Learning Regular Omega Languages

Dana Angluin[1]* and Dana Fisman[2]**

[1] Yale University
[2] University of Pennsylvania

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 omega regular languages, using the notion of families of DFAs. One is of size similar to $L_\$$, a DFA representation recently learned using L^* [7]. The second is based on the syntactic FORC, introduced in [14]. The third is introduced herein. We show that the second can be exponentially smaller than the first, and the third is at most as large as the first two, with up to a quadratic saving with respect to the second.

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 reducing 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

* Research of this author was supported by US NSF grant CCF-0916389.
** Research of this author was supported by US NSF grant CCF-1138996.

uses L^* [5,1,17] and a model checker for the different component playing the role of the teacher. Using L^* allows learning only *safety* properties (a subset of ω-regular properties that state that something bad hasn't happened and 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 [11].

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 [15] 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 [6] 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 any extension of u is not in L. This work is extended in [8] to learning bi-ω languages from subwords.

Saoudi and Yokomori [19] 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* [19] if it is recognizable by a deterministic automaton all of whose states are accepting.

Maler and Pnueli [13] 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 [20]. 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 [10,14] 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 [4]. 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 counter examples 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. [7]. 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 [14] 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 size $O(n)$ 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 as both the syntactic FDFA and an FDFA corresponding to $L_\$$, 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 $\{i, i+1, \ldots, j\}$. 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 $v = a_1 a_2 \ldots a_n$ is a sequence of states $q_0, q_1, \ldots, q_n$ 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 $L_1 = \Sigma^* a^\omega$. We have that $\sim_{L_1}$ consists of just one equivalence class, but obviously an acceptor recognizing L_1 needs more than a single state.

3 Canonical Representations of Regular ω-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. 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.

Second Canonical Representation - Syntactic FORC Searching for a notion of right congruence adequate for regular ω-languages was the subject of many works (c.f. [21,12,9,3,14]). In the latest of these [14] Maler and Staiger proposed the notion of a *family of right-congruences* or FORC.

Definition 1 (FORC, Recognition by FORC [14]). *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* $u, x, y \in \Sigma^*$.

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* $uv \sim_L u$.[1]

Definition 2 (Syntactic FORC [14]). *Let* $x, y, u \in \Sigma^*$*, and* L *be a* ω*-language. We use* $x \approx^u_{\mathrm{S}} y$ *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 an* ω*-language* L *is* $(\sim_L, \{\approx^u_{\mathrm{S}}\}_{u\in[\sim_L]})$.

Theorem 1 (Minimality of the Syntactic FORC [14]). *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.*

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* DFA*s* $\mathcal{F} = (M, \{A_q\})$ *over an alphabet* Σ *consists of a leading automaton* $M = (\Sigma, Q, q^0, \delta)$ *and progress* DFA*s* $A_q = (\Sigma, S_q, s^0_q, \delta_q, F_q)$ *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.[2]

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^u_{S} *be the* DFA*s corresponding to* $\approx^u_{\mathrm{S}}$*, where the accepting states are the equivalence classes* $[v]$ *of* $\approx^u_{\mathrm{S}}$ *for which* $uv \sim_L u$ *and* $uv^\omega \in L$*. We use* $\mathcal{F}_{\mathrm{S}}$ *to denote the* FDFA $(M, \{A^u_{\mathrm{S}}\})$*, and refer to it as the* syntactic FDFA.[3]

[1] 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 [14] that for finite FORCs, covering and saturation coincide. Thus, the definition here only requires that L is covered by $\mathcal{R}$.

[2] In [10] Klarlund also suggested the notion of family of DFAs. However, that work did require the third condition in the definition of FORC to hold.

[3] The syntactic FDFA is well defined since, as shown in [14], $uv^\omega \in L$ implies $[u][v]^\omega \subseteq L$.

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

Proposition 1. *Let L be an ω-language and $\mathcal{F}_{\mathrm{S}} = (M, \{A^u_{\mathrm{S}}\})$ 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 $v \in [\![A^{\tilde{u}}_{\mathrm{S}}]\!]$ where $\tilde{u} = M(u)$.*

To get an understanding of the subtleties in the definition of $\approx^u_{\mathrm{S}}$ 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 $x \approx^u_{\mathrm{P}} y$ 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^u_{P} be the* DFA *corresponding to $\approx^u_{\mathrm{P}}$ where the accepting states are the equivalence classes $[v]$ of $\approx^u_{\mathrm{S}}$ for which $uv^\omega \in L$. We use $\mathcal{F}_{\mathrm{P}}$ to denote the* FDFA *$(M, \{A^u_{\mathrm{P}}\})$, and refer to it as the* periodic FDFA.[4]

It is not hard to see that the following proposition holds.

Proposition 2. *Let L be a regular ω-language and $\mathcal{F}_{\mathrm{P}} = (M, \{A^u_{\mathrm{S}}\})$ the periodic* 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 $v \in [\![A^{\tilde{u}}_{\mathrm{P}}]\!]$ where $\tilde{u} = M(u)$.*

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 it can be factorially more succinct than the periodic FORC, and the same arguments can be used to show that the syntactic FORC is 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, if it accepts some repetition of it. For instance, take $L_2 = (aba + bab)^\omega$. Then A^λ_{P} accepts ab as this is a valid period, yet A^λ_{S} rejects it, since $\lambda \not\sim_L ab$ but it does accept its 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 $\mathcal{F}_{\mathrm{P}}$. And as the following example shows, it can make a very big difference.

Theorem 2. *There exists a family of languages L_n whose syntactic* FDFA *has $O(n)$ states but the 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. 1 on the left.[5] 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 $\perp$ is the trivial DFA accepting the empty language.

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

[5] The automata for the languages L_n are the deterministic version of the automata for a family M_n introduced by Michel [16] to prove there exists an NBA with $O(n)$ states whose complement cannot be recognized by an NBA with fewer than $n!$ states.

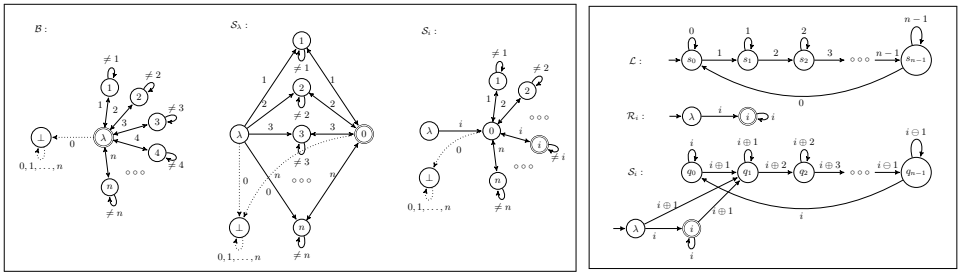

Fig. 1. On the left, Büchi automaton $\mathcal{B}$ for L_n, and the syntactic FDFA for L_n. On the right, leading automaton $\mathcal{L}$ for T_n, the syntactic and recurrent progress automaton for i, $\mathcal{S}_i$ and $\mathcal{R}_i$.

We now show 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 $f(i) = \delta(i, v)$. 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 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.

We now show that we can associate with each π the promised word v_π. With $\pi_0 = (0\ 1\ \dots n)$ 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 $\pi = (i_0\ i_1 \dots\ i_n)$ and $\pi' = (i'_0\ i'_1 \dots\ i'_n)$ 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 $f_{v_\pi} = \pi$.

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	$i_0 = j_0 \neq 0,\ i_k = k$	$i_0 k j_k 0 j_k$
8	$i_0 = j_0 \neq 0,\ j_k = k$	$i_0 k i_k 0 i_k$
9	$i_0 = j_0 \neq 0,\ i_k \neq k,\ j_k \neq k$	$i_0 k j_k 0 j_k i_j$

Table 1. Distinguishing word y.

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

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. [4] 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 an ω-language and $\mathcal{F}_{\mathrm{P}}$ and $\mathcal{F}_{\mathrm{S}}$ the corresponding periodic and syntactic* FDFA*s. Then $[\![\mathcal{F}_{\mathrm{P}}]\!]_{\mathbb{E}}$ is saturated.*

The language $[\![\mathcal{F}_{\mathrm{S}}]\!]_{\mathbb{E}}$ on the other hand, is not necessarily saturated. Consider $L_3 = a^\omega + ab^\omega$. 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 in finding a repetition of it v^k, starting from some u so that reading uv on the leading automaton takes us back to the state we got to after reading u.

Given a right congruence $\sim$ of finite index and a 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 $0 \leq i < j$ 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 (x, y) with respect to $\sim_M$.*

Proposition 4. *Let L be an 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. We show that given an ultimately periodic word w, for any x, y such that $w = xy^\omega$, $(x, y) \in [\![\mathcal{F}_{\text{S}}]\!]_{\mathbb{N}}$ iff $w \in L$. This shows that $[\![\mathcal{F}_{\text{S}}]\!]_{\mathbb{N}}$ is a saturated acceptor of L. Assume towards contradiction that $w = uv^\omega$, $w \in L$ yet $(u, v) \notin [\![\mathcal{F}_{\text{S}}]\!]_{\mathbb{N}}$. Let (x, y) be the normalized factorization of (u, v) with respect to $\sim_L$. We have $xy \sim_L x$, $x = uv^i$ and $y = v^j$ for some i, j. Let $\tilde{x}$ be $M(xy^j)$. Let $\tilde{y} = A_{\tilde{x}}(y)$. Thus $\tilde{y}$ is not an accepting state. Meaning $\tilde{x}\,\tilde{y}^\omega \notin L$.

On the other hand we have that $uv^\omega \in L$ and $uv^\omega = uv^iv^\omega = xv^\omega$. Since $\tilde{x} \sim_L x$ we get that $\tilde{x}y^\omega \in L$ and since $y^\omega = (y^j)^\omega$ that $\tilde{x}(y^j)^\omega \in L$. Since $y \approx^{\tilde{x}}_{\text{S}} \tilde{y}$ and $\tilde{x}y \sim_L \tilde{x}$ it follows that $\tilde{x}\tilde{y}^\omega \notin L$. Contradiction.

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

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 $\approx^u_{\text{S}}$'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 $x \approx^u_{\text{R}} y$ 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 $\mathcal{F}_{\text{R}}$ to denote the* FDFA *$(M, \{A^u_{\text{R}}\})$ 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 $\mathcal{F}_{\text{R}}$ as the* recurrent FDFA*.*

Note that the proof of Proposition 4 did not make use of the additional requirement $ux \sim_L uy$ of $\approx^u_{\text{S}}$. 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 $\mathcal{F}_{\text{R}} = (M, \{A^u_{\text{R}}\})$ be its recurrent* FDFA*. Then $[\![\mathcal{F}_{\text{R}}]\!]_{\mathbb{N}}$ is saturated and is an acceptor of L.*

It follows from the definitions of $\approx^u_{\text{S}}$ and $\approx^u_{\text{R}}$ and $\approx^u_{\text{P}}$ that (a) $\approx^u_{\text{P}}$ refines $\approx^u_{\text{R}}$ (b) $\approx^u_{\text{S}}$ refines $\approx^u_{\text{R}}$ and (c) if $|\sim_L| = n$ and $|\approx^u_{\text{R}}| = m$ then $\approx^u_{\text{S}}$ is of size at most nm. 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.

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

Proof. Consider the alphabet $\Sigma_n = \{a_0, a_1, \ldots, a_{n-1}\}$. Let L_i abbreviate a_i^+. Let U_i be the set of ultimately periodic words $(L_0L_1 \ldots L_{n-1})^*(L_0L_1 \ldots L_i)a_i^\omega$. Finally let T_n be the union $U_0 \cup U_1 \cup \ldots U_{n-1}$. In Figure 1 on the right we show its leading DFA and the syntactic and recurrent progress DFAs for state i. (The sink state 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 $(n + 1) + n(n + 3) + (n + 1)$. □

Algorithm 1: The Learner L^ω

Initialize the leading table $\mathcal{T} = (S, \tilde{S}, E, T)$ with $S = \tilde{S} = \{\lambda\}$ and $E = \{(\lambda, \lambda)\}$.
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 the equivalence query $\mathcal{H} = (M, \{A_u\})$.
while $a =$ *"no"* **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 $\mathcal{H}$

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 [18]. The recurrent FDFA chooses to treat all don't cares as rejecting.

4 Learning ω-regular Languages via Families of DFAs

In the previous section we have provided three canonical representations of regular ω-languages as families of DFAs. 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, given a pair (u, v) answers whether $uv^\omega \in L$, and the equivalence oracle answers whether a given FDFA $\mathcal{F}$, satisfies $\llbracket \mathcal{F} \rrbracket_{\mathbb{N}} = L$ 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 $(L \setminus \llbracket \mathcal{F} \rrbracket_{\mathbb{N}}) \cup (\llbracket \mathcal{F} \rrbracket_{\mathbb{N}} \setminus L)$.

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 Alg. 1. This is a simple 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.

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 and DFR_2. (This is where the algorithms for the different families differ.)

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.

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. The algorithm then repeats the following loop (lines 7-18) until the equivalence query returns "yes".

If the equivalence query returns a counter example (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 is correct. We discuss the time complexity at the end of the section.

Specializing for $\mathcal{F}_\mathbf{p}$, $\mathcal{F}_\mathbf{s}$ and $\mathcal{F}_\mathbf{r}$ We now turn to 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 different learning algorithms differ in the content they put in the progress tables (i.e. procedure ENT_2), in the criterion for differentiating rows in a progress table (i.e. procedure DFR_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 correctness.

For learning the leading automaton, which is same in all 3 families, the following procedures: ENT_1, DFR_1 and Aut_1 are used. For $u \in \Sigma^*$ and $xy^\omega \in \Sigma^\omega$ the procedure $\text{ENT}_1(u, xy^\omega)$ returns 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 $w \in E$ s.t. $T(u_1, w) \neq T(u_2, w)$. 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 $\text{ENT}_\text{P}^u(x, v) = \mathsf{T}$ iff $u(xv)^\omega \in L$, and $\text{DFR}_\text{P}(x_1, x_2)$ is simply $\exists v \in E$ s.t. $T(x_1, v) \neq T(x_2, v)$. The procedure Aut_p declares a state x as accepting if $T(x, \lambda) = \mathsf{T}$.

Theorem 3. *Calling the learner L^ω with $\text{ENT}_1, \text{DFR}_1, Aut_1$ and $\text{ENT}_\text{P}, \text{DFR}_\text{P}, Aut_p$ halts and returns the periodic* FDFA.

Proof. We need to show that in each iteration of the while loop at least one state is added to one of the tables. Suppose the returned counter example 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 $\tilde{x} = M(x)$. 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 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 same answers, we look for an experiment that will distinguish a new state in the progress table of $\tilde{x}$. Let $\tilde{y} = M_{\tilde{x}}(y)$. 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. Thus $s_n = \tilde{y}$. Consider the sequence (λ, y), $(s_1, y[2..n])$, $(s_2, y[3..n])$, up to (s_n, λ). Then we know the entry for first and last return different results, though they should not. 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 $\tilde{x}$. □

For the syntactic FDFA, given $u, x, v \in \Sigma^*$, the procedure $\text{ENT}_\text{S}^u(x, v)$ returns a pair $(m, c) \in \{\mathsf{T}, \mathsf{F}\} \times \{\mathsf{T}, \mathsf{F}\}$ such that $m = \mathsf{T}$ iff $u(xv)^\omega \in L$ and $c = \mathsf{T}$ iff

$uxv \sim_L u$. Given two rows x_1 and x_2 in the progress table corresponding to leading state u, the procedure $\text{DFR}_\text{S}^u(x_1, x_2)$ returns true if either $M(x_1) \neq M(x_2)$, or there exists an experiment $v \in E_u$ for which $T(u_1, v) = (m_1, c_1)$, $T(u_2, v) = (m_2, c_2)$ and $(c_1 \vee c_2) \wedge (m_1 \neq m_2)$. The procedure Aut_s declares a state x as accepting if $T(x, \lambda) = (\mathsf{T}, \mathsf{T})$.

Theorem 4. *Calling the learner L^ω with $\text{ENT}_1, \text{DFR}_1, Aut_1$ and $\text{ENT}_\text{S}^u, \text{DFR}_\text{S}^u, Aut_s$ halts and returns the syntactic* FDFA.

Proof. Again we need to show that each iteration of the while loop creates a state. The proof for the first part is same as in Thm. 3. For the second part, let (x, y) be the normalized factorization of the given counter example w.r.t to current leading automaton. We can consider the sequence of experiments (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 the membership query does not guarantee they would get distinguished, as this fact might be hidden if for both $M(ux_iv) \neq M(u)$. Let (m_0, c_0), $(m_1, c_1), \ldots, (m_n, c_n)$ be the respective results for the entry query. We know that $m_0 \neq m_n$. Also, we know that $c_0 = \mathsf{T}$ since we chose x and y so that $M(xy) = M(x)$. Let i be the smallest for which $m_{i-1} \neq m_i$. If all the c_j's for $j \leq i$ are T, we can find a distinguishing experiment as in the case of periodic FDFA. Otherwise let k be the smallest for which $c_k = \mathsf{F}$. Then $M(xs_{k-1}y[k..n]) = M(x)$ but $M(xs_ky[k+1..n]) \neq M(x)$. Therefore $y[k+1..n]$ distinguishes $s_{k-1}y[k]$ from s_k and so we add it to the experiments $E_{\tilde{x}}$ of the progress table for $\tilde{x}$. □

For the recurrent FDFA, given $u, x, v \in \Sigma^*$ the query $\text{ENT}_\text{R}^u(x, v)$ is same as $\text{ENT}_\text{S}^u(x, v)$. 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 $\text{DFR}_\text{R}^u(x_1, x_2)$ returns true if there exists an experiment v for which $T(x_1, v) = (\mathsf{T}, \mathsf{T})$ and $T(x_2, v) \neq (\mathsf{T}, \mathsf{T})$ or vice versa. The procedure Aut_r also declares a state x as accepting if $T(x, \lambda) = (\mathsf{T}, \mathsf{T})$.

Theorem 5. *Calling the learner L^ω with $\text{ENT}_1, \text{DFR}_1, Aut_1$ and $\text{ENT}_\text{R}^u, \text{DFR}_\text{R}^u, Aut_r$ halts and returns the recurrent* FDFA.

Proof. The first part is same as in the proof of Theorem 3. For the second part, let (x, y) be the normalized factorization of (u, v) with respect to M. Let $\tilde{x}$ be $M(x)$ and let $\tilde{y}$ be $A_{\tilde{x}}(\tilde{y})$. 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 the respective entries (m_0, c_0), $(m_1, c_1), \ldots, (m_n, c_n)$ in the table $T_{\tilde{x}}$. 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. □

Starting with a Given Leading Automaton In [10] 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 showed a

family of languages K_n where $|\sim_{K_n}| = 1$, and its syntactic progress DFA is of size $O(n^n)$, but it has an FDFA with n states in the leading automaton and n states in each of the progress DFAs — thus the total size is $O(n^2)$. The family K_n over the alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\} \cup \{\bar{b} \mid \bar{b} \in \{0,1\}^n\}$ accepts all words where at some point a_i appears infinitely often, all other a_j's stop appearing, and the number of 1's in the i-th track between two occurrences of a_i is exactly n. It can be seen that the recurrent FDFA will also have $O(n^n)$ states. The family that has a total of $O(n)$ states has a leading automaton K with n states, remembering which letter among the a_i's was the last to occur.

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 learnt syntactic/recurrent families would have $O(n^2)$ states as well.

Time Complexity In each iteration of the while loop, i.e. in processing each counter example, 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, since 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. But the worse 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.

5 Conclusions

We provide an algorithm for learning an unknown regular ω-language, using membership and equivalence queries. The learning algorithm L^ω uses the notion of a family of DFAs (FDFA), with which we can represent and learn three different canonical representations: the periodic, syntactic and recurrent FDFAs. The periodic FDFA corresponds to $L_\$$ [4], a representation learned via L^* in [7]. The syntactic FDFA corresponds to the notion of syntatic family of right congruences, introduced in [14]. The recurrent FDFA is introduced herein. We have compared the sizes of the families, and have shown that the recurrent FDFA, is the smallest among these. The learning algorithm running time for the 3 versions is polynomial in the size of the periodic family. For future work we hope to be able to provide a learning algorithm polynomial in the size of the learned family.

Acknowledgment We would like to thank Oded Maler and Ludwig Staiger for referring us to [4], and Daniel Neider for referring us to [7].

References

1. R. Alur, P. Cerný, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. In *POPL*, pages 98–109, 2005.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
3. A. Arnold. A syntactic congruence for rational omega-languages. *Theor. Comput. Sci.*, 39:333–335, 1985.
4. H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational w-languages. In S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, *MFPS*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993.
5. J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.
6. C. de la Higuera and J-C Janodet. Inference of [omega]-languages from prefixes. *Theor. Comput. Sci.*, 313(2):295–312, 2004.
7. A. Farzan, Y-F. Chen, E.M. Clarke, Y-K. Tsay, and B-Y. Wang. Extending automated compositional verification to the full class of omega-regular languages. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008.
8. M. Jayasrirani, M. Humrosia Begam, D. G. Thomas, and J. D. Emerald. Learning of bi-languages from factors. *Journal of Machine Learning Research - Proceedings Track*, 21:139–144, 2012.
9. H. Jürgensen and G. Thierrin. On-languages whose syntactic monoid is trivial. *International Journal of Parallel Programming*, 12(5):359–365, 1983.
10. N. Klarlund. A homomorphism concept for omega-regularity. In L. Pacholski and J. Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1994.
11. M. Leucker. Learning meets verification. In *FMCO*, pages 127–151, 2006.
12. R. Lindner and L. Staiger. Eine bemerkung über nichtkonstantenfreie sequentielle operatoren. *Elektronische Informationsverarbeitung und Kybernetik*, 10(4):195–202, 1974.
13. O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995.
14. O. Maler and L. Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997.
15. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.
16. M. Michel. Complementation is much more difficult with automata on infinite words. In *Manuscript, CNET*, 1988.
17. W. Nam, P. Madhusudan, and R. Alur. Automatic symbolic compositional verification by learning assumptions. *FMSD*, 32(3):207–234, 2008.
18. C.F. Pfleeger. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, 22(12):1099–1102, 1973.
19. A. Saoudi and T. Yokomori. Learning local and recognizable omega-languages and monadic logic programs. In *EUROCOLT*, volume 1121 of *LNCS*, pages 50–59. Springer-Verlag, 1993.
20. L. Staiger. Finite-state omega-languages. *J. Comput. Syst. Sci.*, 27(3):434–448, 1983.
21. B. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math J.*, pages 103–131, 1962.