Without loss of generality, suppose there exist $i, j \geq 1$ with $i + j = n$ such that $f = A^{\tilde{u}}(v^i) = A^{\tilde{u}}(v^{i+j})$, which is depicted in Fig. 11.

In the following, our goal is to find some accepting state f' in the loop such that $f' = A^{\tilde{u}}(v^k) = A^{\tilde{u}}(v^{2k})$ for some $k \geq 1$. Figure 11 shows how to find the accepting state f' along the loop in following two cases.

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

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 state f' is an accepting state. Therefore we have that

$$v^k \in L((A^{\tilde{u}})^{s_{\tilde{u}}}_{f'}) \wedge v^k \in L(A^{f'}_{f'}) \tag{2}$$

From (1) and (2), we conclude that v^k is accepted by the product $\underline{P}_{(\tilde{u},f')}$ of three automata $M^{\tilde{u}}_{\tilde{u}}$, $(A^{\tilde{u}})^{s_{\tilde{u}}}_{f'}$ and $(A^{\tilde{u}})^{f'}_{f'}$ where $s_{\tilde{u}}$ is the initial state of $A^{\tilde{u}}$. In other words, the ω-word uv^ω will be accepted in $\underline{B}$ since $u \cdot (v^k)^\omega \in L(M^{q_0}_{\tilde{u}}) \cdot (L(\underline{P}_{(\tilde{u},f')}))^\omega$. Therefore we complete the proof. ∎

Lemma 5 (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 Definition 8. 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.*

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 $v \in F_u$, thus $\underline{B}$ 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 $v \in F_u$. Therefore $\overline{B}$ is of size $n + nk \cdot nk \in O(n^2k^2)$.

- $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$. Let the ultimately periodic ω-word w be a word accepted by $\underline{B}$, i.e., $w \in \mathrm{UP}(L(\underline{B})) = \bigcup_{u \in Q, v \in F_u} L(M_u^{q_0}) \cdot (L(\underline{P}_{(u,v)}))^\omega$. Therefore, there exists some $\tilde{u}$ and $\tilde{v}$ such that $w = u \cdot v_1 \cdot v_2 \cdot v_n \cdots$ where $u \in L(M_{\tilde{u}}^{q_0})$ and $v_i \in L(\underline{P}_{(\tilde{u},\tilde{v})})$ for every $i \geq 1$. According to Definition 8, $\underline{P}_{(\tilde{u},\tilde{v})}$ is the product of three automata $M_{\tilde{u}}^{\tilde{u}}$, $(A^{\tilde{u}})_{\tilde{v}}^{s_{\tilde{u}}}$ and $(A^{\tilde{u}})_{\tilde{v}}^{\tilde{v}}$ where $s_{\tilde{u}}$ is the initial state in $A^{\tilde{u}}$. It follows that

$$L(M_{\tilde{u}}^{q_0}) \cdot (L(\underline{P}_{(\tilde{u},\tilde{v})}))^* = L(M_{\tilde{u}}^{q_0}) \tag{1}$$

and

$$(L(\underline{P}_{(\tilde{u},\tilde{v})}))^+ = L(\underline{P}_{(\tilde{u},\tilde{v})}) \tag{2}$$

. $L(M_{\tilde{u}}^{q_0}) \cdot (L(\underline{P}_{(\tilde{u},\tilde{v})}))^* = L(M_{\tilde{u}}^{q_0})$ can be justified by the fact that $L(\underline{P}_{(\tilde{u},\tilde{v})}) \subseteq L(M_{\tilde{u}}^{\tilde{u}})$, which implies that $L(M_{\tilde{u}}^{q_0}) \cdot (L(\underline{P}_{(\tilde{u},\tilde{v})}))^* \subseteq L(M_{\tilde{u}}^{q_0})$. Moreover, $\epsilon \in (L(\underline{P}_{(\tilde{u},\tilde{v})}))^*$ and it follows that $L(M_{\tilde{u}}^{q_0}) \subseteq L(M_{\tilde{u}}^{q_0}) \cdot (L(\underline{P}_{(\tilde{u},\tilde{v})}))^*$. Thus, we have proved that (1) holds. Similarly, we can prove $(L(\underline{P}_{(\tilde{u},\tilde{v})}))^+ \subseteq L(\underline{P}_{(\tilde{u},\tilde{v})})$ by the facts that $(L((A^{\tilde{u}})_{\tilde{v}}^{\tilde{v}}))^+ \subseteq L((A^{\tilde{u}})_{\tilde{v}}^{\tilde{v}})$ and $L(\underline{P}_{(\tilde{u},\tilde{v})}) \subseteq L((A^{\tilde{u}})_{\tilde{v}}^{\tilde{v}})$. Together with the fact that $L(\underline{P}_{(\tilde{u},\tilde{v})}) \subseteq (L(\underline{P}_{(\tilde{u},\tilde{v})}))^+$, we conclude that (2) holds.

  By Lemma 5 in [32], let $UV^* = U$ and $V^+ = V$ where $U, V \subseteq \Sigma^*$. Then if $w \in \mathrm{UP}(UV^\omega)$, there must exist $u \in U$ and $v \in V$ such that $w = uv^\omega$. Thus we

let $U = L(M^{q_0}_{\tilde{u}})$ and $V = L(\underline{P}_{(\tilde{u},\tilde{v})})$, we have that $UV^* = U$ (1) and $V^+ = V$ (2). Since $w \in L(UV^\omega)$, there exist two words $x \in L(M^{q_0}_{\tilde{u}})$ and $y \in L(\underline{P}_{(\tilde{u},\tilde{v})})$ such that $w = x \cdot y^\omega$. In other words, we let $\tilde{u} = M(x)$ and then we have $xy \backsim_M x$ and $y \in L(A^{\tilde{u}})$ since $x \in L(M^{q_0}_{\tilde{u}})$, $y \in L(M^{\tilde{u}}_{\tilde{u}})$ and $L(\underline{P}_{(\tilde{u},\tilde{v})}) \subseteq L(A^{\tilde{u}})$. It follows that w is accepted by $\mathcal{F}$.

- $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\overline{B}))$. Suppose an ω-word $w \in \mathrm{UP}(\mathcal{F})$, then there exists a decomposition (u, v) of w such that $uv \backsim_M u$ and $\tilde{v}$ is an accepting state where $\tilde{u} = M(u)$ and $\tilde{v} = A^{\tilde{u}}(v)$. It follows that $u \in L(M^{q_0}_{\tilde{u}})$ since $\tilde{u} = M(u)$. Further, we have $v \in L(\overline{P}_{(\tilde{u},\tilde{v})})$ since $\overline{P}_{(\tilde{u},\tilde{v})} = M^{\tilde{u}}_{\tilde{u}} \times (A^{\tilde{u}})^{s_{\tilde{u}}}_{\tilde{v}}$ according to Definition 8 where $s_{\tilde{u}}$ is the initial state of $A^{\tilde{u}}$. It follows that $u \cdot v^\omega \in L(M^{q_0}_{\tilde{u}}) \cdot (L(\overline{P}_{(\tilde{u},\tilde{v})}))^\omega \subseteq \mathrm{UP}(L(\overline{B}))$.

- $\mathrm{UP}(L(\underline{B})) = \mathrm{UP}(\mathcal{F})$ if $\mathcal{F}$ is a canonical FDFA. For any FDFA $\mathcal{F}$, we have $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$. Thus, the remaining job is to prove that $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\underline{B}))$ if $\mathcal{F}$ is a canonical FDFA, which directly follows from Proposition 1 and Lemma 4. Thus, we complete the proof.

■

Lemma 7 introduces a special kind of canonical FDFA $\mathcal{F}$ for which the over-approximation method produces a BA $\overline{B}$ such that $\mathrm{UP}(\overline{B}) = \mathrm{UP}(\mathcal{F})$. We will first introduce Lemma 6 to prove Lemma 7. Note that Lemma 6 is also used to analyze counterexamples in Sect. 8.1.

Lemma 6. *Given an FDFA $\mathcal{F} = (M, \{A^u\})$ and an ω-word $w \in UP(L(\overline{B}))$ where $\overline{B}$ is constructed from $\mathcal{F}$ by Definition 8. We can construct 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 $uv_i \backsim_M u$.*

Proof. Since we only consider ultimately periodic ω-words in $\overline{B}$, every ω-word can be given by one of its decomposition.

Since $\mathrm{UP}(L(\overline{B})) = \bigcup_{u \in Q, p \in F_u} L(M^{q_0}_u) \cdot (L(\overline{P}_{(u,p)}))^\omega$, suppose ω-word $w = uv^\omega \in \mathrm{UP}(L(\overline{B}))$, then w can be given by a decomposition (u, v) such that $u \in L(M^{q_0}_{\tilde{u}})$ and $v \in (L(\overline{P}_{(\tilde{u},p)}))^+$ 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 $\overline{P}_{(\tilde{u},p)} = M^{\tilde{u}}_{\tilde{u}} \times (A^{\tilde{u}})^{s_{\tilde{u}}}_p$,

we conclude that $uv_i \backsim_M u$ 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 Sect. 8.2 where $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u'v'^\omega = uv^\omega\}$ and \$ is not a letter in Σ. We note that the number of states in $\mathcal{D}_{u\$v}$ is in $O(|v|(|v| + |u|))$ (see Sect. 8.2). In addition, we can construct an FA $\mathcal{A}$ such that $L(\mathcal{A}) = \bigcup_{u\in Q, p\in F_u} L(M_u^{q_0}) \cdot \$ \cdot (L(M_u^u \times (A^u)_p^{s_u}))^+$ where s_u is the initial state of A^u. By fixing a state u in M and an accepting state p of A^u, we can construct an FA $\mathcal{A}_{(u,p)}$ such that $L(\mathcal{A}_{(u,p)}) = L(M_u^{q_0}) \cdot \$ \cdot (L(M_u^u \times (A^u)_p^{s_u}))^+ = L(M_u^{q_0}) \cdot \$ \cdot (L(\overline{P}_{(u,p)}))^+$. Recall that in the overapproximation construction, $\overline{P}_{(u,p)}$ is defined as $M_u^u \times (A^u)_p^{s_u}$. We can identify 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 7. *Given a canonical FDFA $\mathcal{F} = (M, \{A^u\})$, for any progress DFA A^u in $\mathcal{F}$ and any accepting state x of A^u with $x = A^u(x_1) = \cdots = A^u(x_n)$ for $n \geq 1$, we have $x_1 \cdots x_n \in L(A^u)$. Then $UP(\mathcal{F}) = UP(L(\overline{B}))$ where $\overline{B}$ is a BA constructed from $\mathcal{F}$ by the over-approximation method in Definition 8.*

Proof. By Lemma 5, we have $\mathrm{UP}(\mathcal{F}) \subseteq \mathrm{UP}(L(\overline{B}))$ for $\mathcal{F}$. For any ω-word $xy^\omega \in \mathrm{UP}(L(\overline{B}))$, by Lemma 6, we know there is a decomposition $(u, v_1 \cdots v_n)$ of xy^ω for some $n \geq 1$ such that for $i \in [1 \cdots n]$, $v_i \in L(A^{M(u)})$ and $uv_i \backsim_M u$. From the assumption, we know that $v_1 \cdots v_n \in L(A^{M(u)})$. It follows that $uv_1 \cdots v_n \backsim_M u$ and $v_1 \cdots v_n \in L(A^{M(u)})$, which indicates that $u(v_1 \cdots v_n)^\omega$ is accepted by $\mathcal{F}$. Therefore, $xy^\omega \in \mathrm{UP}(\mathcal{F})$, i.e., we have $\mathrm{UP}(L(\overline{B})) \subseteq \mathrm{UP}(\mathcal{F})$. Thus, we complete the proof. ■

Take the three canonical FDFAs depicted in 2, it is easy to see that all of them satisfy Lemma 7. Therefore, we can also use over-approximation method to get the BAs which recognize the ω-regular language $a^\omega + ab^\omega$ from them.

7.1. From FDFA to Limit Deterministic Büchi Automaton

Recall that in the under-approximation (respectively, over-approximation) method, we need first construct an FA $\underline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u} \times (A^u)_v^v$ (respectively $\overline{P}_{(u,v)} = M_u^u \times (A^u)_v^{s_u}$) and then construct an NBA recognizing $L(\underline{P}_{(u,v)})^\omega$ (respectively, $L(\overline{P}_{(u,v)})^\omega$).

In this section, we show that we can construct a DBA A instead of a BA defined in Definition 8 recognizing $L(\underline{P}_{(u,v)})^\omega$ (respectively, $L(\overline{P}_{(u,v)})^\omega$), which yields a limit deterministic Büchi automaton from the given FDFA $\mathcal{F}$.

To make our construction more general, in the following we construct a DBA A with $L(A) = L(D)^\omega$ from a DFA D with only one accepting state. One can check that the FAs $\underline{P}_{(u,v)}$ and $\overline{P}_{(u,v)}$ from the under-approximation and the over-approximation methods indeed are DFAs with one accepting state.

Definition 9. Given a DFA $D = (\Sigma, Q, q_0, q_f, \delta)$ where every state $q \in Q$ can be reached by q_0 and can reach q_f. The DBA A is a tuple $(\Sigma, Q', q_0, \{[q_f]\} \cup \{(q_f, q) \mid q \in Q\}, \delta')$ where

$$Q' = Q \cup (Q \times Q) \cup \{[q] \mid q \in Q\} \cup \{\langle q \rangle \mid q \in Q\}$$

and δ' is defined as follows where $q, q' \in Q$ and $a \in \Sigma$:

1.
$$\delta'(q,a) = \begin{cases} \delta(q,a) & q \neq q_f; \\ (\delta(q_0,a), \delta(q_f,a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) = \emptyset; \\ \langle \delta(q_f,a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q_f,a) \neq \emptyset. \end{cases}$$

2.

$$\delta'((q,q'),a) = \begin{cases} (\delta(q,a),\delta(q',a)) & q \neq q_f \wedge \delta(q,a) \neq \emptyset \wedge \delta(q',a) \neq \emptyset; \\ \langle \delta(q',a) \rangle & q \neq q_f \wedge \delta(q,a) = \emptyset \wedge \delta(q',a) \neq \emptyset; \\ [\delta(q,a)] & q \neq q_f \wedge \delta(q,a) \neq \emptyset \wedge \delta(q',a) = \emptyset; \\ (\delta(q_0,a),\delta(q',a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q',a) \neq \emptyset; \\ \langle \delta(q',a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q',a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q',a) = \emptyset. \end{cases}$$

3.

$$\delta'([q],a) = \begin{cases} [\delta(q,a)] & q \neq q_f; \\ (\delta(q_0,a),\delta(q_f,a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) = \emptyset. \\ \langle \delta(q_f,a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q_f,a) \neq \emptyset. \end{cases}$$

4.

$$\delta'(\langle q \rangle,a) = \begin{cases} \langle \delta(q,a) \rangle & q \neq q_f; \\ (\delta(q_0,a),\delta(q_f,a)) & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) \neq \emptyset; \\ [\delta(q_0,a)] & q = q_f \wedge \delta(q_0,a) \neq \emptyset \wedge \delta(q_f,a) = \emptyset. \\ \langle \delta(q_f,a) \rangle & q = q_f \wedge \delta(q_0,a) = \emptyset \wedge \delta(q_f,a) \neq \emptyset. \end{cases}$$

Note that the transition function δ may not be complete, i.e., $\delta(q,a) = \emptyset$ for some $q \in Q$ and $a \in \Sigma$. Note that we also omit the cases such that $\delta'(q,a) = \emptyset$ where $q' \in Q'$ and $a \in \Sigma$, such as $\delta'((q,q'),a) = \emptyset$ when $\delta(q,a) = \emptyset \wedge \delta(q',a) = \emptyset$ with $q,q' \in Q$ and $a \in \Sigma$. One can check that the definition δ' is well defined in the sense that all possible situations are taken account of.

Intuitively, suppose $U = L(D)$ and $K = L(D^{q_f}_{q_f})$ where $D^{q_f}_{q_f}$ is obtained from DFA D by setting q_f as the initial state and the accepting state. We divide the language $U \cup K$ into three parts, namely $U \backslash K = \{u \in \Sigma^* \mid u \in U \wedge u \notin K\}$, $U \cap K = \{u \in \Sigma^* \mid u \in U \wedge u \in K\}$ and $K \backslash U = \{u \in \Sigma^* \mid u \notin U \wedge u \in K\}$.

Consider the run r of A over ω-word u^ω where $u \in U$. After reading u, r reaches the accepting state q_f and still wants to continue the run, and we try to make it starting

from q_0 again by starting at (q_0, q_f). The first element q_0 of the state pair performs the same behaviors starting at the initial state q_0, while the second element q_f tries to mimic the same behaviors by starting from the accepting state q_f. If $u \in U \cap K$, then u can reach an accepting state (q_f, q_f) by starting at state (q_0, q_f). A finite word $u \in U \backslash K$ may be detected in the run r by observing that if there exists some state $(q_1, q_2) = \delta'((q_0, q_f), u_1)$ such that $\delta(q_1, a)$ is defined and $\delta(q_2, a)$ is not defined where $u = u_1 a u_2$. In order to track the remaining behaviors of u, i.e., the word u_2, we make use of states in form of $[q]$. Similarly, a finite word $u \in K \backslash U$ may be detected by using a symmetric argument. Therefore, we use states in form of $\langle q \rangle$ to keep track of the rest behaviors of u in such a case. Moreover, every time we encounter the states $[q_f]$, (q_f, q) and $\langle q_f \rangle$, we try to make them mimic the behaviors of q_0 if possible. In this way, all the ω-words visiting $[q_f]$ or (q_f, q) infinitely often can be rewritten as the concatenation of infinitely many finite words which are accepted by D.

Theorem 2. *Given a DFA $\mathcal{D}$ with a single accepting state and let A be the DBA constructed by Definition 9, then $L(A) = L(D)^\omega$.*

Proof. Now we prove the theorem by two directions. Recall that we define $U = L(D)$ and $K = L(D^{q_f}_{q_f})$. According to Theorem 1, we only consider ultimately periodic words of ω-regular languages.

1. $\mathrm{UP}(U^\omega) \subseteq \mathrm{UP}(L(A))$. Let $w = (u_0 u_1 \cdots u_i \cdots u_n)^\omega \in \mathrm{UP}(U^\omega)$ where $u_i \in U$ for any $0 \leq i \leq n$ and $n \geq 0$. The goal is to prove that the corresponding run r of A over w visits $[q_f]$ or (q_f, q) for infinitely many times for some state $q \in Q$. According to the transition relation δ', any state $q \in Q$ will not be reached again once r touches the accepting state q_f. Starting from q_0 after reading $u_0 u_1$, it will either stop at the state $[q_f]$, $\langle q_f \rangle$ or state (q_f, q) for some $q \in Q$. In the following, we show that starting from state $[q_f]$, $\langle q_f \rangle$ or state (q_f, q), it will visit (q_f, q) or $[q_f]$ at least once and stop at either $[q_f]$, $\langle q_f \rangle$ or state (q_f, q) after reading arbitrary $u \in U$.

    - If $u \in U \cap K$, then $\delta'([q_f], u) = \delta'(\langle q_f \rangle, u) = \delta'((q_0, q_f), u) = (q_f, q_f)$ and $\delta'((q_f, q), u) = (q_f, p)$ for some $p, q \in Q$ according to Definition 9.

- Otherwise $u \in U \backslash K$. According to Definition 9, upon reading word u, $[q_f]$ and $\langle q_f \rangle$ will restart at (q_0, q_f), while (q_f, q) will restart at (q_0, q). After reading u, the run r will either stop at $[q_f]$ or (q_f, q') for some $q' \in Q$. The reason is that starting from the first element q_0 of (q_0, q_f) and (q_0, q), the run will not visit any states in form of $\langle q \rangle$ since $u \in U \backslash K$.

Therefore, with infinitely many $u \in U$ in w, the run r will visit some accepting state in A infinitely often, which implies that $w \in \mathrm{UP}(L(A))$.

2. $\mathrm{UP}(L(A)) \subseteq \mathrm{UP}(U^\omega)$. Suppose ultimately periodic word w is accepted by A and one of its corresponding run r is in the form of $q_0 \xrightarrow{u_0} q_1 \xrightarrow{u_1} q_2 \xrightarrow{u_2} \cdots q_h \xrightarrow{u_h} q_{h+1} \xrightarrow{u_{h+1}} \cdots q_{n-1} \xrightarrow{u_{n-1}} q_n \xrightarrow{u_n} q_h$ where $u_i \in \Sigma^+$ and $q_i \in \{(q_f, p), [q_f], \langle q_f \rangle\}$ and $p \in Q$ for every $1 \leq i \leq n$ and q_h is the first accepting state which occurs infinitely often. Here we have $1 \leq h \leq n$. We remark that there is no state $q \in \{(q_f, p), [q_f], \langle q_f \rangle\}$ between q_i and q_{i+1} for every $i \geq 0$. Since w is accepted by A, we have that $q_h \in \{[q_f], (q_f, q)\}$ for some state $q \in Q$.
The goal is to prove that w can be written as an ω-word $v_0 v_1 v_2 \cdots v_d (v_{d+1} \cdots v_m)^\omega$ such that $v_i \in U$ for every $0 \leq i \leq m$ and $m \geq d \geq 0$. In the following, we explain why it is possible to find finite word $v \in U$ from the given run r.

   - If there is a fragment run of r from q_i to some state q_{i+1} in the form of $[q_f]$ or $([q_f], q)$ for some state q via finite word u_i, which is depicted as follows.

$$q_i \xrightarrow{u_i} [q_f] \text{ or } q_i \xrightarrow{u_i} ([q_f], q) \tag{1}$$

     In this situation, we let $v = u_i$ and according to the construction, we have that u_i is accepted by D, that is, $v \in U$.

   - If there is a fragment run of r from q_i to some state q_{i+1} in the form of $\langle q_f \rangle$ via finite word u_i, then we should find all consecutive states in the form of $\langle q_f \rangle$ behind q_{i+1}, which is depicted as follows.

$$q_i \xrightarrow{u_i} q_{i+1} = \langle q_f \rangle \xrightarrow{u_{i+1}} \cdots \xrightarrow{u_{j-1}} q_j = \langle q_f \rangle \xrightarrow{u_j} q_{j+1} \in \{[q_f], (q_f, q)\} \tag{2}$$

     In this case, we let $v = u_i \cdot u_{i+1} \cdots u_{j-1}$. It is easy to verify that $v \in U$ according to the construction.

Therefore, it follows that it is possible that we rewrite ω-word w into the specific form we mentioned above. Therefore, since A is a DBA, for any ultimately periodic word $u_0(u_1)^\omega \in \mathrm{UP}(L(A))$, we can always decompose it into the form $v_0 \cdots v_i(v_{i+1} \cdots v_n)^\omega$ such that $v_i \in U$ for every $0 \leq i \leq n$.

■

Figure 12 gives an example for the DBA A constructed from the DFA D.

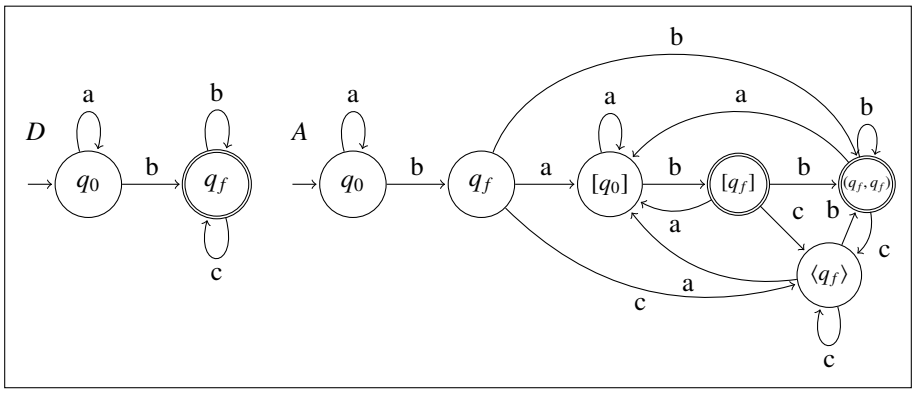


Figure 12: An example for the LDBA construction

Lemma 8 (Size of DBA). *Given a DFA $\mathcal{D}$ with a single accepting state. Let Q be the state set of $\mathcal{D}$ and A be the DBA constructed by Definition 9, then the number of states in A is in $O(|Q|^2)$.*

Proof. The proof is trivial since the state set Q' of A is defined as $Q \cup (Q \times Q) \cup \{[q] \mid q \in Q\} \cup \{\langle q \rangle \mid q \in Q\}$. ■

Corollary 1 (Sizes of LDBA). *Let $\mathcal{F}$ be an FDFA and $\underline{B}$, $\overline{B}$ be the LDBAs constructed from $\mathcal{F}$ by replacing the BAs in Definition 8 with DBAs in Definition 9. 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 $O(n^3k^5)$ and $O(n^3k^3)$, respectively.*

Proof. The sizes of $\underline{P}_{(u,v)}$ and $\overline{P}_{(u,v)}$ are in $O(nk^2)$ and $O(nk)$ respectively. Thus the sizes of DBAs recognizing $\underline{N}_{(u,v)}$ and $\overline{N}_{(u,v)}$ are in $O(n^2k^4)$ and $O(n^2k^2)$ respectively

according to Lemma 8. Moreover, the number of (u, v) pairs are at most nk. Thus we complete the proof. ■

Corollary 2 (Languages of LDBA). *Let $\mathcal{F}$ be an FDFA and $\underline{B}_d$, $\overline{B}_d$ be the LDBAs constructed from $\mathcal{F}$ by replacing the BAs in Definition 8 with DBAs in Definition 9. Let $\underline{B}$ and $\overline{B}$ be the BAs constructed from $\mathcal{F}$ by Definition 8. Then we have that $L(\underline{B}) = L(\underline{B}_d)$ and $L(\overline{B}) = L(\overline{B}_d)$.*

Proof. The proof directly follows the construction for the LDBAs. ■

8. Counterexample Analysis for FDFA Teacher

In this section, we first show how to extract valid counterexamples for the FDFA learner from the counterexamples returned from the BA teacher and give their correctness proofs in Sect. 8.1. Since the counterexample analysis makes use of three DFAs, namely $\mathcal{D}_{u\$v}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ (see Sect. 8.1), we give the automaton construction for $\mathcal{D}_{u\$v}$ in Sect. 8.2 and the automata constructions for $\mathcal{D}_1$ and $\mathcal{D}_2$ in Sect. 8.3 respectively.

8.1. Counterexample Analysis

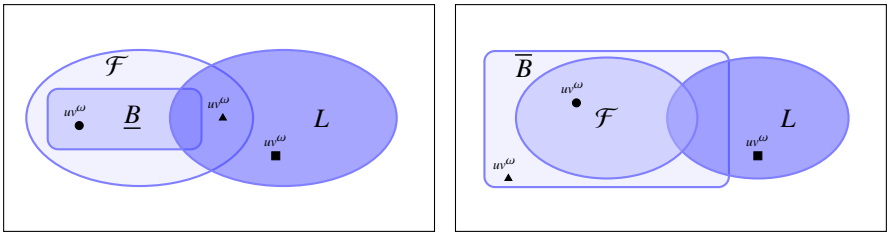
During the learning procedure, if we failed the equivalence query for the BA B, the BA teacher will return a counterexample uv^ω to the FDFA teacher.

Definition 10 (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 $uv^\omega \in \mathrm{UP}(L)$ and $uv^\omega \notin \mathrm{UP}(L(B))$,
- uv^ω is a *negative counterexample* if $uv^\omega \notin \mathrm{UP}(L)$ and $uv^\omega \in \mathrm{UP}(L(B))$.

Obviously, the above is different to the counterexample for the FDFA learner in Definition 7. 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*



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

Figure 13: The Case for Counterexample Analysis

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 FDFA teacher is *spurious* if $uv^\omega \in \text{UP}(\mathcal{F})$ (respectively $uv^\omega \notin \text{UP}(\mathcal{F})$). Suppose we use the under-approximation method to construct the BA $\underline{B}$ from $\mathcal{F}$ depicted in Fig. 10. The BA teacher returns a spurious positive counterexample b^ω, which is in $\text{UP}(\mathcal{F})$ but not in $\text{UP}(L(\underline{B}))$. 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 three DFAs below:

- 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. The constructions for the three DFAs will be introduced in Sect. 8.2 and Sect. 8.3. 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 $u'v' \backsim_M u'$.

We use different counterexample analysis procedures for the under-approximation method and the over-approximation method of FDFA to BA translation since our han-

dling with the returned counterexample has to first identify which kind of the counterexample is as in Fig. 13 and then deal with it accordingly as follows.

Given a BA $\underline{B}$ constructed by the under-approximation method to construct a BA $\underline{B}$ from $\mathcal{F}$, we have that $\mathrm{UP}(L(\underline{B})) \subseteq \mathrm{UP}(\mathcal{F})$. Figure 13(a) depicts all possible cases of $uv^\omega \in \mathrm{UP}(L(\underline{B})) \ominus \mathrm{UP}(L)$.

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') of uv^ω such that $u'v' \backsim_M u'$ and $v' \in L(A^{M(u')})$. 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 Lemma 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 $uv^\omega \in \mathrm{UP}(L(\overline{B})) \ominus \mathrm{UP}(L)$ is depicted in Fig. 13(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 Lemma 6, 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, we terminate the learning procedure and report we are not able to find suitable counterexample to refine the current FDFA.

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 $L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_1)$.

Lemma 9. *Suppose the BA teacher returns a counterexample uv^ω. For underapproximation method, we can always return a valid counterexample for the FDFA learner (u', v'). For overapproximation method, if counterexample analysis returns a decomposition (u', v'), then it is a valid counterexample for the FDFA learner.*

Proof. Recall that M is the leading automaton of the FDFA $\mathcal{F}$. Suppose the BA teacher returns a counterexample uv^ω. We prove the lemma by following cases.

- Case U1 and O1: $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \notin \mathrm{UP}(\mathcal{F})$. By Definition 7, we know that uv^ω is a positive counterexample and we need to 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 $uv^\omega \notin \mathrm{UP}(\mathcal{F})$, 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 $xy \backsim_M x$ according to [1]. 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 $\tilde{x} = M(x)$ and $xy \backsim_M x$. It follows that $x\$y \in L(\mathcal{D}_2)$ according to Proposition 5 (introduced in Sect. 8.3). 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.

- Case U3: $uv^\omega \in \mathrm{UP}(L) \wedge uv^\omega \in \mathrm{UP}(\mathcal{F})$. In this case, uv^ω is a spurious positive counterexample, which happens when we use the under-approximation method to construct the Büchi automaton. Thus we need to return a counterexample (u', v') such that $u'\$v' \in L(\mathcal{D}_{u\$v}) \cap L(\mathcal{D}_2)$. Since $uv^\omega \in \mathrm{UP}(\mathcal{F})$, then there exists some decomposition of uv^ω, say (u, v), is accepted by $\mathcal{F}$. We observe that $uv^\omega \notin \mathrm{UP}(L(\underline{B}))$, which follows that there exists some $k \geq 1$ such that (u, v^k) is not

accepted by $\mathcal{F}$ by Lemma 4. Together with $uv \backsim_M u$, we conclude that $uv^k \backsim_M u$ since M is a DFA. It follows that $u\$v^k \in L(\mathcal{D}_2)$ according to Proposition 5 (introduced in Sect. 8.3). 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.

- Case U2 and O2: $uv^\omega \notin \text{UP}(L) \wedge uv^\omega \in \text{UP}(\mathcal{F})$. 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 $uv^\omega \in \text{UP}(\mathcal{F})$, then there exists some decomposition (u', v') of uv^ω is accepted by $\mathcal{F}$. It follows that $u'\$v' \in L(\mathcal{D}_1)$ according to Proposition 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.

- Case O3: $uv^\omega \notin \text{UP}(L) \wedge uv^\omega \notin \text{UP}(\mathcal{F})$. 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 Lemma 6, 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 $uv_i \backsim_M u$. If we find some $i \geq 1$ such that $uv_i^\omega \notin \text{UP}(L)$, then we let $u' = u$ and $v' = v_i$. Clearly, (u', v') is a negative counterexample for FDFA learner. ■

8.2. From ω-word uv^ω to DFA $\mathcal{D}_{u\$v}$

In [32], they presented a canonical representation $L_\$ = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$ for an ω-regular language L. In principle, we can apply their method to obtain the $\mathcal{D}_{u\$v}$ automaton from an ω-word uv^ω. However, the number of states in their constructed DFA is in $O(2^{|u|+|v|})$. In this section, we present a more effective method to build the DFA $\mathcal{D}_{u\$v}$ such that $L(\mathcal{D}_{u\$v}) = \{u'\$v' \mid u' \in \Sigma^*, v' \in \Sigma^+, u'v'^\omega = uv^\omega\}$ for a given ω-word uv^ω and the number of states in $\mathcal{D}_{u\$v}$ is in $O(|v|(|v| + |u|))$. A similar construction for $\mathcal{D}_{u\$v}$ has been proposed in [31], which first computes the regular expression to represent all possible decompositions of uv^ω and then constructs a DFA

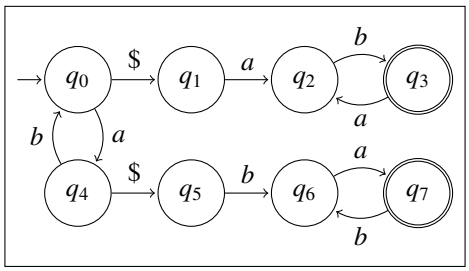


Figure 14: $\mathcal{D}_{u\$v}$ for ω-word (aba, ba)

from the regular expression. Compared to the construction in [31], ours is a direct construction from an ω-word uv^ω to DFA $\mathcal{D}_{u\$v}$ and we also give the complexity of the construction.

We first give an example automaton $\mathcal{D}_{u\$v}$ for ω-word $(ab)^\omega$ in Fig. 14. From the example, we can find that both decompositions (aba, ba) and $(ababa, bababa)$ have the same suffix $(ba)^\omega$, which gives us the hint that the second element of a decomposition can be simplified as long as we do not change the periodic word.

In the following, we denote by $u \trianglelefteq v$ to represent that u is a prefix of v, i.e., there exists some $j \geq 1$ such that $u = v[1 \cdots j]$. We use $u \triangleleft v$ if $u \trianglelefteq v$ and $u \neq v$. We give the definition of a *smallest period* in an ω-word w given by (u, v) where $v \in \Sigma^+$.

Definition 11 (Smallest period). For any ω-word w given by (u, v), we say r is the smallest period of (u, v) if $r \trianglelefteq v$, $r^\omega = v^\omega$ and for any $t \triangleleft r$, $t^\omega \neq r^\omega$.

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 shifting the first letter of ab to its tail. We prove that in Lemma 10, given an ω-word w, the length of its smallest period is fixed no matter what decomposition of w is given.

Lemma 10. *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 we have that either there exists $j \geq 2$ such that $r = t[j \cdots n] \cdot t[1 \cdots j-1]$ or $r = t$ where $|t| = n$.*

Proof. According to Definition 11, $w = uv^\omega = ur^\omega = xy^\omega = xt^\omega$. We prove it by

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

contradiction. Without loss of generality, we assume that $|r| > |t|$. If $|u| = |x|$, we conclude that $r^\omega = t^\omega$, which follows that r is not a smallest period of (u, v) since $t \lhd r$. Therefore $|r| > |t|$ cannot hold in this case. Otherwise if $|u| \neq |x|$, we can either prove that $r = t$ or find some $j \geq 2$ such that $z = t[j \cdots n] \cdot t[1 \cdots j-1] \lhd r$ and $z^\omega = r^\omega$ in following cases. Recall that $\oplus$ is the standard modular arithmetic operator.

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

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

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

Lemma 10 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 11. *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 $x' \trianglelefteq u$ 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 Definition 11 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 $u = x[1 \cdots k]$ and repeat the procedure.

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 3. *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 Corollary 3, 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.

8.2.1. The algorithm to construct $\mathcal{D}_{u\$v}$

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 Lemma 10. 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 $A = (\Sigma, \{q_0, \cdots, q_{2n}\}, q_0, \{q_{2n}\}, \delta)$ 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 $A = (\Sigma, \{q_0, \cdots, q_{2n+m}\}, q_0, \{q_{m+2n}\}, \delta)$ 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 $u'\$v' \in L(D_i)$, 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 Lemma 11. Suppose (x, y) is the first shortest form used in the $\mathcal{D}_{u\$v}$ construction. By Lemma 10, 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 Corollary 3, which follows that $u'\$v' \in L(D_j) \subseteq L(\mathcal{D}_{u\$v})$.

Therefore, we complete the proof. ∎

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 $O(|r| \times (|r| + |u|)) \in O(|v|(|u| + |v|)$.

8.3. From FDFA $\mathcal{F}$ to DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$

In this section, we provide the constructions for the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$ from a given FDFA $\mathcal{F} = (M, \{A^u\})$. For a given state u in the leading DFA M, we define a DFA $\overline{A^u} = (\Sigma, Q^u, s^u, Q^u \setminus F^u, \delta^u)$ from its corresponding progress automaton $A^u = (\Sigma, Q^u, s^u, F^u, \delta^u)$ 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 $s \in Q^u, a \in \Sigma$. To make the constructions simple, we define two automata N_u and $\overline{N_u}$ for every state u in the leading automaton M as follows.

- For $\mathcal{D}_1$ construction, we define $N_u = M^u_u \times A^u$. Intuitively, we only keep the words from $L(A^u)$ which can make a run start at u and go back to u in the leading automaton. In other words, $L(N_u) = \{v \in \Sigma^* \mid uv \frown_M u, v \in L(A^u)\}$.
- For $\mathcal{D}_2$ construction, we define $\overline{N_u} = M^u_u \times \overline{A^u}$. Similarly, we have $L(\overline{N_u}) = \{v \in \Sigma^* \mid uv \frown_M u, v \notin L(A^u)\}$.

Recall that M^u_u is obtained from M by setting the initial and final state to u. Formally, the construction is defined as follows.

Definition 12. Let $\mathcal{F} = \{M, \{A^u\}\}$ be an FDFA where $M = (\Sigma, Q, q_0, \delta)$ and for every $u \in Q$, the corresponding progress automaton $A^u = (\Sigma, Q^u, s^u, F^u, \delta^u)$. Let N_u (respectively $\overline{N_u}$) be given by $(\Sigma, Q_u, s_u, F_u, \delta_u)$. The DFA $\mathcal{D}_1$ (respectively $\mathcal{D}_2$) is defined as

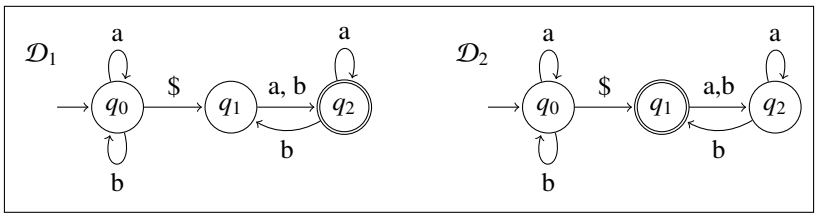


Figure 15: $\mathcal{D}_1$ and $\mathcal{D}_2$ for $\mathcal{F}$ in Fig. 1

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.

Intuitively, we use the \$ transitions to connect the leading DFA M and the DFAs N_u (respectively $\overline{N_u}$). In Fig. 15, we give the DFAs $\mathcal{D}_1$ and $\mathcal{D}_2$ constructed from $\mathcal{F}$ in Fig. 1.

Proposition 4. *Given an FDFA $\mathcal{F} = (M, \{A^u\})$, $\mathcal{D}_1$ is the DFA defined by Definition 12 from $\mathcal{F}$. Then $L(\mathcal{D}_1) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, \tilde{u} = M(u), v \in L(A^{\tilde{u}})\}$.*

Proof. According to Definition 12, it is easy to show that for any $u \in \Sigma^*$, $\tilde{u} = M(u) = \mathcal{D}_1(u)$. Moreover, 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. In Sect. 2, we defined that (u, v) is accepted by $\mathcal{F}$ iff we have $uv \backsim_M u$ and $v \in L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. Therefore we only need to prove that $u\$v$ is accepted by $\mathcal{D}_1$ iff (u, v) is accepted by $\mathcal{F}$.

- $\subseteq$. We prove that if (u, v) is accepted by $\mathcal{F}$, then $u\$v \in L(\mathcal{D}_1)$. Since (u, v) is accepted by $\mathcal{F}$, that is, $uv \backsim_M u$ and $v \in L(A^{\tilde{u}})$, we have that $v \in L(N_{\tilde{u}})$. It 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, $u\$v \in L(\mathcal{D}_1)$.

- $\supseteq$. First, we have that $L(\mathcal{D}_1) \subseteq \Sigma^*\$\Sigma^*$ by Definition 12. 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 $uv \backsim_M u$ and $v \in L(A^{\tilde{u}})$. Thus, we conclude that (u, v) is accepted by $\mathcal{F}$.

■

Proposition 5. *Given an FDFA $\mathcal{F} = (M, \{A^u\})$, $\mathcal{D}_2$ is the DFA defined by Definition 12 from $\mathcal{F}$. Then $L(\mathcal{D}_2) = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^*, uv \backsim_M u, \tilde{u} = M(u), v \notin L(A^{\tilde{u}})\}$.*

Proof. Similar to the construction of $\mathcal{D}_1$, we have that for any $u \in \Sigma^*$, $\tilde{u} = M(u) = \mathcal{D}_2(u)$. For any $u, v \in \Sigma^*$, we have that $\overline{N}_{\tilde{u}}(v) = \mathcal{D}_2(u\$v)$ where $\tilde{u} = M(u)$ since $\mathcal{D}_2$ is a DFA.

- $\supseteq$. Assume that we have $uv \backsim_M u$ and $v \notin L(A^{\tilde{u}})$ where $\tilde{u} = M(u)$. By $uv \backsim_M u$, we have that $v \in L(M_{\tilde{u}}^{\tilde{u}})$. Further, we conclude that $v \in L(\overline{A^{\tilde{u}}})$ from the fact that $v \notin L(A^{\tilde{u}})$. 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 also an accepting state. Therefore, $u\$v \in L(\mathcal{D}_2)$.

- $\subseteq$. First, we have that $L(\mathcal{D}_2) \subseteq \Sigma^*\$\Sigma^*$ by Definition 12. 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 $uv \backsim_M u$ and $v \notin L(A^{\tilde{u}})$.

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

Proof. Suppose n is the number of states in M and k is the number of states in the largest progress automaton, then the numbers of states of $\mathcal{D}_1$ and $\mathcal{D}_2$ are both in $O(n + n^2k)$ according to Definition 12. ■

9. Correctness and Complexity Analysis

In this section, we first discuss the correctness of the tree-based FDFA learning algorithms in Sect 9.1 and then present the complexity in Sect. 9.2. Together with the correctness of BA constructions and counterexample analysis, it leads to our main result Theorem 4 in Sect. 9.2.

9.1. Correctness of Tree-based FDFA Learning Algorithm

In the following, we fix an FDFA structure $\mathcal{F} = (M, \{A^u\})$ and its progress tree $(\mathcal{T}, \{\mathcal{T}_u\})$. Lemma 12 establishes the correctness of our tree-based learning algorithm for the periodic progress trees and the leading trees.

Lemma 12. *Given an ω-regular language L, the tree-based learning algorithm will never classify two finite words which belong to the same equivalence class into two different terminal nodes in the leading tree and the periodic progress trees.*

Proof. We prove by contradiction. Suppose there are two finite word $x_1, x_2 \in \Sigma^*$ which are actually 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 first have the premise $x_1 \backsim_L x_2$. Suppose x_1 and x_2 have currently been assigned to terminal nodes t_1 and t_2 and $t_1 \neq t_2$. 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 $\mathbf{TE}(x_1, (y, v)) = \text{F}$ and $\mathbf{TE}(x_2, (y, v)) = \text{T}$. It follows that $x_1(yv)^\omega \notin \text{UP}(L)$ and $x_2(yv)^\omega \in \text{UP}(L)$, which implies that $x_1 \not\backsim_L x_2$. Contradiction.

- $\mathcal{T} = \mathcal{T}_u$ is a progress tree in periodic FDFA. We have the premise $x_1 \approx_P^u x_2$. Similarly, we assume that x_1 and x_2 have been assigned to terminal nodes t_1 and t_2 of $\mathcal{T}_u$ and $t_1 \neq t_2$. 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 $\mathbf{TE}(x_1, v) = \text{F}$ and $\mathbf{TE}(x_2, v) = \text{T}$. It follows that $u(x_1v)^\omega \notin \text{UP}(L)$ and $u(x_2v)^\omega \in \text{UP}(L)$, which implies that $x_1 \not\approx_P^u x_2$. Contradiction.

■

Recall that in Definition 4 and 5, $x \approx_S^u y$ and $x \approx_R^u y$ are defined by using the right congruence $\backsim_L$. In other words, the progress trees in the syntactic and recurrent FDFAs

are constructed according to current leading automaton M. Therefore, the progress trees can be correctly constructed if current leading automaton M is already *consistent* with $\backsim_L$. We say a leading automaton M is consistent with $\backsim_L$ iff for any $x_1, x_2 \in \Sigma^*$, we have $M(x_1) = M(x_2) \iff x_1 \backsim_L x_2$. We prove the correctness of tree-based learning algorithms for the syntactic and the recurrent FDFAs by Lemma 13.

Lemma 13. *Given an ω-regular language L, the tree-based algorithm will never classify two finite words which belong to the same equivalence class into two different terminal nodes in the progress trees of the syntactic and the recurrent FDFA if the leading automaton M is consistent with $\backsim_L$.*

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

Proof. Note that the progress trees $\mathcal{T}_u$ in syntactic and recurrent FDFAs are constructed with respect to the current leading automaton M where u is a state in M. We prove the lemma in following cases by contradiction.

- $\mathcal{T}_u$ is a progress tree in syntactic FDFA. We first have the premise $x_1 \approx_S^u x_2$. Suppose x_1 and x_2 have been assigned to different terminal nodes 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 the syntactic FDFA defined in Sect. 6.2, we let $d_1 := \mathbf{TE}(x_1, v) = (M(ux_1), m_1)$ 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 $m_1 \neq m_2$. In order to prove the lemma, we have to prove $d_1 \neq d_2$ does not hold in the following.

  If M is consistent with $\backsim_L$, we prove $d_1 = d_2$ in the following cases.

  - $M(ux_1) \neq M(ux_2)$. Since $x_1 \approx_S^u x_2$, we have $ux_1 \backsim_L ux_2$ according to the definition of $\approx_S^u$ in Definition 4. It immediately follows that $M(ux_1) = M(ux_2)$ since M is consistent with $\backsim_L$. Contradiction.

– $m_1 \neq m_2$. Similarly, since $x_1 \approx_S^u x_2$, we have $ux_1 \backsim_L ux_2$. It follows that $M(ux_1) = M(ux_2)$ since M is consistent with $\backsim_L$. Moreover, we have that $M(ux_1v) = M(ux_2v)$ for any $v \in \Sigma^*$ since M is deterministic. We discuss the equality of m_1 and m_2 for some $v \in \Sigma^*$ in the following two cases.

  * $u = M(ux_1v)$. It follows that $ux_1v \backsim_L u$ since M is consistent with $\backsim_L$, which immediately implies that $u(x_1v)^\omega \in \mathrm{UP}(L) \iff u(x_2v)^\omega \in \mathrm{UP}(L)$ according the definition of $x_1 \approx_S^u x_2$. Moreover, we have $u = M(ux_2v)$ since $ux_1 \backsim_L ux_2$. Therefore, we conclude that $m_1, m_2 \in \{A, B\}$ by the definition of **TE** in Sect. 6.2. Without loss of generality, we let $m_1 = A$ and $m_2 = B$, which implies that $u(x_1v)^\omega \in \mathrm{UP}(L)$ while $u(x_2v)^\omega \notin \mathrm{UP}(L)$. Contradiction.
  * $u \neq M(ux_1v)$. According to the definition of **TE** in Sect. 6.2, we have $m_1 = m_2 = C$, which follows that $d_1 = d_n$ since $M(ux_1) = M(ux_2)$. Contradiction.

  Therefore, t_1 and t_2 cannot be different terminal nodes if M is consistent with $\backsim_L$.

- $\mathcal{T}_u$ is a progress tree in recurrent FDFA. The analysis is similar to the analysis for the syntactic FDFA. We first have premise $x_1 \approx_R^u x_2$. Suppose x_1 and x_2 have been assigned to different terminal nodes 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 $d_1, d_2 \in \{\mathrm{F}, \mathrm{T}\}$. Since t_1 and t_2 are in different subtrees of n, we thus have $d_1 \neq d_2$. Without loss of generality, we let $d_1 = \mathrm{F}$ and $d_2 = \mathrm{T}$. We first assume that M is consistent with $\backsim_L$. Since $d_2 = \mathrm{T}$, we have that $u = M(ux_2v)$ and $u(x_2v)^\omega \in \mathrm{UP}(L)$ according to the definition of **TE** in Sect. 6.2. It follows that $ux_2v \backsim_L u$ since M is consistent with $\backsim_L$. Moreover, we conclude that $u = M(ux_1v)$ and $u(x_1v)^\omega \in \mathrm{UP}(L)$ by the fact that $x_1 \approx_R^u x_2$. By the definition of **TE**, we have $d_1 = \mathrm{T}$. Contradiction. Therefore, t_1 and t_2 cannot be different terminal nodes.

From above analysis, we can also conclude that if the classification of equivalence classes in the progress trees are not correct, then M is not consistent with $\backsim_L$. ■

We have already proved that the tree-based algorithm will not do any "bad" things, that is, it will not classify two different words which actually belong to the same equivalence class into different terminal nodes. In the following, we show that the tree-based algorithm will do "good" things, namely making progress for learning the unknown ω-regular language L.

Here we recall Lemma 3, which states that there will be a new state added to the leading automaton M or the corresponding progress automaton $A^{\tilde{u}}$ after each refinement step. Lemma 3 is a critical property for the termination of the tree-based FDFA learning algorithm since it means that we either make progress for the leading automaton or the corresponding progress automaton after every refinement.

In Lemma 13, we encounter a situation where the progress trees of the syntactic and the recurrent FDFAs may classify two finite words which actually are in the same equivalence class into different terminal nodes if M is not consistent with $\backsim_L$. One may worry that if the FDFA teacher chooses to refine the progress automaton continually, the learning algorithm may not terminate. Lemma 14 shows that it will terminate since the number of equivalence classes of the progress automata with respect to current leading automaton M is finite. More precisely, if we fix the leading automaton M, we are actually learning a DFA induced by the right congruence $\approx^u_{S'}$. We define $x \approx^u_{S'} y$ if and only if $M(ux) = M(uy)$ and for every $v \in \Sigma^*$, if $M(uxv) = u$, then $u(xv)^\omega \in L \iff u(xv)^\omega \in L$. One can easily verify that $x \approx^u_{S'} y$ is a right congruence. We remark that if M is consistent with $\backsim_L$, then $x \approx^u_{S'} y$ is equivalent to $x \approx^u_S y$.

Lemma 14. *Given the leading automaton M, then for every state u in M, the index of $\approx^u_{S'}$ is bounded by $|Q| \cdot |\approx^u_P|$ where Q is the state set of M.*

Proof. We use q_i to denote the state which can be reached by u where $1 \leq i \leq |Q|$. Given a finite word $x \in \Sigma^*$, we classify x into an equivalence class of $\approx^u_{S'}$ as follows. According to the definition of $\approx^u_{S'}$, we can first find state $q_i = M(ux)$ for some $1 \leq i \leq |Q|$. For every $y \in \Sigma^*$ such that $q_i \neq M(uy)$, x and y should not be in the same equivalence class. Therefore, x can be first be classified into one of $|Q|$ classes since the number of possible q_i is $|Q|$. Now we fix the q_i, in order to distinguish x with finite word y such that $q_i = M(uy)$, we have to check whether for $\forall v \in \Sigma^*$, if $M(uxv) = u$,

then $u(xv)^\omega \in L \iff u(yv)^\omega \in L$, which implies that whether $x \approx_P^u y$ holds. Thus we can use $(q_i, [x]_{\approx_P^u})$ to represent the equivalence class where x belongs to. Therefore, the index of the right congruence $\approx_{S'}^u$ is $|Q| \cdot | \approx_P^u |$. ■

Similarly, if we fix the leading automaton M and learn recurrent FDFA, we are actually learning DFA induced by the right congruence $\approx_{R'}^u$ defined as that $x \approx_{R'}^u y$ 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 first assuming $x \approx_{S'}^u y$ and then conclude that for any $v \in \Sigma^*$, we have that $uyv \backsim_M u \wedge u(yv)^\omega \in L$ if $uxv \backsim_M u \wedge u(xv)^\omega \in L$. First, assume that $uxv \backsim_M u \wedge u(xv)^\omega \in L$ and $x \approx_{S'}^u y$, one can easily conclude that $u(yv)^\omega \in L$. In addition, one can combine the result $ux \backsim_M uy$ from $x \approx_{S'}^u y$ and assumption $uxv \backsim_M u$ together to prove $uyv \backsim_M u$ since M is deterministic and $\backsim_M$ is an equivalence relation.

Lemma 15. *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.*

Theorem 3. *Given the FDFA teacher that is able to answer membership and equivalence queries about ω-regular language L for FDFA, the tree-based FDFA learning algorithm will terminate and can learn the three canonical FDFAs.*

Proof (Sketch). We prove the theorem by following cases.

- For the periodic FDFAs, together with Lemma 12 and Lemma 3, the tree-based algorithm will terminate and learn the corresponding periodic FDFA $\mathcal{F}$ of L since the number of states in $\mathcal{F}$ is finite.
- For the syntactic and the recurrent FDFAs, with Lemma 12 and Lemma 13, we conclude that the tree-based algorithm can classify the finite words correctly if the leading automaton M is consistent with $\backsim_L$. If M is not consistent with $\backsim_L$, the FDFA teacher will be able to return a counterexample to refine current M. If the leading automaton changes, the algorithm for the syntactic and the recurrent FDFAs should learn all progress automata from scratch with respect to current

leading automaton M and the learning procedure for progress automata will terminate which is justified by Lemma 3, Lemma 14 and Lemma 15. At some point, the leading automaton M will be consistent with $\backsim_L$ since its states number increases after every refinement. Thus, the learning algorithm will terminate since the numbers of states in the corresponding syntactic FDFAs and recurrent FDFAs of L are finite.

Therefore, we complete the proof. ■

9.2. Complexity for Tree-based Learning Algorithm

In the following, we denote by $\mathcal{F} = (M, \{A^u\})$ the corresponding periodic FDFA of the unknown ω-regular language L. In this section, we 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 unless stated otherwise. 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 $|u| + |v|$.

Proposition 7. *Suppose the FDFA learner poses an equivalence query for $\mathcal{F} = (M, \{A^u\})$. The number of states in M is n and the number of states in the largest progress automaton A^u is k. In the FDFA teacher, suppose the counterexample uv^ω returned by the BA teacher is given by 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. • According to Lemma 5, it is immediate that the time and space complexity for building $\underline{B}$ (respectively $\overline{B}$) is in $O(n^2k^3)$ (respectively, $O(n^2k^2)$).

- From Proposition 3 and 6, 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 $|v|(|v| + |u|)$ since (u, v) is the returned decomposition of the counterexample uv^ω. Note that except for case O3 in the over approximation construction method, $\mathcal{D}_{u\$v}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ are used in counterexample analysis. When we analyze the spurious negative counterexample, i.e., case O3, 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 Lemma 6. Since the time and space complexity for case O1 and O3 are both in $O(n^2k \cdot (|v|(|v| + |u|))$, we conclude that the time and space complexity for the overapproximation method are in $O(n^2k^2 \cdot (|v|(|v| + |u|))$ and in $O(n^2k(|v|(|v| + |u|))$ respectively.

Therefore, we complete the proof. ∎

Theorem 4 (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 $O((n + n^3k) \cdot (|u| + |v| + (n + nk) \cdot |\Sigma|))$.

Proof. Theorem 4 can be concluded from Lemma 12, Lemma 14, Lemma 15 and Theorem 3. Suppose $\mathcal{F} = (M, \{A^u\})$ is the corresponding periodic FDFA 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 $|u|$ when we refine the leading automaton and is at most $|v|$ when we refine the progress automaton. Therefore, the number of membership queries used in analyzing counterexample is bounded by $|u| + |v|$. We remark that one can also use binary search to reduce the number of membership queries used by counterexample analysis to $\log(|u| + |v|)$. The membership queries are also needed in constructing the corresponding automata M or $A^{M(u)}$ after the classification tree has been refined. Suppose the new added terminal node is labeled by p, the terminal node which needs to 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 automata 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 $|u| + |v| + m + |\Sigma| \cdot m + |\Sigma|$, where $|u| + |v|$ membership queries are used for analyzing the counterexample and $m + (m + 1)|\Sigma|$ membership queries are used to fill the table.

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

- For periodic FDFA. During the learning procedure, when receiving a counterexample for FDFA 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 FDFA is at most $n + nk$. In periodic FDFA, 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 $(n+nk)\cdot(|u|+|v|+2(n+k)\cdot|\Sigma|) \in \mathcal{O}((n+nk)\cdot(|u|+|v|+(n+k)\cdot|\Sigma|))$ in the worst case.

- For syntactic and recurrent FDFAs, when receiving a counterexample for FDFA 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 labeled by ϵ again, so the number of states in the progress automata of the FDFA may decrease at that point, otherwise it refines the progress automaton and the number of states in FDFA will be increased 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 Lemma 14 and Lemma 15. When all progress trees cannot be refined any more, either the learning task finishes or the FDFA teacher returns a counterexample to refine the current leading automaton. For the latter case, the number of states in the leading automaton will be increased 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 FDFA is bounded by $(m + 1)^2 \cdot k$. Therefore, the number of equivalence queries needed for tree-based algorithm is bounded by $(1 + 1 \cdot 1 \cdot k) + (1+2\cdot 2\cdot k)+\cdots(1+(n-1)\cdot(n-1)\cdot k)+(1+n\cdot n\cdot k) \in O(n+n^3k)$. 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.

■

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

Theorem 5 (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 $O((nk + nk\cdot\ |\Sigma|)\cdot nk)$.

PROOF. As we mentioned in Sect. 5, the FDFA 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 FDFA, the number of states in the FDFA 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 the recurrent FDFA. The learning procedure for the leading automaton is the same as the one of the periodic FDFA. Thus the space required by table-based and tree-based algorithm remain the same.

  For learning progress automata, the number of states in each progress automaton is at most nk according to Lemma 14 and Lemma 15. 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 $O(nk)$.

■

Theorem 6 (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 a Büchi automaton in the worst case. If the BA learning algorithm based on the over-approximation method terminates not because that it cannot find valid counterexamples for the FDFA learner when dealing with the counterexamples in case O3, the output BA B must recognize L since B has passed the equivalence query. This theorem is justified by Lemma 2, Lemma 5 and Theorem 3. ■

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.

10. 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 [46, 47] 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 Table 1. In this section, we use $L^{\$}$ to denote the ω-regular learning algorithm in [31], and L^{Periodic}, $L^{\mathsf{Syntactic}}$, and $L^{\mathsf{Recurrent}}$ to represent the periodic, syntactic, and recurrent FDFA learning algorithm introduced in Sect. 5 and 6. 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 counterparts. (2) L^{Periodic} uses fewer membership queries comparing to $L^{\mathsf{Syntactic}}$ and $L^{\mathsf{Recurrent}}$. The

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. EQ(%) is the percentage of the time for the equivalence queries in the total running 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

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.

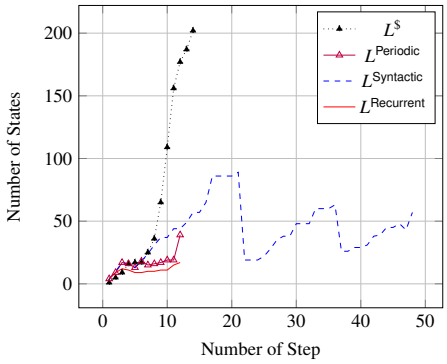


Figure 16: Growth of state counts in BA

In the experiment, we observe that table-based $L^\$$ has 4 cases cannot be finished within the timeout period, which is the largest number among all learning algorithms[2]. 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

[2]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.

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. 16, 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 cannot 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 at those iterations and thus the algorithms have to redo the learning of all progress automata from scratch.

To our surprise, in our experiment, the size of BAs $\overline{B}$ produced by the overapproximation method is not much smaller than the BAs $\underline{B}$ produced by the underapproximation 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 $M_u^u \times (A^u)_v^{s_u}$ (Sect. 7). 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 Sect. 6.2, 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 9 more cases that were not solved before.

Given an FDFA $\mathcal{F}$, the number of states of LDBA constructed by Definition 9 is quadratically larger than that in the corresponding NBA constructed by Definition 8. Therefore, in order to learn a LDBA we can construct NBAs for the equivalence queries and construct a LDBA from the FDFA for final result once the learning task succeeds. We use this strategy to learn the LDBAs from the BAs of the BüchiStore. Since the learning procedure is the same as usual except we construct a LDBA from the final FDFA, we only compare the number of states between NBAs and LDBAs learned from the tree-based algorithms in Fig. 17. The comparison of number of states be-

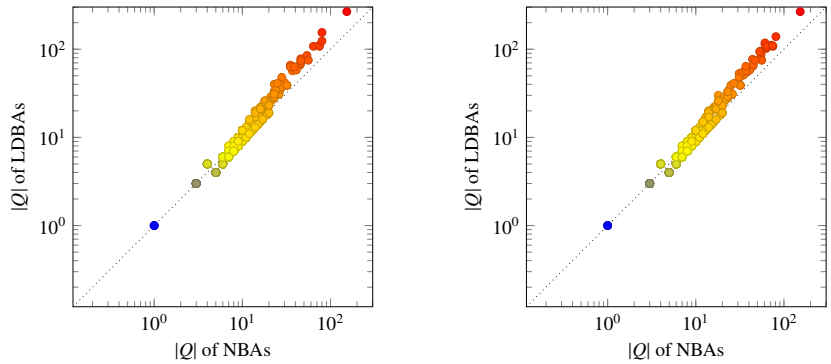


Figure 17: The number of states comparison between NBAs and LDBAs learned from tree-based algorithms

tween NBAs and LDBAs learned from table-based algorithms share the same trend and thus are omitted. From Fig. 17, we can see that most of points are above the diagonal which indicates the number of states in LDBAs are larger than those in NBAs when we consider large BAs. This is because the number of states of LDBAs are quadratically larger than those of NBAs. We also observe that there are a lot of points which are below the diagonal. The reason is that the construction of LDBA in Definition 9 requires first remove all states which cannot reach the accepting states in the DFA, while construction of BAs in Definition 8 does not use the same removal operation.

11. 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 algorithms is significantly better than the table-based ones 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 are 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 [48] 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 testing.

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.

There are learning algorithms for residual NFA [2], 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.

Acknowledgement. This work has been supported 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] D. Angluin, D. Fisman, Learning regular omega languages, Theor. Comput. Sci. 650 (2016) 57–72.

[2] B. Bollig, P. Habermehl, C. Kern, M. Leucker, Angluin-style Learning of NFA, in: IJCAI, 2009, pp. 1004–1009.

[3] M. J. Kearns, U. V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, USA, 1994.

[4] D. Angluin, Learning Regular Sets from Queries and Counterexamples, Inf. Comput. 75 (2) (1987) 87–106.

[5] R. L. Rivest, R. E. Schapire, Inference of Finite Automata Using Homing Sequences, in: STOC, 1989, pp. 411–420.

[6] J. M. Cobleigh, D. Giannakopoulou, C. S. Păsăreanu, Learning assumptions for compositional verification, in: TACAS, 2003, pp. 331–346.

[7] S. Chaki, E. Clarke, N. Sinha, P. Thati, Automated assume-guarantee reasoning for simulation conformance, in: CAV, 2005, pp. 534–547.

[8] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, B.-Y. Wang, Learning minimal separating DFA's for compositional verification, in: TACAS, 2009, pp. 31–45.

[9] O. Grumberg, Y. Meller, Learning-Based Compositional Model Checking of Behavioral UML Systems, Dependable Software Systems Engineering 45 (2016) 117.

[10] S.-W. Lin, E. André, Y. Liu, J. Sun, J. S. Dong, Learning assumptions for compositional verification of timed systems, IEEE Transactions on Software Engineering 40 (2) (2014) 137–153.

[11] R. Alur, P. Madhusudan, W. Nam, Symbolic compositional verification by learning assumptions, in: CAV, 2005, pp. 548–562.

[12] L. Feng, M. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: ICSE, 2011, pp. 2–17.

[13] F. He, X. Gao, B. Wang, L. Zhang, Leveraging Weighted Automata in Compositional Reasoning about Concurrent Probabilistic Systems, in: POPL, 2015, pp. 503–514.

[14] D. Peled, M. Y. Vardi, M. Yannakakis, Black box checking, Journal of Automata, Languages and Combinatorics 7 (2) (2002) 225–246.

[15] A. Hagerer, H. Hungar, O. Niese, B. Steffen, Model generation by moderated regular extrapolation, in: FASE, 2002, pp. 80–95.

[16] F. Wang, J. Wu, C. Huang, K. Chang, Evolving a Test Oracle in Black-Box Testing, in: FASE, 2011, pp. 310–325.

[17] R. Alur, P. Černỳ, P. Madhusudan, W. Nam, Synthesis of interface specifications for Java classes, in: POPL, 2005, pp. 98–109.

[18] F. Howar, D. Giannakopoulou, Z. Rakamarić, Hybrid learning: interface generation through static, dynamic, and symbolic analysis, in: ISSTA, 2013, pp. 268–279.

[19] D. Giannakopoulou, Z. Rakamarić, V. Raman, Symbolic learning of component interfaces, in: SAS, 2012, pp. 248–264.

[20] J. Sun, H. Xiao, Y. Liu, S.-W. Lin, S. Qin, TLV: abstraction through testing, learning, and validation, in: FSE, 2015, pp. 698–709.

[21] F. Aarts, B. Jonsson, J. Uijen, F. Vaandrager, Generating models of infinite-state communication protocols using regular inference with abstraction, Formal Methods in System Design 46 (1) (2015) 1–41.

[22] M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, M. Tautschnig, Learning the Language of Error, in: ATVA, 2015, pp. 114–130.

[23] Y.-F. Chen, C. Hsieh, O. Lengál, T.-J. Lii, M.-H. Tsai, B.-Y. Wang, F. Wang, PAC Learning-based Verification and Model Synthesis, in: ICSE, 2016, pp. 714–724.

[24] H. Xiao, J. Sun, Y. Liu, S.-W. Lin, C. Sun, Tzuyu: Learning stateful typestates, in: ASE, 2013, pp. 432–442.

[25] F. Vaandrager, Model Learning, Communications of the ACM 60 (2) (2017) 86–95.

[26] F. Howar, B. Steffen, B. Jonsson, S. Cassel, Inferring Canonical Register Automata, in: VMCAI, 2012, pp. 251–266.

[27] M. Isberner, F. Howar, B. Steffen, Learning register automata: from languages to program structures, Machine Learning 96 (1-2) (2014) 65–98.

[28] B. Alpern, F. B. Schneider, Recognizing safety and liveness, Distributed computing 2 (3) (1987) 117–126.

[29] M. Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: LICS, 1986, pp. 322–331.

[30] C. S. Lee, N. D. Jones, A. M. Ben-Amram, The size-change principle for program termination, in: POPL, 2001, pp. 81–92.

[31] A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, B.-Y. Wang, Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages, in: TACAS, 2008, pp. 2–17.

[32] H. Calbrix, M. Nivat, A. Podelski, Ultimately Periodic Words of Rational ω-Languages, in: MFPS, 1993, pp. 554–566.

[33] O. Maler, L. Staiger, On Syntactic Congruences for Omega-Languages, in: STACS, 1993, pp. 586–594.

[34] M. Heizmann, J. Hoenicke, A. Podelski, Termination analysis by learning terminating programs, in: CAV, Springer-Verlag New York, Inc., New York, NY, USA, 2014, pp. 797–813.

[35] C. Courcoubetis, M. Yannakakis, The complexity of probabilistic verification, J. ACM 42 (4) (1995) 857–907.

[36] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, D. R. Piegdon, libalf: The Automata Learning Framework, in: CAV, 2010, pp. 360–364.

[37] M. Isberner, F. Howar, B. Steffen, The open-source LearnLib, in: CAV, 2015, pp. 487–495.

[38] Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, Y.-W. Chang, Büchi Store: An Open Repository of Büchi Automata, in: TACAS, 2011, pp. 262–266.

[39] Y. Li, Y. Chen, L. Zhang, D. Liu, A Novel Learning Algorithm for Büchi Automata based on Family of DFAs and Classification Trees, in: TACAS, 2017, pp. 208–226.

[40] S. Sickert, J. Esparza, S. Jaax, J. Křetínský, Limit-Deterministic Büchi Automata for Linear Temporal Logic , in: CAV, 2016, pp. 312–332.

[41] F. Blahoudek, M. Heizmann, S. Schewe, J. Strejček, M.-H. Tsai, Complementing semi-deterministic büchi automata, in: TACAS, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 770–787.

[42] J. R. Büchi, On a decision method in restricted second order arithmetic, in: Int. Congress on Logic, Methodology and Philosophy of Science, 1962, pp. 1–11.

[43] A. Arnold, A syntactic congruence for rational ω-languages, Theoretical Computer Science 39 (1985) 333–335.

[44] O. Maler, L. Staiger, On syntactic congruences for omega-languages, in: STACS, Vol. 665 of LNCS, 1993, pp. 586–594.

[45] D. Angluin, U. Boker, D. Fisman, Families of DFAs as Acceptors of omega-Regular Languages, in: MFCS, 2016, pp. 11:1–11:14.

[46] P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, T. Vojnar, Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing, in: CAV, 2010, pp. 132–147.

[47] P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, T. Vojnar, Advanced Ramsey-Based Büchi Automata Inclusion Testing, in: CONCUR, 2011, pp. 187–202.

[48] D. Angluin, Queries and Concept Learning, Mach. Learn. 2 (4) (1988) 319–342.

Appendix

In this section, we show that although our acceptance condition defined in Sect. 2 is different from the original one defined in [1], but the ultimately periodic words of the FDFA will be preserved.

Recall that the original acceptance condition for periodic FDFA in [1] 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 FDFA in [1] 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 FDFA $\mathcal{F}$ is defined as $\mathrm{UP}(\mathcal{F}) = \{uv^\omega \mid (u, v)$ is accepted by $\mathcal{F}\}$. The acceptance condition for periodic FDFA used in this paper is different from the original one in [1]. We prove that the acceptance condition does not change the ultimately periodic words of the periodic FDFAs.

Lemma 16. *Let $\mathcal{F}$ be a periodic (syntactic, recurrent) FDFA under the acceptance condition in [1], 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 FDFA $\mathcal{F}$, the original acceptance condition of periodic FDFA 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 FDFA. 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 $xy \backsim_M x$ for some $i \geq 0, j \geq 1$ according to [1]. Further, it is known that periodic FDFA 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 [1], 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 FDFA, then every decomposition of uv^ω is accepted by the FDFA. 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.