Theoretical Computer Science 650 (2016) 57–72

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Learning regular omega languages

Dana Angluin[a,1], Dana Fisman[b,*,2]

[a] Yale University, United States
[b] University of Pennsylvania, United States

ARTICLE INFO

Keywords:
Active learning
Language inference
Infinitary languages
Membership queries
Equivalence queries
L^*
Büchi automata

ABSTRACT

We provide an algorithm for learning an unknown regular set of infinite words using membership and equivalence queries. Three variations of the algorithm learn three different canonical representations of regular omega languages using the notion of families of DFAS. One is of size similar to $L_\$$, a DFA representation recently learned using L^* by Farzan et al. The second is based on the syntactic FORC, introduced by Maler and Staiger. The third is introduced herein. We show that the second and third can be exponentially smaller than the first, and the third is at most as large as the second, with up to a quadratic saving with respect to the second.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

The L^* algorithm learns an unknown regular language in polynomial time using membership and equivalence queries [2]. It has proved useful in many areas including AI, neural networks, geometry, data mining, verification and many more. Some of these areas, in particular verification, call for an extension of the algorithm to regular ω-languages, i.e., regular languages over infinite words.

Regular ω-languages are the main means to model reactive systems and are used extensively in the theory and practice of formal verification and synthesis. The question of learning regular ω-languages has several natural applications in this context. For instance, a major critique of *reactive-system synthesis*, the problem of synthesizing a reactive system from a given temporal logic formula, is that it shifts the problem of implementing a system that adheres to the specification in mind to formulating a temporal logic formula that expresses it. A potential customer of a computerized system may find it hard to specify his requirements by means of a temporal logic formula. Instead, he might find it easier to provide good and bad examples of ongoing behaviors (or computations) of the required system, or classify a given computation as good or bad — a classical scenario for interactive learning of an unknown language using membership and equivalence queries.

Another example concerns *compositional reasoning*, a technique aimed to improve scalability of verification tools by decomposing the original verification task into subproblems. The simplification is typically based on the assume-guarantee reasoning principles and requires identifying adequate environment assumptions for components. A recent approach to the automatic derivation of assumptions uses L^* [1,6,18] with a model checker to answer queries to the teacher. Using L^* allows learning only *safety* properties (a subset of ω-regular properties that state that something bad has not happened and that

* Corresponding author.
E-mail address: dana.fisman@gmail.com (D. Fisman).
1 Research of this author was supported by US NSF grant CCF-0916389.
2 Research of this author was supported by US NSF grant CCF-1138996.

http://dx.doi.org/10.1016/j.tcs.2016.07.031
0304-3975/© 2016 Elsevier B.V. All rights reserved.

can be expressed by automata on finite words). To learn *liveness* and *fairness* properties, we need to extend L^* to the full class of regular ω-languages — a problem considered open for many years [12].

The first issue confronted when extending to ω-languages is how to cope with infinite words. Some finite representation is needed. There are two main approaches for that: one considers only finite prefixes of infinite computations and the other considers ultimately periodic words, i.e., words of the form uv^ω where v^ω stands for the infinite concatenation of v to itself. It follows from McNaughton's theorem [16] that two ω-regular languages are equivalent if they agree on the set of ultimately periodic words, justifying their use for representing examples.

Work by de la Higuera and Janodet [7] gives positive results for polynomially learning in the limit *safe* regular ω-languages from *prefixes*, and negative results for learning any strictly subsuming class of regular ω-languages from prefixes. A regular ω-language L is *safe* if for all $w \notin L$ there exists a prefix u of w such that no extension of u is in L. This work is extended in [9] to learning bi-ω languages from subwords.

Saoudi and Yokomori [20] consider ultimately periodic words and provide an algorithm for learning in the limit the class of *local* ω-languages and what they call *recognizable* ω-languages. An ω-language is said to be *local* if there exist $I \subseteq \Sigma$ and $C \subseteq \Sigma^2$ such that $L = I\Sigma^\omega - \Sigma^* C \Sigma^\omega$. An ω-language is referred to as *recognizable* [20] if it is recognizable by a deterministic automaton all of whose states are accepting.

Maler and Pnueli [14] provide an extension of the L^* algorithm, using ultimately periodic words as examples, to the class of regular ω-languages which are recognizable by both deterministic Büchi and deterministic co-Büchi automata. This is the subset for which the straightforward extension of right congruence to infinite words gives a Myhill–Nerode characterization [21]. Generalizing this to wider classes calls for finding a Myhill–Nerode characterization for larger classes of regular ω-languages. This direction of research was taken in [11,15] and is one of the starting points of our work.

In fact the full class of regular ω-languages can be learned using the result of Calbrix, Nivat and Podelski [5]. They define for a given ω-language L the set $L_\$ = \{u\$v \mid u \in \Sigma^*, v \in \Sigma^+, uv^\omega \in L\}$ and show that $L_\$$ is regular by constructing an NFA and a DFA accepting it. Since DFAs are canonical for regular languages, it follows that a DFA for $L_\$$ is a canonical representation of L. Such a DFA can be learned by the L^* algorithm provided the teacher's counterexamples are ultimately periodic words, given e.g. as a pair (u, v) standing for uv^ω — a quite reasonable assumption that is common to the other works too. This DFA can be converted to a Büchi automaton recognizing it. This approach was studied and implemented by Farzan et al. [8]. For a Büchi automaton with m states, Calbrix et al. provide an upper bound of $2^m + 2^{2m^2+m}$ on the size of a DFA for $L_\$$.

So the full class of regular ω-languages can be learnt using membership and equivalence queries, yet not very efficiently. We thus examine an alternative canonical representation of the full class of regular ω-languages. Maler and Staiger [15] show that regular ω-languages can be represented by a family of right congruences (FORC, for short). With a given ω-language they associate a particular FORC, the *syntactic* FORC, which they show to be the coarsest FORC recognizing the language. We adapt and relax the notion of FORC to families of DFAS (FDFA, for short). We show that the syntactic FORC can be factorially smaller than $L_\$$. That is, there exists a family of languages L_n for which the syntactic FDFA is of total size $O(n^2)$ and the minimal DFA for $L_\$$ is of size $\Omega(n!)$. We then provide a third representation, the *recurrent* FDFA. We show that the recurrent FDFA is at most as large the syntactic FDFA, with up to a quadratic saving with respect to the syntactic FDFA.

We provide a learning algorithm L^ω that can learn an unknown regular ω-language using membership and equivalence queries. The learned representations use the notion of families of DFAS (FDFAS). Three variations of the algorithm can learn the three canonical representations: the periodic FDFA (the FDFA corresponding to $L_\$$), the syntactic FDFA (the FDFA corresponding to the syntactic FORC) and the recurrent FDFA. The running time of the three learning algorithms is polynomial in the size of the periodic FDFA.

2. Preliminaries

Let Σ be a finite set of symbols. The set of finite words over Σ is denoted Σ^*, and the set of infinite words, termed ω-words, over Σ is denoted Σ^ω. A *language* is a set of finite words, that is, a subset of Σ^*, while an ω-language is a set of ω-words, that is, a subset of Σ^ω. Throughout the paper we use u, v, x, y, z for finite words, w for ω-words, a, b, c for letters of the alphabet Σ, and i, j, k, l, m, n for natural numbers. We use $[i..j]$ for the set $\{i, i+1, \dots, j\}$. We use $w[i]$ for the i-th letter of w and $w[i..k]$ for the subword of v starting at the i-th letter and ending at the k-th letter, inclusive.

An *automaton* is a tuple $M = \langle \Sigma, Q, q_0, \delta \rangle$ consisting of a finite alphabet Σ of symbols, a finite set Q of states, an initial state q_0 and a transition function $\delta : Q \times \Sigma \to 2^Q$. A run of an automaton on a finite word $v = a_1 a_2 \dots a_n$ is a sequence of states $q_0, q_1, \dots, q_n$ such that $q_{i+1} \in \delta(q_i, a_{i+1})$. A run on an infinite word is defined similarly and results in an infinite sequence of states. The transition function can be extended to a function from $Q \times \Sigma^*$ by defining $\delta(q, \lambda) = q$ and $\delta(q, av) = \delta(\delta(q, a), v)$ for $q \in Q$, $a \in \Sigma$ and $v \in \Sigma^*$. We often use $M(v)$ as a shorthand for $\delta(q_0, v)$ and $|M|$ for the number of states in Q. A transition function is *deterministic* if $\delta(q, a)$ is a singleton for every $q \in Q$ and $a \in \Sigma$, in which case we use $\delta(q, a) = q'$ rather than $\delta(q, a) = \{q'\}$.

By augmenting an automaton with an acceptance condition α, obtaining a tuple $\langle \Sigma, Q, q_0, \delta, \alpha \rangle$, we get an *acceptor*, a machine that accepts some words and rejects others. An acceptor accepts a word if one of the runs on that word is accepting. For finite words the acceptance condition is a set $F \subseteq Q$ and a run on v is accepting if it ends in an accepting state, i.e., if $\delta(q_0, v) \in F$. For infinite words, there are many acceptance conditions in the literature; here we mention three: Büchi, co-Büchi and Muller. Büchi and co-Büchi acceptance conditions are also a set $F \subseteq Q$. A run of a Büchi automaton is accepting if it visits F infinitely often. A run of a co-Büchi is accepting if it visits F only finitely many times. A Muller acceptance condition is a map $\tau : 2^Q \to \{+,-\}$. A run of a Muller automaton is accepting if the set S of states visited infinitely often along the run is such that $\tau(S) = +$. The set of words accepted by an acceptor A is denoted $[\![A]\!]$.

We use three letter acronyms to describe acceptors. The first letter is D or N: D if the transition relation is deterministic and N if it is not. The second letter is one of {F,B,C,M}: F if this is an acceptor over finite words, B, C, M if it is an acceptor over infinite words with Büchi, co-Büchi or Muller acceptance condition, respectively. The third letter is always A for acceptor. For finite words DFAs and NFAs have the same expressive power. For infinite words the theory is much more involved. For instance, DBAs are weaker than NBAs, DMAs are as expressive as NMAs, and NBAs are as expressive as DMAs. A language is said to be *regular* if it is accepted by a DFA. An ω-language is said to be *regular* if it is accepted by a DMA.

An equivalence relation $\sim$ on Σ^* is a *right congruence* if $x \sim y$ implies $xv \sim yv$ for every $x, y, v \in \Sigma^*$. The *index* of $\sim$, denoted $|\sim|$ is the number of equivalence classes of $\sim$. Given a language L its *canonical right congruence* $\sim_L$ is defined as follows: $x \sim_L y$ iff $\forall v \in \Sigma^*$ we have $xv \in L \iff yv \in L$. We use $[\sim]$ to denote the equivalence classes of the right congruence $\sim$ (instead of the more common notation $\Sigma^*/\sim$). For a word $v \in \Sigma^*$ the notation $[v]$ is used for the class of $\sim$ in which v resides.

A right congruence $\sim$ can be naturally associated with an automaton $M_\sim = \langle \Sigma, Q, q_0, \delta \rangle$ as follows: the set of states Q are the equivalence classes of $\sim$. The initial state q_0 is the equivalence class $[\lambda]$. The transition function δ is defined by $\delta([u], \sigma) = [u\sigma]$. Similarly, given an automaton $M = \langle \Sigma, Q, q_0, \delta \rangle$ we can naturally associate with it a right congruence as follows: $x \sim_M y$ iff $\delta(q_0, x) = \delta(q_0, y)$. The Myhill–Nerode Theorem states that a language L is regular iff $\sim_L$ is of finite index. Moreover, if L is accepted by a DFA A then $\sim_A$ refines $\sim_L$. Finally, the index of $\sim_L$ gives the size of the minimal DFA for L.

For ω-languages, the right congruence $\sim_L$ is defined similarly, by quantifying over ω-words. That is, $x \sim_L y$ iff $\forall w \in \Sigma^\omega$ we have $xw \in L \iff yw \in L$. Given a deterministic automaton M we can define $\sim_M$ exactly as for finite words. However, for ω-regular languages, right congruence alone does not suffice to obtain a "Myhill–Nerode" characterization. As an example consider the language $L = \Sigma^* a^\omega$. We have that $\sim_L$ consists of just one equivalence class, but obviously an acceptor recognizing L_1 needs more than a single state.

3. Canonical representations of regular omega languages

As mentioned in the introduction, the language $L_\$ = \{u\$v \mid u \in \Sigma^*,\ v \in \Sigma^+,\ uv^\omega \in L\}$ provides a canonical representation for a regular ω-language L. For instance, the language $L_1 = \Sigma^* a^\omega$ can be represented by $\Sigma^*\$a^+$. As the upper bound of going from a given Büchi automaton of size m to $L_\$$ is quite large ($2^m + 2^{2m^2+m}$) we investigate other canonical representations.

3.1. Second canonical representation – syntactic FORC

Searching for a notion of right congruence adequate for regular ω-languages has been the subject of many works (including [4,10,13,15,22]). In the latest of these [15] Maler and Staiger proposed the notion of a *family of right congruences* or FORC.

Definition 1 *(FORC, recognition by FORC [15]).* A *family of right congruences* (in short FORC) is a pair $\mathcal{R} = (\sim, \{\approx^u\}_{u\in[\sim]})$ such that

1. $\sim$ is a right congruence,
2. $\approx^u$ is a right congruence for every $u \in [\sim]$, and
3. $x \approx^u y$ implies $ux \sim uy$ for every $u, x, y \in \Sigma^*$.

We refer to $\sim$ as the *leading congruence* and to $\approx^u$ as the *progress congruence for* u. An ω-language L is recognized by a FORC $\mathcal{R} = (\sim, \approx^u)$ if it can be written as a union of sets of the form $[u]([v]_u)^\omega$ such that $uv \sim_L u$.[3]

For instance, the language $L_1 = \Sigma^* a^\omega$ can be represented by a FORC whose leading congruence consists of one equivalence class, where the progress congruence for the single equivalence class consists of two equivalence classes, for a^+ and $\Sigma^* \setminus a^+$.

Definition 2 *(Syntactic FORC [15]).* Let $x, y, u \in \Sigma^*$, and L be a regular ω-language. We use $x \approx^u_S y$ iff $ux \sim_L uy$ and $\forall v \in \Sigma^*$ if $uxv \sim_L u$ then $u(xv)^\omega \in L \iff u(yv)^\omega \in L$. The *syntactic* FORC of L is $(\sim_L, \{\approx^u_S\}_{u\in[\sim_L]})$.

[3] The original definition of recognition by a FORC requires also that the language L be saturated by $\mathcal{R}$. An ω-language L is saturated by $\mathcal{R}$ if for every u, v s.t. $uv \sim u$ it holds that $[u]([v]_u)^\omega \subseteq L$. It is shown in [15] that for finite FORCs, covering and saturation coincide. Thus, the definition here only requires that L is covered by $\mathcal{R}$.

Theorem 1 *(Minimality of the syntactic FORC [15]). An ω-language is regular iff it is recognized by a finite* FORC*. Moreover, for every regular ω-language, its syntactic* FORC *is the coarsest* FORC *recognizing it.*

As noted in [4] for an earlier notion of right congruence, the finiteness of the respective right congruence, or families thereof, does not imply the languages is regular. For instance, the language Ult, which consists of all ultimately periodic words, has a syntactic FORC with one equivalence class in its leading congruence and one equivalence class in its progress congruence, and the $L_\$$ representation for Ult is $\Sigma^*\$\Sigma^+$, whose right congruence has two equivalence classes. However, Ult is not regular since no DMA recognizes it.

3.2. Moving to families of DFAs

We have seen in the preliminaries how a right congruence defines an automaton, and that the latter can be augmented with an acceptance criterion to get an acceptor for regular languages. In a similar way, we would like to define a family of automata, and augment it with an acceptance criterion to get an acceptor for regular ω-languages.

Definition 3 *(Family of DFAs (FDFA)).* 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 $A_q = (\Sigma, S_q, s_q^0, \delta_q, F_q)$ for each $q \in Q$.

Note that the definition of FDFA, does not impose the third requirement in the definition of FORC. If needed this condition can be imposed by the progress DFAS themselves.[4]

Definition 4 *(Syntactic FDFA).* Let L be a regular ω-language, and let M be the automaton corresponding to $\sim_L$. For every equivalence class $[u]$ of $\sim_L$ let A_{S}^u be the DFA corresponding to $\approx_{\text{S}}^u$, where the accepting states are the equivalence classes $[v]$ of $\approx_{\text{S}}^u$ for which $uv \sim_L u$ and $uv^\omega \in L$. We use $\mathcal{F}_{\text{S}}$ to denote the FDFA $(M, \{A_{\text{S}}^u\})$, and refer to it as the *syntactic* FDFA.[5]

Example 1. The column labeled "Syntactic" in Fig. 1 gives the progress automata of the syntactic family for the language $L = a^\omega + ab^\omega$. The leading automaton is given in the column labeled "Leading." The other columns of the figure will be discussed in the sequel. It can be seen that although the progress automata $\mathcal{S}_\lambda$ and $\mathcal{S}_a$ accept nothing, the respective syntactic right congruences $\approx_{\text{S}}^\lambda$ and $\approx_{\text{S}}^a$ do have 5 and 4 equivalence classes, respectively.

The following is a direct consequence of Theorem 1 and Definitions 1 and 4.

Proposition 1. *Let L be a regular ω-language and $\mathcal{F}_{\text{S}} = (M, \{A_{\text{S}}^u\})$ the syntactic* FDFA*. Let $w \in \Sigma^\omega$ be an ultimately periodic word. Then $w \in L$ iff there exists u and v such that $w = uv^\omega$, $uv \sim_L u$ and $v \in [\![A_{\text{S}}^{\tilde{u}}]\!]$ where $\tilde{u} = M(u)$.*

To get an understanding of the subtleties in the definition of $\approx_{\text{S}}^u$ we consider the following simpler definition of a right congruence for the progress automata, and the corresponding FDFA. It is basically the FDFA version of $L_\$$.

Definition 5 *(Periodic FDFA).* Let $x, y, u \in \Sigma^*$ and L be an ω-language. We use $x \approx_{\text{P}}^u y$ iff $\forall v \in \Sigma^*$ we have $u(xv)^\omega \in L \iff u(yv)^\omega \in L$. Let M be the automaton corresponding to $\sim_L$. For every equivalence class $[u]$ of $\sim_L$ let A_{P}^u be the DFA corresponding to $\approx_{\text{P}}^u$ where the accepting states are the equivalence classes $[v]$ of $\approx_{\text{P}}^u$ for which $uv^\omega \in L$. We use $\mathcal{F}_{\text{P}}$ to denote the FDFA $(M, \{A_{\text{P}}^u\})$, and refer to it as the *periodic* FDFA.[6]

Example 2. The column labeled "Periodic" in Fig. 1 gives the progress automata of the periodic family for the language $L = a^\omega + ab^\omega$; the leading automaton is given in the column labeled "Leading."

The following proposition relates L to its periodic FDFA.

Proposition 2. *Let L be a regular ω-language and $\mathcal{F}_{\text{P}} = (M, \{A_{\text{P}}^u\})$ the periodic* FDFA*. Let $w = uv^\omega \in \Sigma^\omega$ be an ultimately periodic word. Then $w \in L$ iff $v \in [\![A_{\text{P}}^{\tilde{u}}]\!]$ where $\tilde{u} = M(u)$.*

Proof. Because $\tilde{u} = M(u)$, we have $u \sim_L \tilde{u}$. Thus $uv^\omega \in L$ if and only if $\tilde{u}v^\omega \in L$. By the definition of the periodic FDFA of L, we have $\tilde{u}v^\omega \in L$ if and only if $v \in [\![A_{\text{P}}^{\tilde{u}}]\!]$, which concludes the proof. $\square$

[4] In [11] Klarlund also suggested the notion of a family of DFAS. However, that work did require the third condition in the definition of FORC to hold.

[5] The syntactic FDFA is well defined since, as shown in [15], $uv^\omega \in L$ implies $[u][v]^\omega \subseteq L$.

[6] It is easy to see that $uv^\omega \in L$ implies $[u][v]^\omega \subseteq L$. Thus the periodic FDFA is well defined.

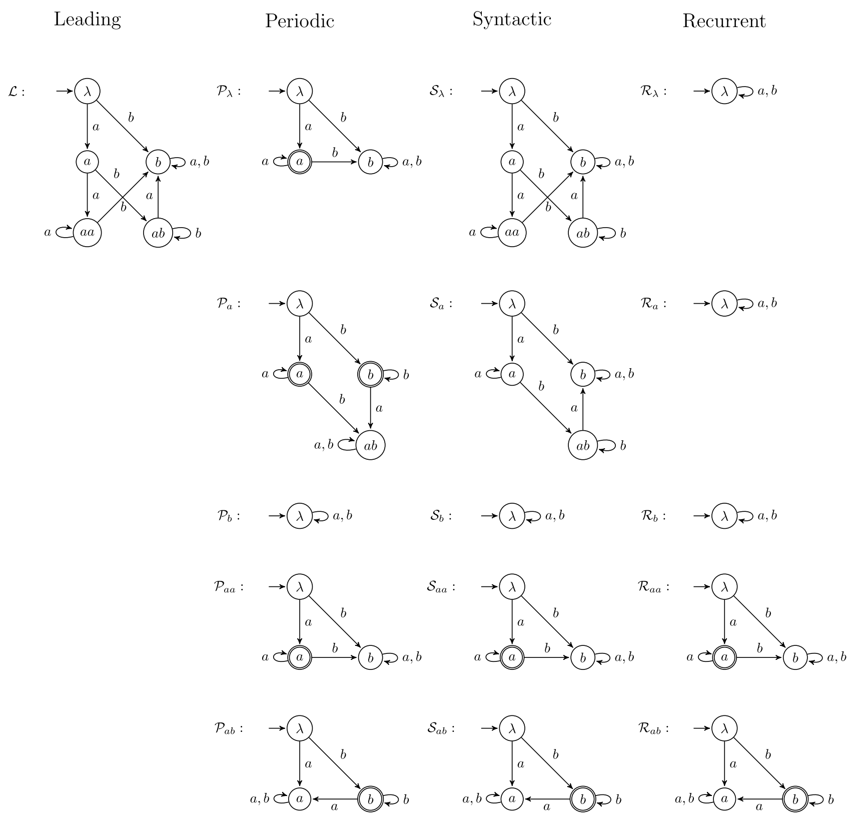

Fig. 1. The periodic FDFA $(\mathcal{L}, \{\mathcal{P}\})$, syntactic FDFA $(\mathcal{L}, \{\mathcal{S}\})$ and recurrent FDFA $(\mathcal{L}, \{\mathcal{R}\})$ of $L = a^\omega + ab^\omega$.

Let L be a fixed regular ω-language. It follows from the definitions that for all $u, x, y \in \Sigma^*$, if $x \sim_L y$ and $x \approx^u_{\mathrm{P}} y$ then $x \approx^u_{\mathrm{S}} y$. Thus

$$|\approx^u_{\mathrm{S}}| \leq |\sim_L| \cdot |\approx^u_{\mathrm{P}}|.$$

Thus the total size of the syntactic FDFA is bounded by a polynomial in the total size of the periodic FDFA.

Maler and Staiger show that the syntactic FORC is the coarsest FORC. They do not compare its size with that of other representations. Below we show that the syntactic FDFA can be factorially more succinct than the periodic FDFA, and the same arguments can be used to show that the syntactic FORC can be factorially more succinct than the DFA for $L_\$$. Intuitively, the reason is that $\mathcal{F}_{\mathrm{P}}$ pertinaciously insists on finding every period of u, while $\mathcal{F}_{\mathrm{S}}$ may not accept a valid period, provided it accepts some repetition of it. For instance, let $L = (aba + bab)^\omega$. Then A^λ_{P} accepts ab because $(ab)^\omega \in L$, and while A^λ_{S} rejects ab because $\lambda \nsim_L ab$, it does accept the repetition $ababab$. This flexibility is common to all acceptance conditions used in the theory of ω-automata (Büchi, Muller, etc.) but is missing from $L_\$$ and $\mathcal{F}_{\mathrm{P}}$. As the example in the following theorem shows, this can make a very big difference.

Theorem 2. *There exists a family of languages L_n whose syntactic* FDFA *has a total of $O(n^2)$ states but whose periodic* FDFA *has at least $n!$ states.*

Proof. Consider the languages L_n over the alphabet $\Sigma_n = [0..n]$ described by the DBA $\mathcal{B}$ in Fig. 2 on the left.[7] The leading automaton $\mathcal{L}$ looks like $\mathcal{B}$ but has no accepting states. The syntactic progress DFA for the empty word is described by $\mathcal{S}_\lambda$ (in the middle), the syntactic progress DFA for any $i \in [1..n]$ is given by $\mathcal{S}_i$ (on the right), and the syntactic progress DFA for $\bot$ is the trivial DFA accepting the empty language.

We claim that the progress automaton for λ in the periodic FDFA requires at least $(n+1)!$ states. The idea of the proof is as follows. Given a word v we use f_v to denote the function from $[0..n]$ to $[0..n]$ that satisfies $f_v(i) = \delta_{\mathcal{B}}(i, v)$, where $\delta_{\mathcal{B}}$ is the transition relation of the Büchi automaton $\mathcal{B}$. We show that with each permutation $\pi = (i_0\ i_1\ \dots\ i_n)$ of $[0..n]$ we can associate a word v_π (of length at most $3n$) such that $f_{v_\pi} = \pi$ (i.e. $f_{v_\pi}(k) = i_k$ for every $k \in [0..n]$). Let V be the set of all

[7] The automata for the languages L_n are the deterministic version of the automata for a family M_n introduced by Michel [17] to prove there exists an NBA with $O(n)$ states whose complement cannot be recognized by an NBA with fewer than $n!$ states.

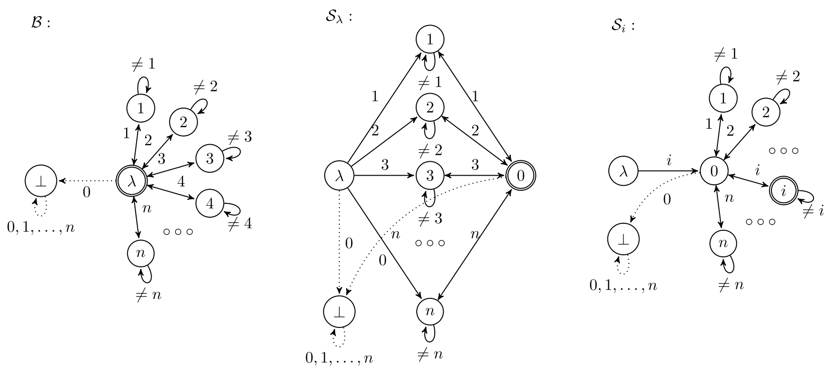

Fig. 2. A Büchi automaton $\mathcal{B}$ for L_n, and the syntactic FDFA for L_n.

Table 1
Distinguishing word y.

Case	Condition	y
1	$i_0 = 0,\ j_0 \neq 0$	$j_0 0 j_0$
2	$i_0 \neq 0,\ j_0 = 0$	$i_0 0 i_0$
3	$i_0 \neq 0,\ j_0 \neq 0,\ i_0 \neq j_0$	$i_0 0 i_0 j_0$
4	$i_0 = j_0 = 0,\ i_k \neq j_k,\ i_k = k$	$k j_k 0 j_k$
5	$i_0 = j_0 = 0,\ i_k \neq j_k,\ j_k = k$	$k i_k 0 i_k$
6	$i_0 = j_0 = 0,\ i_k \neq j_k,\ i_k \neq k,\ j_k \neq k$	$k j_k 0 j_k i_k$
7	$i_0 = j_0 \neq 0,\ i_k \neq j_k,\ i_k = k$	$i_0 k j_k 0 j_k$
8	$i_0 = j_0 \neq 0,\ i_k \neq j_k,\ j_k = k$	$i_0 k i_k 0 i_k$
9	$i_0 = j_0 \neq 0,\ i_k \neq j_k,\ i_k \neq k,\ j_k \neq k$	$i_0 k j_k 0 j_k i_k$

such words. We then show that for each $v_1, v_2 \in V$ we can find a word y such that $v_1 y^\omega \in L_n$ iff $v_2 y^\omega \notin L$. Since V is of size $(n+1)!$ any DFA with fewer than $(n+1)!$ states is bound to make a mistake.

First, we show that we can associate with each π the promised word v_π. With $\pi_0 = (0\ 1\ \ldots\ n)$ we associate the word λ. It is known that one can get from any permutation π to any other permutation π' by a sequence of at most n transpositions (transformations switching exactly two elements). It thus suffices to provide for any permutations $\pi = (i_0\ i_1\ \ldots\ i_n)$ and $\pi' = (i'_0\ i'_1\ \ldots\ i'_n)$ differing in only two elements, a word u such that $f_{v_\pi u} = \pi'$. Suppose π and π' differ in indices j and k. If both $i_j \neq 0$ and $i_k \neq 0$ then the word $i_j i_k i_j$ will take state i_j to i_k and state i_k to i_j and leave all other states unchanged. If $i_j = 0$ the word i_k does the job, and symmetrically if $i_k = 0$ we choose i_j. We have thus shown that with each permutation π we can associate a word v_π such that $f_{v_\pi} = \pi$.

Now, we show that for each two such words v_1, v_2 we can find a differentiating word y. Let $f_{v_1} = (i_0\ i_1\ \ldots\ i_n)$ and $f_{v_2} = (j_0\ j_1\ \ldots\ j_n)$. Table 1 explains how we choose y. In the first three cases $i_0 \neq j_0$ and we get $f_{v_1 y}(0) = 0$ and $f_{v_2 y}(0) = \bot$ or vice versa. In the rest of the cases we choose $k > 0$ such that $i_k \neq j_k$ and we get $f_{v_1 y}(0) = k$, $f_{v_1 y}(k) = 0$ and $f_{v_2 y}(0) = k, f_{v_2 y}(k) = \bot$ or vice versa. Thus $f_{(v_1 y)^2} = f^2_{v_1 y}(0) = 0$ and $f_{(v_2 y)^n} = f^n_{v_2 y}(0) = \bot$ for any $n \geq 2$, or vice versa. Thus $(v_1 y)^\omega \in L$ iff $(v_2 y)^\omega \notin L$. □

3.3. Families of FDFAs as acceptors

Families of automata are not an operational acceptor. The answer to whether a given ultimately periodic word $w \in \Sigma^\omega$ is accepted by the FDFA relies on the existence of a decomposition of w into uv^ω, but it is not clear how to find such a decomposition. We would like to use families of automata as acceptors for pairs of words, such that (u, v) being accepted implies uv^ω is. We can try defining acceptance as follows.

Definition 6 *(FDFA exact acceptance).* Let $\mathcal{F} = (M, \{A_u\})$ be a FDFA and u, v finite words. We say that $(u, v) \in [\![\mathcal{F}]\!]_{\mathbb{E}}$ if $v \in [\![A_{\tilde{u}}]\!]$ where $\tilde{u} = M(u)$.

Since our goal is to use families of automata as acceptors for regular ω-languages, and an ultimately periodic ω-word w may be represented by different pairs (u, v) and (x, y) such that $w = uv^\omega = xy^\omega$ (where $u \neq x$ and/or $v \neq y$) it makes sense to require the representation to be *saturated*, in the following sense.

Definition 7 *(Saturation).* A language L of pairs of finite words is said to be *saturated* if for every u, v, x, y such that $uv^\omega = xy^\omega$ we have $(u, v) \in L \iff (x, y) \in L$.

Calbrix et al. [5] have showed that (1) $L_\$$ is saturated, and (2) a regular language of pairs K is saturated iff it is $L_\$$ for some regular ω-language L. It is thus not surprising that the periodic family is saturated as well.

Proposition 3. *Let L be a regular ω-language and $\mathcal{F}_{\mathrm{P}}$ the corresponding periodic* FDFA. *Then $[\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{E}}$ is saturated.*

Proof. Suppose $w = uv^\omega = xy^\omega$ is an ultimately periodic word. By Proposition 2, we have $(u, v) \in [\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{E}}$ iff $w \in L$ iff $(x, y) \in [\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{E}}$, so either both (u, v) and (x, y) are members of $[\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{E}}$ or both non-members, proving saturation. □

The language $[\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{E}}$ on the other hand, is not necessarily saturated. Consider $L = a^\omega + ab^\omega$. Its periodic and syntactic family are given in Fig. 1. Let $x = aa$, $y = a$, $u = a$, $v = a$. It can be seen that although $xy^\omega = uv^\omega$ we have $(aa, a) \in [\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{E}}$ yet $(a, a) \notin [\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{E}}$. The reason is that, in order to be smaller, the syntactic family does not insist on finding every possible legitimate period v of u (e.g. period a of a in this example). Instead, it suffices to find a repetition v^k of it such that in the leading automaton, v^k leads back to the state reached on input u.

Given a right congruence $\sim$ of finite index and an ultimately periodic word w we say that (x, y) is a *factorization* of w with respect to $\sim$ if $w = xy^\omega$ and $xy \sim x$. If w is given by a pair (u, v) so that $w = uv^\omega$ we can define the *normalized factorization* of (u, v) as the pair (x, y) such that (x, y) is a factorization of uv^ω, $x = uv^i$, $y = v^j$ and i, j are the smallest for which $uv^i \sim_L uv^{i+j}$. Since $\sim$ is of finite index, there must exist such i and j such that $i + j < |{\sim}| + 1$. If we base our acceptance criteria on the normalized factorization, we achieve that $[\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{N}}$ is saturated as well.

Definition 8 *(FDFA normalized acceptance).* Let $\mathcal{F} = (M, \{A_u\})$ be an FDFA, and u, v finite words. We say that $(u, v) \in [\![\mathcal{F}]\!]_{\mathbb{N}}$ if $y \in [\![A_{M(x)}]\!]$ where (x, y) is the normalized factorization of (u, v) with respect to $\sim_M$.

Proposition 4. *Let L be a regular ω-language and $\mathcal{F}_{\mathrm{P}}$ and $\mathcal{F}_{\mathrm{S}}$ the corresponding periodic and syntactic families. Then $[\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{N}}$ and $[\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{N}}$ are saturated.*

Proof. Let $w = uv^\omega$ be an ultimately periodic word. We show that $w \in L$ iff $(u, v) \in [\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{N}}$, which proves that $[\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{N}}$ is a saturated acceptor of L. Let (x, y) be the normalized factorization of (u, v) with respect to $\sim_L$. Then for some i and j, $x = uv^i$, $y = v^j$ and $xy \sim_L x$. Let $\tilde{x} = M(x)$. Thus $x \sim_L \tilde{x}$ and $\tilde{x}y \sim_L \tilde{x}$. Then $(u, v) \in [\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{N}}$ iff $y \in [\![A_{\tilde{x}}]\!]$ iff $\tilde{x}y \sim_L \tilde{x}$ and $\tilde{x}y^\omega \in L$ iff $xy^\omega \in L$ iff $w \in L$.

Similar arguments show that $[\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{N}}$ is also saturated. □

3.4. New canonical representation – the recurrent FDFA

We note that there is some redundancy in the definition of the syntactic FDFA: the condition that $ux \sim_L uy$ can be checked on the leading automaton rather than refine the definitions of the $\approx^u_{\mathrm{S}}$'s. We thus propose the following definition of right congruence, and corresponding FDFA.

Definition 9 *(Recurrent FDFA).* Let $x, y, u \in \Sigma^*$ and L be an ω-language. We use $x \approx^u_{\mathrm{R}} y$ iff $\forall v \in \Sigma^*$ if $uxv \sim_L u$ and $u(xv)^\omega \in L$ then $uyv \sim_L u$ and $u(yv)^\omega \in L$. We use $\mathcal{F}_{\mathrm{R}}$ to denote the FDFA $(M, \{A^u_{\mathrm{R}}\})$ where the accepting states of A^u_{R} are those v for which $uv \sim_L u$ and $uv^\omega \in L$. We refer to $\mathcal{F}_{\mathrm{R}}$ as the *recurrent* FDFA.

Example 3. The column labeled "Recurrent" in Fig. 1 gives the progress automata of the recurrent family for the language $L = a^\omega + ab^\omega$; the leading automaton is given in the column labeled "Leading." It can be seen that compared to the syntactic family, the recurrent progress automata for λ and a do not have more states than necessary.

Note that the proof of Proposition 4 did not make use of the additional requirement $ux \sim_L uy$ of $\approx^u_{\mathrm{S}}$. The same arguments thus show that the recurrent FDFA is a saturated acceptor of L.

Proposition 5. *Let L be a regular ω-language and $\mathcal{F}_{\mathrm{R}} = (M, \{A^u_{\mathrm{R}}\})$ be its recurrent* FDFA. *Then $[\![\mathcal{F}_{\mathrm{R}}]\!]_{\mathbb{N}}$ is saturated and is an acceptor of L.*

Let L be a fixed regular ω-language. It follows from the definitions of $\approx^u_{\mathrm{S}}$ and $\approx^u_{\mathrm{R}}$ that for all $u, x, y \in \Sigma^*$, $x \approx^u_{\mathrm{S}} y$ implies $x \approx^u_{\mathrm{R}} y$, that is, $\approx^u_{\mathrm{S}}$ refines $\approx^u_{\mathrm{R}}$. Also for all $u, x, y \in \Sigma^*$, if $x \sim_L y$ and $x \approx^u_{\mathrm{R}} y$ then $x \approx^u_{\mathrm{S}} y$. Thus

$$|\approx^u_{\mathrm{R}}| \leq |\approx^u_{\mathrm{S}}| \leq |\sim_L| \cdot |\approx^u_{\mathrm{R}}|.$$

Thus there is at most a quadratic size reduction in the recurrent FDFA, with respect to the syntactic FDFA. We show a matching lower bound.

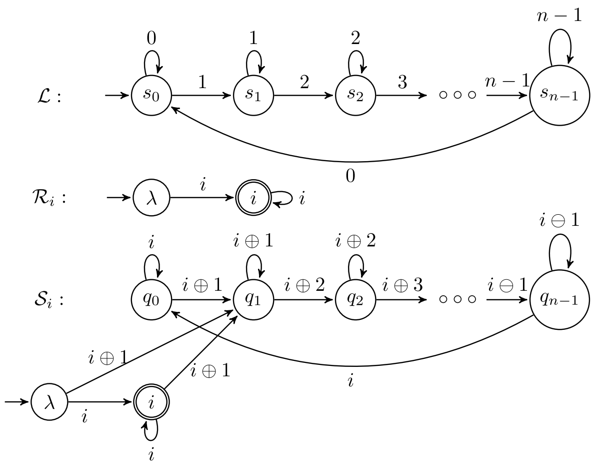

Fig. 3. The leading automaton $\mathcal{L}$ for T_n, and the syntactic and recurrent progress automata for i, $\mathcal{S}_i$ and $\mathcal{R}_i$.

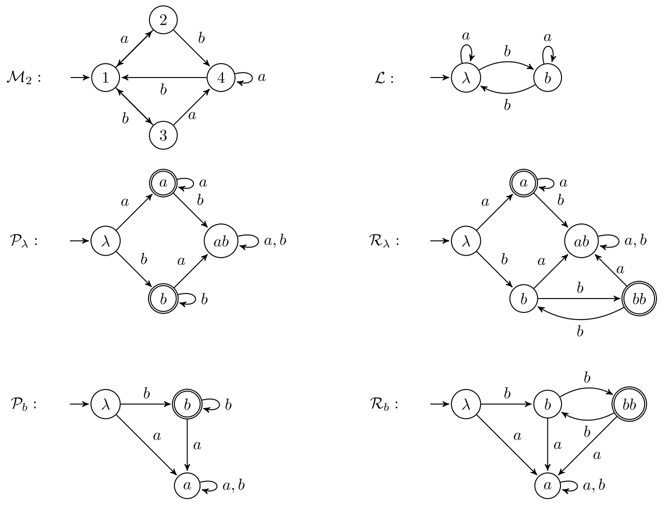

Fig. 4. An example where the periodic family is smaller than the recurrent one.

Proposition 6. *There exists a family of languages T_n such that the total size of the syntactic* FDFA *for T_n is $\Theta(n^2)$ and the total size of the recurrent* FDFA *is $\Theta(n)$.*

Proof. Consider the alphabet $\Sigma_n = \{0, 1, \ldots, n-1\}$. Let L_i abbreviate i^+. Let U_i be the set of ultimately periodic words $(L_0L_1 \ldots L_{n-1})^* (L_0L_1 \ldots L_i) i^\omega$. Finally let T_n be the union $U_0 \cup U_1 \cup \ldots .U_{n-1}$. In Fig. 3 we show its leading DFA $\mathcal{L}$, the syntactic and recurrent progress DFAs for state i, $\mathcal{S}_i$ and $\mathcal{R}_i$, respectively. The sink state in each automaton is not shown, and $\oplus, \ominus$ are plus and minus modulo n. The total number of states for the recurrent FDFA is $(n+1) + 3n + 1$ and for the syntactic FDFA it is $(n+1) + n(n+3) + (n+1)$. □

We observe that the recurrent family may not produce a minimal result. Working with the normalized acceptance criterion, we have that a progress DFA P_u for leading state u should satisfy $[u](\llbracket P_u \rrbracket \cap C_u) = L \cap [u]C_u$ where $C_u = \{v \mid uv \sim_L u\}$. Thus, in learning P_u we have *don't cares* for all the words that are not in C_u. Minimizing a DFA with don't cares is an NP-hard problem [19]. The recurrent FDFA chooses to treat all don't cares as rejecting.

Example 4. Fig. 4 is an example of a case in which the periodic family is smaller than the recurrent one. The Muller automaton $\mathcal{M}_2$ whose accepting sets are $\{1, 2\}$ and $\{1, 3\}$ accepts the union of the following two languages:

1. any string of a's and b's, followed by b^ω
2. any string of a's and b's with an even number of b's, followed by a^ω.

Its leading automaton $\mathcal{L}$ has two states λ and b, and the figure shows the corresponding periodic progress automata $\mathcal{P}_\lambda$ and $\mathcal{P}_b$ and the corresponding recurrent progress automata $\mathcal{R}_\lambda$ and $\mathcal{R}_b$. The states b and bb are distinguished in $\mathcal{R}_b$ but not in $\mathcal{P}_b$. To see why set $u = b$, $x = bb$, $y = b$ and $v = \lambda$. We get that $uxv \sim_L u$ (because $bbb \sim_L b$) and $u(xv)^\omega = b(bb)^\omega \in L$. However, while $u(yv)^\omega = b(b)^\omega \in L$, $uyv \nsim_L u$ (because $bb \nsim_L b$).

4. Learning regular omega languages via families of DFAs

Algorithm 1: The Learner L^ω.

```
Initialize leading table $\mathcal{T} = (S, \tilde{S}, E, T)$ with $S = \tilde{S} = \{\lambda\}$, $E = \{(\lambda, \sigma) | \sigma \in \Sigma\}$.
$CloseTable(\mathcal{T}, \text{ENT}_1, \text{DFR}_1)$ and let $M = Aut_1(\mathcal{T})$.
forall $u \in \tilde{S}$ do
    Initialize the table for $u$, $\mathcal{T}_u = (S_u, \tilde{S_u}, E_u, T_u)$, with $S_u = \tilde{S_u} = E_u = \{\lambda\}$.
    $CloseTable(\mathcal{T}_u, \text{ENT}_2^u, \text{DFR}_2^u)$ and let $A_u = Aut_2(\mathcal{T}_u)$.
Let $(a, u, v)$ be the teacher's response on equivalence query $\mathcal{H} = (M, \{A_u\})$.
while $a \neq$"yes" do
    Let $(x, y)$ be the normalized factorization of $(u, v)$ with respect to $M$.
    Let $\tilde{x}$ be $M(x)$.
    if $\text{MQ}(x, y) \neq \text{MQ}(\tilde{x}, y)$ then
        $E = E \cup FindDistinguishingExperiment(x, y)$.
        $CloseTable(\mathcal{T}, \text{ENT}_1, \text{DFR}_1)$ and let $M = Aut_1(\mathcal{T})$.
        forall $u \in \tilde{S}$ do
            $CloseTable(\mathcal{T}_u, \text{ENT}_2^u, \text{DFR}_2^u)$ and let $A_u = Aut_2(\mathcal{T}_u)$.
    else
        $E_{\tilde{x}} = E_{\tilde{x}} \cup FindDistinguishingExperiment(\tilde{x}, y)$.
        $CloseTable(\mathcal{T}_{\tilde{x}}, \text{ENT}_2^{\tilde{x}}, \text{DFR}_2^{\tilde{x}})$ and let $A_x = Aut_2(\mathcal{T}_{\tilde{x}})$.
    Let $(a, u, v)$ be the teacher's response on equivalence query $\mathcal{H} = (M, \{A_u\})$.
return $\mathcal{H}$
```

In the previous section we have provided three canonical representations of regular ω-languages as families of DFAs. For each of them, $[\![\mathcal{F}]\!]_\mathbb{N}$ is saturated. Thus we do not subsequently distinguish between $[\![\mathcal{F}]\!]_\mathbb{N}$ as a set of ordered pairs (u, v) and the ω-regular language L consisting of all strings uv^ω such that $(u, v) \in [\![\mathcal{F}]\!]_\mathbb{N}$.

The L^* algorithm provides us an efficient way to learn a DFA for an unknown regular language. Have we reduced the problem to using L^* for the different DFAs of the family? Not quite. This would be true if we had oracles answering membership and equivalence for the languages of the leading and progress DFAs. But the question we consider assumes we have oracles for answering membership and equivalence queries for the unknown regular ω-language. Specifically, the membership oracle answers whether a given pair (u, v) of finite strings satisfies $uv^\omega \in L$, and the equivalence oracle answers whether a given FDFA $\mathcal{F}$ satisfies $[\![\mathcal{F}]\!]_\mathbb{N} = L$, and returns a counterexample if not. The counterexample is in the format (a, u, v) where a is one of the strings "yes" or "no", and if it is "no" then uv^ω is in $(L \setminus [\![\mathcal{F}]\!]_\mathbb{N}) \cup ([\![\mathcal{F}]\!]_\mathbb{N} \setminus L)$.

We use a common scheme for learning the three families ($\mathcal{F}_\text{P}$, $\mathcal{F}_\text{S}$ and $\mathcal{F}_\text{R}$) under the normalized acceptance criteria, see Algorithm 1. This is a modification of the L^* algorithm to learn an unknown DFA using membership and equivalence queries [2]. We first explain the general scheme. Then we provide the necessary details for obtaining the learning algorithm for each of the families, and prove correctness.

4.1. Auxiliary procedures

The algorithm makes use of the notion of an *observation table*. An observation table is a tuple $\mathcal{T} = (S, \tilde{S}, E, T)$ where S is a prefix-closed set of strings, E is a set of experiments trying to differentiate the S strings, and $T : S \times E \to D$ stores in entry $T(s, e)$ an element in some domain D. Some criterion should be given to determine when two strings $s_1, s_2 \in S$ should be considered distinct (presumably by considering the contents of the respective rows of the table). The component $\tilde{S}$ is the subset of strings in S considered distinct. A table is *closed* if S is prefix closed and for every $s \in \tilde{S}$ and $a \in \Sigma$ we have $sa \in S$.

The procedure *CloseTable* thus uses two sub-procedures ENT and DFR to fulfill its task. Procedure ENT is used to fill in the entries of the table. This procedure invokes a call to the membership oracle. The procedure DFR is used to determine which rows of the table should be differentiated. Closing the leading table is done using ENT_1 and DFR_1. Closing the progress table for u is done using ENT_2^u and DFR_2^u.

A closed table can be transformed into an automaton by identifying the automaton states with $\tilde{S}$, the initial state with the empty string, and for every letter $a \in \Sigma$ defining the transition $\delta(s_1, a) = s_2$ iff $s_2 \in \tilde{S}$ is the representative of $s_1 a$. By designating certain states as accepting, e.g. those for which $T(s, \lambda) = d_*$ for some designated $d_* \in D$, we get a DFA. Procedures $Aut_1(\mathcal{T})$ and $Aut_2(\mathcal{T})$ are used for performing this transformation for the leading and progress tables respectively.

4.2. The main scheme

The algorithm starts by initializing and closing the leading table (lines 1–2), and the respective progress tables (lines 3–5) and asking an equivalence query about the resulting hypothesis (line 6). The algorithm then repeats the following loop (lines 7–18) until an equivalence query returns "yes".

If the equivalence query returns a counterexample (u, v) the learner first obtains the normalized factorization (x, y) of (u, v) with respect to its current leading automaton (line 8). It then checks whether membership queries for (x, y) and $(\tilde{x}, y)$, where $\tilde{x}$ is the state M arrives at after reading x, return different results. If so, it calls the procedure *FindDistinguishingExperiment* to find a distinguishing experiment to add to the leading table (line 11). It then closes the leading table and all the progress tables (lines 12–14) and obtains a new hypothesis $\mathcal{H}$ (line 18).

If membership queries for (x, y) and $(\tilde{x}, y)$ return the same results, it calls the procedure *FindDistinguishingExperiment* to find a distinguishing experiment in the progress automaton for $\tilde{x}$ (line 16). It then closes this table (line 17) and obtains a new hypothesis (line 18).

It is clear that if the learning algorithm halts, its output $\mathcal{H}$ is such that $[\![\mathcal{H}]\!]_{\mathbb{N}} = L$, where L is the regular ω-language to be learned. We prove that the algorithm halts; its time complexity is discussed at the end of the section.

4.3. Specializing for the periodic, syntactic and recurrent families

We now provide the details for specializing L^ω to learn the different families $\mathcal{F}_{\text{P}}$, $\mathcal{F}_{\text{S}}$ and $\mathcal{F}_{\text{R}}$.

The algorithms differ in the content they put in the progress tables (i.e., procedure ENT^u_2), in the criterion for differentiating rows in a progress table (i.e., procedure DFR^u_2), the states they choose to be accepting (i.e., procedure Aut_2) and the way they find a distinguishing experiment (i.e., procedure *FindDistinguishingExperiment*). The details of the latter are given within the respective proofs of termination.

For learning the leading automaton, which is same in all three families, the following procedures: ENT_1, DFR_1 and Aut_1 are used. For $u, x, y \in \Sigma^*$, the procedure $\text{ENT}_1(u, (x, y))$ returns T or F depending on whether uxy^ω is in the unknown language L. Given two row strings $u_1, u_2 \in S$ the procedure $\text{DFR}_1(u_1, u_2)$ returns true if there exists $w \in E$ such that $T(u_1, w) \neq T(u_2, w)$. We use Aut_1 for the procedure transforming the leading table into a DFA with no accepting states.

For the periodic FDFA, given $u, x, v \in \Sigma^*$, we have $\text{ENT}^u_{\text{P}}(x, v) = \text{T}$ iff $u(xv)^\omega \in L$, and $\text{DFR}^u_{\text{P}}(x_1, x_2)$ is simply $\exists v \in E_u$ such that $T_u(x_1, v) \neq T_u(x_2, v)$. The procedure Aut_{P} declares a state x as accepting if $T_u(x, \lambda) = \text{T}$.

Theorem 3. *When the learner L^ω is called with ENT_1, DFR_1, Aut_1 and ENT^u_{P}, DFR^u_{P}, Aut_{P}, it halts and returns a correct FDFA for L whose total size does not exceed that of the periodic FDFA for L.*

Proof. It is clear that if two states s_1 and s_2 in the leading table are distinguished, then $s_1 \not\sim_L s_2$, so the number of states of the leading automaton M cannot exceed $|\sim_L|$. Similarly, if u is a state of the leading automaton then the number of states in the progress automaton A_u is never more than $|\approx^u_{\text{P}}|$, the number of states in the progress automaton corresponding to u in the periodic FDFA for L. Thus it suffices to show that in each iteration of the while loop at least one new state is added to one of the tables.

Suppose the returned counterexample is (u, v), and its normalized factorization with respect to the current leading automaton M is (x, y). The learner then checks whether membership queries for (x, y) and $(\tilde{x}, y)$ return different results where $\tilde{x} = M(x)$.

If membership queries for (x, y) and $(\tilde{x}, y)$ return different answers, then we may find an experiment to distinguish states in the leading table as follows. Let $|x| = n$ and for $i \in [1..n]$ let $s_i = M(x[1..i])$ be the state of the leading automaton reached after reading the first i symbols of x. Then $\tilde{x} = s_n$, and we know that a sequence of membership queries with (x, y), $(s_1x[2..n], y)$, $(s_2x[3..n], y)$, and so on, up to $(s_n, y) = (\tilde{x}, y)$ has different answers for the first and last queries. Thus, a sequential search of this sequence suffices to find a consecutive pair, say $(s_{i-1}x[i..n], y)$ and $(s_ix[i+1..n], y)$, with different answers to membership queries. This shows that the experiment $(x[i+1..n], y)$ distinguishes $s_{i-1}x[i]$ from s_i in the leading table, though $\delta(s_{i-1}, x[i]) = s_i$, so that after adding it there will be at least one more state in the leading automaton.

If membership queries for (x, y) and $(\tilde{x}, y)$ return the same answers, we look for an experiment that will distinguish a new state in the progress table of $\tilde{x}$. Let $|y| = n$ and for $i \in [1..n]$ let $s_i = A_{\tilde{x}}(y[1..i])$ be the state reached by $A_{\tilde{x}}$ after reading the first i symbols of y. Consider the sequence (λ, y), $(s_1, y[2..n])$, $(s_2, y[3..n])$, up to (s_n, λ). Then we know that membership queries for $(\tilde{x}, y)$ and $(\tilde{x}, s_n)$ return different results. We can thus find, in an analogous manner to the first case, a suffix y' of y that is a differentiating experiment for the progress table for $\tilde{x}$. $\square$

For the syntactic FDFA, given $u, x, v \in \Sigma^*$, the procedure $\text{ENT}^u_{\text{S}}(x, v)$ returns a pair $(m, c) \in \{\text{T}, \text{F}\} \times \{\text{T}, \text{F}\}$ such that $m = \text{T}$ iff $u(xv)^\omega \in L$ and $c = \text{T}$ iff $M(uxv) = M(u)$. Given two rows x_1 and x_2 in the progress table T_u corresponding to leading state u, the procedure $\text{DFR}^u_{\text{S}}(x_1, x_2)$ returns true if either $M(ux_1) \neq M(ux_2)$, or there exists an experiment $v \in E_u$ for which $T_u(x_1, v) = (m_1, c_1)$, $T_u(x_2, v) = (m_2, c_2)$ and $(c_1 \vee c_2) \wedge (m_1 \neq m_2)$. The procedure Aut_{S} declares a state x as accepting if $T_u(x, \lambda) = (\text{T}, \text{T})$.

Theorem 4. *When the learner L^ω is called with* ENT_1, DFR_1, Aut_1 *and* ENT^u_{S}, DFR^u_{S}, Aut_{S}, *it halts and returns a correct* FDFA *for L whose total size does not exceed that of the syntactic* FDFA *for L.*

Proof. In the case of the syntactic FDFA, the upper bound on the number of states of the current leading automaton M remains $|\sim_L|$. However, if u is a state of M, then states x_1 and x_2 of the progress automaton for u are distinguished either because $M(ux_1) \neq M(ux_2)$ or there exists $v \in \Sigma^*$ such that $u(x_1 v)^\omega \in L$ and $u(x_2 v)^\omega \notin L$ or vice versa. The number of such pairwise distinguished states cannot exceed $|\sim_L| \cdot |\approx^u_{\text{P}}|$.

Thus it suffices to show that each iteration of the while loop creates a new state in some table. The proof for the first part is same as in Theorem 3. For the second part, let (x, y) be the normalized factorization of the counterexample (u, v) with respect to M, and let $\tilde{x} = M(x)$. We consider the sequence of pairs (λ, y), $(s_1, y[2..n])$, $(s_2, y[3..n])$, up to (s_n, λ) as we did in the periodic case. Now, however, the fact that two experiments (x_1, v), (x_2, v) differ in the answer to membership queries does not guarantee they will be distinguished, as this fact might be hidden if for both $M(\tilde{x}x_i v) \neq M(\tilde{x})$. Let (m_0, c_0), $(m_1, c_1), \ldots, (m_n, c_n)$ be the respective results for the entry query for $\tilde{x}$ for each pair. Because $M(xy) = M(x)$ and $M(x) = M(\tilde{x})$, we know that $M(\tilde{x}y) = M(\tilde{x})$ and $c_0 = \mathsf{T}$. We consider two cases: either $c_i = \mathsf{T}$ for all i, or not.

If $c_i = \mathsf{T}$ for $0 \leq i \leq n$, then we know that one of (m_0, c_0) and (m_n, c_n) is (T, T) and the other is (F, T). Let i be the least positive integer such that $m_{i-1} \neq m_i$. Then the experiment $y[i+1 \ldots n]$ distinguishes row $s_i y[i]$ from row s_{i-1} in the progress table for $\tilde{x}$. If it is not the case that $c_i = \mathsf{T}$ for all $0 \leq i \leq n$, then let i be the least nonnegative integer such that $c_i = \mathsf{F}$. Then $i > 0$ and $c_{i-1} = \mathsf{T}$. Thus, $M(\tilde{x}s_{i-1}y[i \ldots n]) = M(\tilde{x})$ and $M(\tilde{x}s_i y[i+1 \ldots n] \neq M(\tilde{x})$, so we must have $M(\tilde{x}s_{i-1}y[i]) \neq M(\tilde{x}s_i)$, and the experiment $y[i+1 \ldots n]$ distinguishes row $s_{i-1}y[i]$ from row s_i in the progress table for $\tilde{x}$. In either case, we can find a suffix y' of y to add to the experiments $E_{\tilde{x}}$ to distinguish a new state for this progress automaton. □

For the recurrent FDFA, given $u, x, v \in \Sigma^*$ the query $\text{ENT}^u_{\text{R}}(x, v)$ is same as $\text{ENT}^u_{\text{S}}(x, v)$. The criterion for differentiating rows is more relaxed though. Given two rows x_1 and x_2 in the progress table corresponding to leading state u, the procedure $\text{DFR}^u_{\text{R}}(x_1, x_2)$ returns true if there exists an experiment v for which $T_u(x_1, v) = (\mathsf{T}, \mathsf{T})$ and $T_u(x_2, v) \neq (\mathsf{T}, \mathsf{T})$ or vice versa. The procedure Aut_{R} also declares a state x as accepting if $T_u(x, \lambda) = (\mathsf{T}, \mathsf{T})$.

Theorem 5. *When the learner L^ω is called with* ENT_1, DFR_1, Aut_1 *and* ENT^u_{R}, DFR^u_{R}, Aut_{R}, *it halts and returns a correct* FDFA *for L whose total size does not exceed that of the recurrent* FDFA *for L.*

Proof. In the case of the recurrent FDFA, again the size of the current leading automaton M cannot exceed $|\sim_L|$. If u is a state of M, then states x_1 and x_2 of the progress automaton for u are distinguished either because $M(ux_1) = M(u)$ and $M(ux_2) \neq M(u)$ or vice versa, or $M(ux_1) = M(ux_2) = M(u)$ and there exists $v \in \Sigma^*$ such that $u(x_1 v)^\omega \in L$ and $u(x_2 v)^\omega \notin L$ or vice versa. The number of such pairwise distinguished strings cannot exceed $|\sim_L| \cdot |\approx^u_{\text{P}}|$. Thus it suffices to show that each iteration of the while loop creates at least one new state in some table.

Let (x, y) be the normalized factorization of (u, v) with respect to M, and let $\tilde{x}$ be $M(x)$, where M is the current leading automaton. As in the proof of Theorem 4, consider the sequence of experiments (λ, y), $(s_1, y[2..n])$, $(s_2, y[3..n])$ up to (s_n, λ) and let (m_0, c_0), $(m_1, c_1), \ldots, (m_n, c_n)$ be the respective values of the entry query for $\tilde{x}$. We know that out of (m_0, c_0) and (m_n, c_n) one is (T, T) and the other one is not. Therefore for some i we should have that (m_i, c_i) is (T, T) and (m_{i-1}, c_{i-1}) is not, or vice versa. Thus, the experiment $y[i+1..n]$ distinguishes $s_{i-1}y[i]$ from s_i in $T_{\tilde{x}}$. □

4.4. Starting with a given leading automaton

In [11] Klarlund has shown that while the syntactic FORC is the coarsest FORC recognizing a certain language, it is not necessarily the minimal one. That is, taking a finer (bigger) leading congruence may yield smaller progress congruences. In particular, he defines a family of languages K_n over the alphabet $\Sigma_n = \Sigma_{a,n} \cup \Sigma_{b,n}$ where $\Sigma_{a,n} = \{a_1, a_2, \ldots, a_n\}$ and $\Sigma_{b,n} = \{\bar{b} \mid \bar{b} \in \{0,1\}^n\}$. The $\Sigma_{b,n}$ letters can be regarded as column vectors of length n. We use $\bar{b}[i]$ for the i-th bit in such a vector. We say that such a letter has one on its i'th track if $\bar{b}[i] = 1$. We say that a string of letters $\bar{b}_1 \bar{b}_2 \cdots \bar{b}_m$ over $\Sigma^*_{b,n}$ has k ones on its i'th track, if $\sum_{j \in [1..m]} \bar{b}_j[i] = k$. The language K_n accepts all words where at some point a_i appears infinitely often, all other a_j's stop appearing, and the number of ones in the i-th track between two occurrences of a_i is exactly n.

The DFA for $\sim_{K_n}$ has a single state, since membership in the language can be determined solely by the periodic part. The syntactic progress DFA for a leading automaton of size one is of size $\Omega(n^n)$, as it needs to track for all a_i, for $i \in [1..n]$ the number of ones in the i-th track until the next a_i. However, K_n can be recognized by an FDFA of size $O(n^2)$ by delegating some work to the leading automaton. Indeed we can use a leading automaton K that records the last a_i letter seen. Using the normalized acceptance condition, K thus tell us which a_i is read infinitely often. Then, the progress automaton for a_i only needs to count the number of ones in the i-th track (and verify that no other a_j occurs). In this case the leading automaton is of size n and so are its n progress automata, resulting in a total size of $n(n+1)$.

We can change L^ω so that it starts with a given leading automaton, and proceeds exactly as before. The resulting algorithm may end up refining the leading automaton if necessary. If we apply it to learn K_n by giving it K as the leading automaton, the learned syntactic/recurrent families would have $O(n^2)$ states as well.

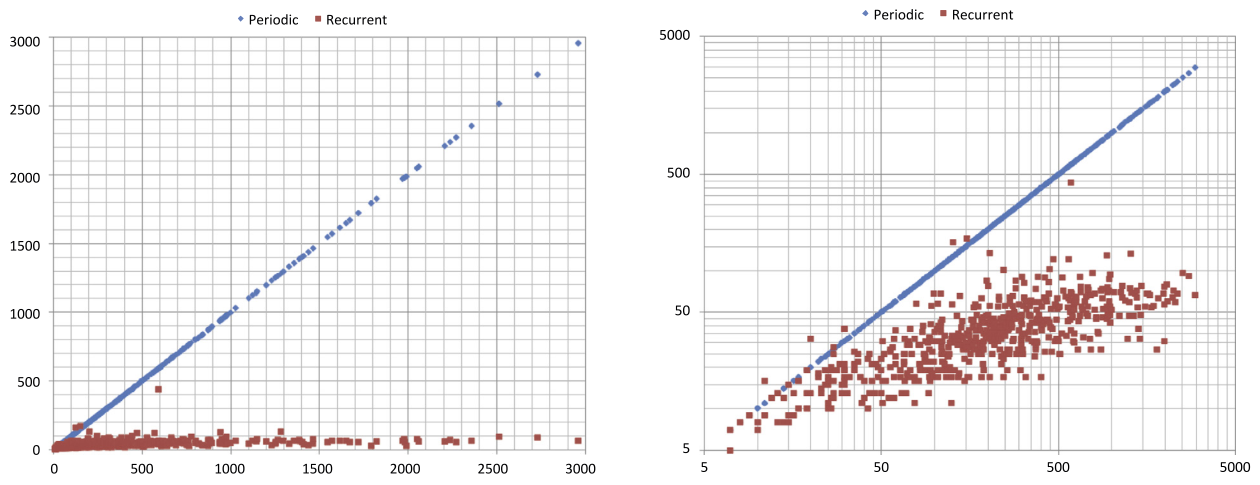

Fig. 5. The overall size of the periodic and recurrent families on randomly generated Muller automata, in linear and logarithmic scales (on the left and right, respectively). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article).

4.5. *Size of progress automata and time complexity*

In each iteration of the while loop, i.e., in processing each counterexample, at least one new state is added either to the leading automaton or to one of the progress automata. If the leading automaton is learned first, we are guaranteed that we have not distinguished more states than necessary, and so, since each operation of the while loop is polynomial in the size of the learned family, the entire procedure will run in time polynomial in the size of the learned family. However, it can be that we will unnecessarily add states to a progress automaton because the leading automaton has not been fully learned yet, in which case the progress automaton may try to learn the exact periods as does the periodic family. At a certain point the leading automaton will be exact and the size of that progress automaton will shrink as necessary. Hence the worst case time complexity for all three families is thus polynomial in the size of the periodic family, rather than the size of the learned family. In each case, the learned FDFA will not exceed the size of the corresponding periodic, syntactic or recurrent FDFA for L, though the normalized acceptance condition may permit a smaller correct FDFA to be returned.

5. Empirical results

We have implemented our algorithms and run the periodic and recurrent learning algorithms on randomly generated Muller automata over an alphabet of two letters, with five states and two sets of accepting states. On over 600 random examples, the resulting average size of a periodic automaton was 55.12 and the resulting average size of the recurrent automaton was 9.64. Fig. 5 on the left gives the size of the overall family for the recurrent family (in red) and the periodic family (in blue), where the x-axis correspond to the number of states in the resulting periodic family. Fig. 5 on the right gives the same data over a logarithmic scale.

Out of 619 randomly generated automata, on all but 6 instances the recurrent family came out smaller than the periodic family. On all but 88 instances the recurrent family was more than twice smaller than the periodic family. On 2 instances the recurrent family was more than 64 times smaller than the periodic. The following table shows the number of instances for which the bound on the ratio between the size of the recurrent and periodic families is as given by the column header. For instance, the fifth column indicates that the number of instances for which the overall size of the recurrent family was at least 8 times smaller than the overall size of the periodic family but less than 16 times smaller was 133.

0)	[0, 2)	[2, 4)	[4, 8)	[8, 16)	[16, 32)	[32, 64)	[64
6	82	110	216	133	60	10	2

6. Running example

We consider an extended example of the learning algorithm for recurrent families of automata for the particular target Muller automaton M_3 shown in Fig. 6. The counterexamples we consider were generated during one random run. Some examples of sequences accepted by M_3 are

$$b^\omega, ab^\omega, (abb)^\omega, b(bab)^\omega, (baab)^\omega.$$

Some examples of sequences rejected by M_3 are

$$a^\omega, ba^\omega, b(abb)^\omega, (bab)^\omega, (bbab)^\omega.$$

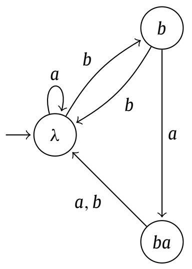

Fig. 6. The Muller automaton M_3, with one accepting state set: $\{\lambda, b\}$.

	a^ω	b^ω
λ	$-$	$+$
a	$-$	$+$
b	$-$	$+$

Fig. 7. Initial leading observation table.

	λ
λ	$-+$
a	$-+$
b	$++$
ba	$-+$
bb	$++$

Fig. 8. Initial progress observation table.

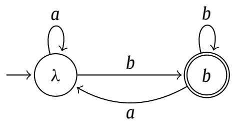

Fig. 9. Initial progress automaton.

	λ	b
λ	$-+$	$++$
a	$-+$	$-+$
b	$++$	$++$
aa	$-+$	$-+$
ab	$-+$	$++$
ba	$-+$	$-+$
bb	$++$	$++$

Fig. 10. Progress table with new experiment b.

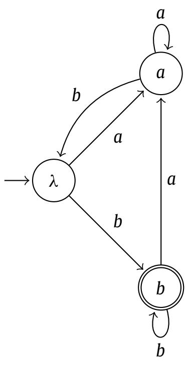

Fig. 11. Progress automaton with three states.

If we take the initial experiments for the leading automaton to be the sequences a^ω and b^ω, then the initial observation table for the leading automaton is shown in Fig. 7. In this table, the entry $+$ for row a and column b^ω indicates that M_3 accepts ab^ω, and the entry $-$ for row b and column a^ω indicates that M_3 rejects ba^ω. This table yields a one-state leading automaton.

The progress automaton for this leading automaton is constructed using just the experiment λ, which gives the observation table in Fig. 8.

In this table, the entries are pairs: the first element indicates acceptance $(+)$ or rejection $(-)$ of the corresponding sequence by M_3, and the second element indicates whether $(+)$ or not $(-)$ the corresponding string loops in the leading automaton from the state for this progress automaton back to itself. Because the current leading automaton has only one state, all the second elements of entries in this table are $+$. For example, the entry $++$ for row b and column λ has first element $+$ because M_3 accepts the sequence $\lambda \cdot (b \cdot \lambda)^\omega$, and the entry $-+$ for row ba and column λ has first element $-$ because M_3 rejects the sequence $\lambda \cdot (ba \cdot \lambda)^\omega$. From this observation table, a progress automaton with 2 states is constructed, shown in Fig. 9. An equivalence query for this family of size $(1, 2)$ returns the counterexample $(ab)^\omega$, which is rejected by M_3 but accepted by the family.

This counterexample does not change the leading automaton, but causes a new progress experiment, b, to be added to the observation table for the progress automaton, shown in Fig. 10. From this observation table, a progress automaton with 3 states (corresponding to the rows λ, a, and b) is constructed, shown in Fig. 11. An equivalence query with this new family of size $(1, 3)$ returns the counterexample $(bba)^\omega$, which is accepted by M_3 but rejected by the family.

This counterexample does not change the 1-state leading automaton, but a new experiment, a, is added to the progress table. From the new observation table (not shown), a progress automaton with 5 states is constructed, shown in Fig. 12. An equivalence query with this new family of size $(1, 5)$ yields the counterexample $(babb)^\omega$, which is rejected by M_3 but accepted by the family.

The leading automaton (still 1 state) is not changed, but the experiment bb is added to observation table for the progress automaton. From this observation table (not shown), a progress automaton with 7 states is constructed, as shown in Fig. 13. An equivalence query with the resulting family of size $(1, 7)$ returns the counterexample $b(abb)^\omega$, which is rejected by M_3 but accepted by the family.

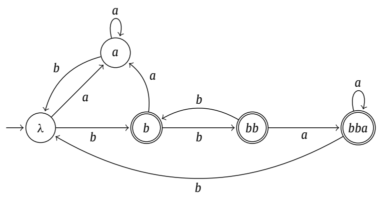

Fig. 12. Progress automaton with five states.

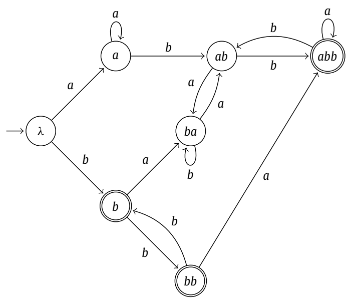

Fig. 13. Progress automaton with seven states.

	a^ω	b^ω	$(abb)^\omega$
λ	$-$	$+$	$+$
a	$-$	$+$	$+$
b	$-$	$+$	$-$
ba	$-$	$+$	$+$
bb	$-$	$+$	$+$

Fig. 14. Leading table with new experiment $(abb)^\omega$.

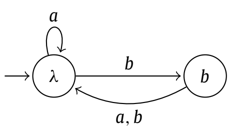

Fig. 15. The leading automaton now has two states.

	λ	b	a	bb
λ	$-+$	$+-$	$-+$	$++$
a	$-+$	$--$	$-+$	$++$
b	$+-$	$++$	$--$	$+-$
ba	$-+$	$--$	$-+$	$-+$
bb	$++$	$+-$	$++$	$++$
baa	$-+$	$+-$	$-+$	$-+$
bab	$--$	$-+$	$-+$	$--$
bba	$++$	$--$	$++$	$++$
bbb	$+-$	$++$	$-+$	$+-$

Fig. 16. Progress table for state [λ].

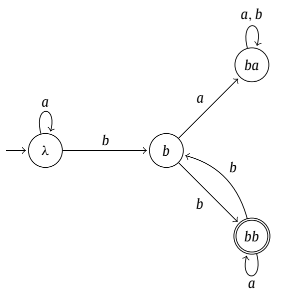

Fig. 17. Progress automaton for state [λ].

	λ	b	a	bb
λ	$-+$	$+-$	$-+$	$++$
a	$-+$	$--$	$-+$	$++$
b	$+-$	$++$	$--$	$+-$
aa	$--$	$-+$	$--$	$+-$
ab	$-+$	$--$	$--$	$-+$
ba	$--$	$--$	$--$	$--$
bb	$++$	$+-$	$--$	$++$
bba	$--$	$-+$	$+-$	$--$
bbb	$+-$	$++$	$--$	$+-$

Fig. 18. Progress table for state [b].

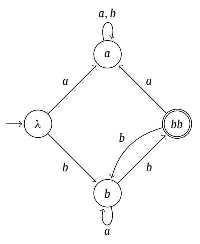

Fig. 19. Progress automaton for state [b].

This counterexample causes the experiment $(abb)^\omega$ to be added to the observation table for the leading automaton, which now appears as in Fig. 14.

From this table, a new leading automaton with 2 states is constructed, shown in Fig. 15. For each of the states, $[\lambda]$ and $[b]$, of the leading automaton an observation table and progress automaton are constructed. The observation table for $[\lambda]$ is as shown in Fig. 16. Note that the second elements of these entries are not all $+$. In constructing an automaton from the observation table, we distinguish entries $++$ from the other three possibilities, which are identified with each other. For example, the rows for b and bb are distinguished by the experiment λ because although b^ω and $(bb)^\omega$ are both accepted by M_3, the string b does not go from state $[\lambda]$ to itself in the leading automaton, while the string bb does. The progress automaton for leading state $[\lambda]$ is shown in Fig. 17.

The observation table for the state $[b]$ is shown in Fig. 18. The progress automaton for leading state $[b]$ is shown in Fig. 19. Note that the previous 7-state progress automaton for leading state $[\lambda]$ has shrunk to a 4-state progress automaton for leading state $[\lambda]$ because of the change to the leading automaton.

An equivalence query is made for this family of size $(2, 4, 4)$, and the counterexample $(bab)^\omega$ is returned, which is rejected by M_3 but accepted by the family. This causes the experiment $b(bab)^\omega$ to be added to the observation table for the leading automaton, which is shown in Fig. 20.

	a^ω	b^ω	$(abb)^\omega$	$b(bab)^\omega$
λ	$-$	$+$	$+$	$+$
a	$-$	$+$	$+$	$+$
b	$-$	$+$	$-$	$-$
ba	$-$	$+$	$+$	$-$
bb	$-$	$+$	$+$	$+$
baa	$-$	$+$	$+$	$+$
bab	$-$	$+$	$+$	$+$

Fig. 20. Leading table with new experiment $b(bab)^\omega$.

The leading automaton constructed from this table is isomorphic to the transition function of M_3, shown in Fig. 6. The previous progress automata for states $[\lambda]$ and $[b]$ are unchanged and a progress automaton of 1 state (rejecting all strings) is added for the state $[ba]$. An equivalence query with this family of size $(3, 4, 4, 1)$ shows that it is equivalent to M_3. By contrast, the equivalent exact periodic family has size $(3, 12, 16, 12)$.

7. Discussion

We provide three learning algorithms for learning an unknown regular ω-language using membership and equivalence queries. The three algorithms learn three different representations of ω-languages, using the notion of a family of DFAS: the *periodic*, *syntactic*, and *recurrent* families.

The periodic family is reminiscent of $L_\$$, a representation introduced by [5], for which a learning algorithm was given by [8]. Given finite words u and v, to answer whether $uv^\omega \in L$ using a periodic family, we check whether v is accepted by the progress automaton corresponding to u.

The syntactic family corresponds to the syntactic FORC, introduced by [15]. The syntactic family, in order to be smaller than the periodic, does not try to give a definite answer for every pair of words (u, v), only for their normalized representation. The normalization of (u, v) is the pair (uv^i, v^j) for the smallest i and $j > 0$ for which uv^i and uv^{i+j} reach the same state of the leading automaton. We show that this may lead to an exponential savings in total size over the periodic family.

The recurrent family builds on the idea of the syntactic family, but makes an additional relaxation, which can lead to a quadratic savings in total size over the syntactic family.

We have tested the algorithms on randomly generated Muller automata. In 99% of over 600 cases, the recurrent family was significantly smaller than the periodic family.

There are many interesting directions for future research. On the automata theory side, we propose finding smaller canonical representations, and comparing the sizes of the families with those of the non-canonical yet well studied representations of regular ω-languages, namely Büchi, Muller, Rabin, Streett and Parity automata. On the application side, we propose developing symbolic versions of L^ω, and finding a way to produce a good leading automaton to use as a starting point for L^ω, possibly allowing a smaller FDFA to be learned.

Acknowledgements

This is a corrected and extended version of the authors' paper "Learning Regular Omega Languages," which was presented at ALT 2014 [3]. The authors are grateful to the reviewers of ALT 2014 and TCS for their thoughtful comments on the paper.

References

[1] Rajeev Alur, Pavol Černý, P. Madhusudan, Wonhong Nam, Synthesis of interface specifications for Java classes, in: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '05, ACM, New York, NY, USA, 2005, pp. 98–109.
[2] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (2) (1987) 87–106.
[3] Dana Angluin, Dana Fisman, Learning regular omega languages, in: Peter Auer, Alexander Clark, Thomas Zeugmann, Sandra Zilles (Eds.), Algorithmic Learning Theory, Proceedings of 25th International Conference, ALT 2014, Bled, Slovenia, October 8–10, 2014, in: Lecture Notes in Computer Science, vol. 8776, Springer, 2014, pp. 125–139.
[4] A. Arnold, A syntactic congruence for rational omega-languages, Theoret. Comput. Sci. 39 (1985) 333–335.
[5] Hugues Calbrix, Maurice Nivat, Andreas Podelski, Ultimately periodic words of rational w-languages, in: Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics, Springer-Verlag, London, UK, 1994, pp. 554–566.
[6] Jamieson M. Cobleigh, Dimitra Giannakopoulou, Corina S. Păsăreanu, Learning assumptions for compositional verification, in: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 331–346.
[7] C. de la Higuera, J.-C. Janodet, Inference of [omega]-languages from prefixes, Theoret. Comput. Sci. 313 (2) (2004) 295–312.
[8] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, Bow-Yaw Wang, Extending automated compositional verification to the full class of omega-regular languages, in: C.R. Ramakrishnan, Jakob Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 4963, Springer, Berlin, Heidelberg, 2008, pp. 2–17.
[9] M. Jayasrirani, M. Humrosia Begam, D.G. Thomas, J.D. Emerald, Learning of bi-ω languages from factors, in: Proceedings of ICGI 2012, in: JMLR Workshop and Conference Proceedings, vol. 21, 2012, pp. 139–144.
[10] H. Jürgensen, G. Thierrin, On ω-languages whose syntactic monoid is trivial, Int. J. Parallel Program. 12 (5) (1983) 359–365.
[11] Nils Klarlund, A homomorphism concept for omega-regularity, in: L. Pacholski, J. Tiuryn (Eds.), Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25–30, 1994, Selected Papers, in: Lecture Notes in Computer Science, vol. 933, Springer, 1994, pp. 471–485.

[12] Martin Leucker, Learning meets verification, in: F.S. de Boer, M.M. Bonsangue, S. Graf, W.P. de Roever (Eds.), Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7–10, 2006, Revised Lectures, in: LNCS, vol. 4709, Springer, 2006, pp. 127–151.
[13] R. Lindner, L. Staiger, Eine Bemerkung über nichtkonstantenfreie sequentielle operatoren, Elektron. Inf.verarb. Kybern. 10 (4) (1974) 195–202.
[14] O. Maler, A. Pnueli, On the learnability of infinitary regular sets, Inform. and Comput. 118 (2) (1995) 316–326.
[15] O. Maler, L. Staiger, On syntactic congruences for omega-languages, Theoret. Comput. Sci. 183 (1) (1997) 93–112.
[16] Robert McNaughton, Testing and generating infinite sequences by a finite automaton, Inf. Control 9 (5) (1966) 521–530.
[17] M. Michel, Complementation is much more difficult with automata on infinite words, Manuscript, CNET, 1988.
[18] Wonhong Nam, P. Madhusudan, Rajeev Alur, Automatic symbolic compositional verification by learning assumptions, Form. Methods Syst. Des. 32 (3) (2008) 207–234.
[19] C.F. Pfleeger, State reduction in incompletely specified finite-state machines, IEEE Trans. Comput. 22 (12) (1973) 1099–1102.
[20] A. Saoudi, T. Yokomori, Learning local and recognizable omega-languages and monadic logic programs, in: EUROCOLT, in: LNCS, vol. 1121, Springer-Verlag, 1993, pp. 50–59.
[21] L. Staiger, Finite-state omega-languages, J. Comput. System Sci. 27 (3) (1983) 434–448.
[22] B. Trakhtenbrot, Finite automata and monadic second order logic, Sib. Math. J. (1962) 103–131.