Families of DFAs as Acceptors of $\omega$-Regular Languages

Dana Angluin$^1$, Udi Boker$^2$, and Dana Fisman$^3$

1 Yale University, New Haven, CT, USA
2 The Interdisciplinary Center (IDC), Herzliya, Israel
3 University of Pennsylvania, Philadelphia, PA, USA

24th July 2016

Abstract

Families of DFAs (FDFAs) provide an alternative formalism for recognizing $\omega$-regular languages. The motivation for introducing them was a desired correlation between the automaton states and right congruence relations, in a manner similar to the Myhill-Nerode theorem for regular languages. This correlation is beneficial for learning algorithms, and indeed it was recently shown that $\omega$-regular languages can be learned from membership and equivalence queries, using FDFAs as the acceptors.

In this paper, we look into the question of how suitable FDFAs are for defining $\omega$-regular languages. Specifically, we look into the complexity of performing Boolean operations, such as complementation and intersection, on FDFAs, the complexity of solving decision problems, such as emptiness and language containment, and the succinctness of FDFAs compared to standard deterministic and nondeterministic $\omega$-automata.

We show that FDFAs enjoy the benefits of deterministic automata with respect to Boolean operations and decision problems. Namely, they can all be performed in nondeterministic logarithmic space. We provide polynomial translations of deterministic Büchi and co-Büchi automata to FDFAs and of FDFAs to nondeterministic Büchi automata (NBAs). We show that translation of an NBA to an FDFA may involve an exponential blowup. Last, we show that FDFAs are more succinct than deterministic parity automata (DPAs) in the sense that translating a DPA to an FDFA can always be done with only a polynomial increase, yet the other direction involves an inevitable exponential blowup in the worst case.

1 Introduction

The theory of finite-state automata processing infinite words was developed in the early sixties, starting with Büchi [3] and Muller [13], and motivated by problems in logic and switching theory. Today, automata for infinite words are extensively used in verification and synthesis of reactive systems, such as operating systems and communication protocols.

An automaton processing finite words makes its decision according to the last visited state. On infinite words, Büchi defined that a run is accepting if it visits a designated set of states infinitely often. Since then several other accepting conditions were defined, giving rise to various $\omega$-automata, among which are Muller, Rabin, Streett and parity automata.

The theory of $\omega$-regular languages is more involved than that of finite words. This was first evidenced by Büchi’s observation that nondeterministic Büchi automata are more expressive than their deterministic counterpart. While for some types of $\omega$-automata the nondeterministic and deterministic variants have the same expressive power, none of them...
Families of DFAs as Acceptors of $\omega$-Regular Languages

possesses all the nice qualities of acceptors for finite words. In particular, none has a corresponding Myhill-Nerode theorem [16], i.e. a direct correlation between the states of the automaton and the equivalence classes corresponding to the canonical right congruence of the recognized language.

The absence of a Myhill-Nerode like property in $\omega$-automata has been a major drawback in obtaining learning algorithms for $\omega$-regular languages, a question that has received much attention lately due to applications in verification and synthesis, such as black-box checking [17], assume-guarantee reasoning [14], error localization [5], regular model checking [15] and more. The reason is that learning algorithms typically build on this correspondence between the automaton and the right congruence.

Recently, two algorithms for learning an unknown $\omega$-regular language were proposed, both using non-conventional acceptors. One uses a reduction due to [4] named $L_\omega$-automata of $\omega$-regular languages to regular languages [6], and the other uses a representation termed families of DFAs [1]. Both representations are founded on the following well known property of $\omega$-regular languages: two $\omega$-regular languages are equivalent iff they agree on the set of ultimately periodic words. An ultimately periodic word $uv^\omega$, where $u \in \Sigma^*$ and $v \in \Sigma^+$, can be represented as a pair of finite words $(u,v)$. Both $L_\omega$-automata and families of DFAs process such pairs and interpret them as the corresponding ultimately periodic words. Families of DFAs have been shown to be up to exponentially more succinct than $L_\omega$-automata [1].

A family of DFAs (FDFA) is composed of a leading automaton $Q$ with no accepting states and for each state $q$ of $Q$, a progress DFA $P_q$. Intuitively, the leading automaton is responsible for processing the non-periodic part $u$, and depending on the state $q$ reached when $Q$ terminated processing $u$, the respective progress DFA $P_q$ processes the periodic part $v$, and determines whether the pair $(u,v)$, which corresponds to $uv^\omega$, is accepted. (The exact definition is more subtle and is provided in Section 3.) If the leading automaton has $n$ states and the size of the maximal progress DFA is $k$, we say that the FDFA is of size $(n,k)$. An earlier definition of FDFA, given in [9], provided a machine model for the families of right congruences of [10]. They were redefined in [1], where their acceptance criterion was adjusted, and their size was reduced by up to a quadratic factor. We follow the definition of [1].

In order for an FDFA to properly characterize an $\omega$-regular language, it must satisfy the saturation property: considering two pairs $(u,v)$ and $(u',v')$, if $uv^\omega = u'v'^\omega$ then either both $(u,v)$ and $(u',v')$ are accepted or both are rejected (cf. [4, 20]). Saturated FDFA are shown to exactly characterize the set of $\omega$-regular languages. Saturation is a semantic property, and the check of whether a given FDFA is saturated is shown to be in PSPACE. Luckily, the FDFA that result from the learning algorithm of [1] are guaranteed to be saturated.

Saturated FDFA bring an interesting potential – they have a Myhill-Nerode like property, and while they are “mostly” deterministic, a nondeterministic aspect is hidden in the separation of the prefix and period parts of an ultimately periodic infinite word. This gives rise to the natural questions of how “dominant” are the determinism and nondeterminism in FDFA, and how “good” are they for representing $\omega$-regular languages. These abstract questions translate to concrete questions that concern the succinctness of FDFA and the complexity of solving their decision problems, as these measures play a key role in the

---

1 Another related formalism is of Wilke algebras [26, 27], which are two-sorted algebras equipped with several operations. An $\omega$-language over $\Sigma^\omega$ is $\omega$-regular if and only if there exists a two-sorted morphism from $\Sigma^\omega$ into a finite Wilke structure [26]. A central difference between the FDFA theory and the algebraic theory of recognition by monoids, semigroups, $\omega$-semigroups, and Wilke structures is that the former relates to right-congruences, while the latter is based on two-sided congruences.
usefulness of applications built on top of them.

Our purpose in this paper is to analyze the F DFA formalism and answer these questions. Specifically, we ask: What is the complexity of performing the Boolean operations of complementation, union, and intersection on saturated FDFAs? What is the complexity of solving the decision problems of membership, emptiness, universality, equality, and language containment for saturated FDFAs? How succinct are saturated FDFAs, compared to deterministic and nondeterministic \(\omega\)-automata?

We show that saturated FDFAs enjoy the benefits of deterministic automata with respect to Boolean operations and decision functions. Namely, the Boolean operations can be performed in logarithmic space, and the decision problems can be solved in nondeterministic logarithmic space. The constructions and algorithms that we use extend their counterparts on standard DFAs. In particular, complementation of saturated FDFAs can be obtained on the same structure, and union and intersection is done on a product of the two given structures. The correctness proof of the latter is a bit subtle.

As for the succinctness, which turns out to be more involved, we show that saturated FDFAs properly lie in between deterministic and nondeterministic \(\omega\)-automata. We provide polynomial translations from deterministic \(\omega\)-automata to FDFAs and from FDFAs to nondeterministic \(\omega\)-automata, and show that an exponential state blowup in the opposite directions is inevitable in the worst case.

Specifically, a saturated F DFA of size \((n,k)\) can always be transformed into an equivalent nondeterministic Büchi automaton (NBA) with \(O(n^2k^3)\) states. As for the other direction, transforming an NBA with \(n\) states to an equivalent F DFA is shown to be in \(2^{\Theta(n \log n)}\). This is not surprising since, as shown by Michel [12], complementing an NBA involves a \(2^{\Omega(n \log n)}\) state blowup, while F DFA complementation requires no state blowup.

Considering deterministic \(\omega\)-automata, a Büchi or co-Büchi automaton (DBA or DCA) with \(n\) states can be transformed into an equivalent F DFA of size \((n,2n)\), and a deterministic parity automaton (DPA) with \(n\) states and \(k\) colors can be transformed into an equivalent F DFA of size \((n,kn)\). As for the other direction, since DBA and DCA do not recognize all the \(\omega\)-regular languages, while saturated FDFAs do, a transformation from an F DFA to a DBA or DCA need not exist. Comparing FDFAs to DFAs, which do recognize all \(\omega\)-regular languages, we get that FDFAs can be exponentially more succinct: We show a family of languages \(\{L_n\}_{n \geq 1}\), such that for every \(n\), there exists an F DFA of size \((n+1,n^2)\) for \(L_n\), but any DPA recognizing \(L_n\) must have at least \(2^{n-1}\) states. (A deterministic Rabin or Streett automaton for \(L_n\) is also shown to be exponential in \(n\), requiring at least \(2^{2^n}\) states.)

## 2 Preliminaries

An **alphabet** \(\Sigma\) is a finite set of symbols. The set of finite words over \(\Sigma\) is denoted by \(\Sigma^*\), and the set of infinite words, termed \(\omega\)-words, over \(\Sigma\) is denoted by \(\Sigma^\omega\). As usual, we use \(x^*, x^+, \) and \(x^\omega\) to denote finite, non-empty finite, and infinite concatenations of \(x\), respectively, where \(x\) can be a symbol, a finite word, or a language. We use \(\epsilon\) for the empty word and \(\Sigma^+\) for \(\Sigma^* \backslash \{\epsilon\}\). An infinite word \(w\) is **ultimately periodic** if there are two finite words \(u \in \Sigma^*\) and \(v \in \Sigma^+\), such that \(w = uv^\omega\). A **language** is a set of finite words, that is, a subset of \(\Sigma^*\), while an **\(\omega\)-language** is a set of \(\omega\)-words, that is, a subset of \(\Sigma^\omega\). For natural numbers \(i, j\) and a word \(w\), we use \([i..j]\) for the set \(\{i, i+1, \ldots, j\}\), \(w[i]\) for the \(i\)-th letter of \(w\), and \(w[i..j]\) for the subword of \(w\) starting at the \(i\)-th letter and ending at the \(j\)-th letter, inclusive.

An **automaton** is a tuple \(A = \langle \Sigma, Q, \epsilon, \delta \rangle\) consisting of an alphabet \(\Sigma\), a finite set \(Q\) of states, an initial state \(\epsilon \in Q\), and a transition function \(\delta : Q \times \Sigma \rightarrow 2^Q\). A run of an
Families of DFAs as Acceptors of \( \omega \)-Regular Languages

A deterministic finite automaton on a finite word \( v = a_1a_2 \ldots a_n \) is a sequence of states \( q_0, q_1, \ldots, q_n \) such that \( q_0 = \iota \), and for each \( i \geq 0 \), \( q_{i+1} \in \delta(q_i, a_i) \). A run on an infinite word is defined similarly and results in an infinite sequence of states. The transition function is naturally extended to a function \( \delta : Q \times \Sigma^* \to 2^Q \); by defining \( \delta(q, \epsilon) = \{ q \} \), and \( \delta(q, av) = \cup_{p \in \delta(q,a)} \delta(p, v) \) for \( q \in Q \), \( a \in \Sigma \), and \( v \in \Sigma^* \). We often use \( A(v) \) as a shorthand for \( \delta(\iota, v) \) and \( |A| \) for the number of states in \( Q \). We use \( A^q \) to denote the automaton \( \langle \Sigma, Q, q, \delta \rangle \) obtained from \( A \) by replacing the initial state with \( q \). We say that \( A \) is deterministic if \( |\delta(q, a)| \leq 1 \) and complete if \( |\delta(q, a)| \geq 1 \), for every \( q \in Q \) and \( a \in \Sigma \). For simplicity, we consider all automata to be complete. (As is known, every automaton can be linearly translated to an equivalent complete automaton.)

By augmenting an automaton with an acceptance condition \( \alpha \), thereby obtaining a tuple \( \langle \Sigma, Q, q, \delta, \alpha \rangle \), we get an acceptor, a machine that accepts some words and rejects others. An acceptor accepts a word if at least one of the runs on that word is accepting. For finite words the acceptance condition is a set \( F \subseteq Q \) of accepting states, and a run on a word \( v \) is accepting if it ends in an accepting state, i.e., if \( \delta(\iota, v) \) contains an element of \( F \). For infinite words, there are various acceptance conditions in the literature; here we mention three: Büchi, co-Büchi, and parity. The 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 automaton is accepting if it visits \( F \) only finitely many times. A parity acceptance condition is a map \( \kappa : Q \to [1..k] \) assigning each state a color (or rank). A run is accepting if the minimal color visited infinitely often is odd. We use \( [A] \) to denote the set of words accepted by a given acceptor \( A \), and say that \( A \) accepts or recognizes \( [A] \). Two acceptors \( A \) and \( B \) are equivalent if \( [A] = [B] \).

We use three letter acronyms to describe acceptors, where the first letter is either D or N depending on whether the automaton is deterministic or nondeterministic, respectively. The second letter is one of \{F,B,C,P\}: F if this is an acceptor over finite words, B, C, or P if it is an acceptor over infinite words with Büchi, co-Büchi, or parity acceptance condition, respectively. The third letter is always A for acceptor.

For finite words, NFA and DFA have the same expressive power. A language is said to be regular if it is accepted by an NFA. For infinite words, the theory is more involved. While NPA, DPA, and NBA have the same expressive power, DBA, NC, and DC are strictly weaker than NBA. An \( \omega \)-language is said to be \( \omega \)-regular if it is accepted by an NBA.

3 Families of DFAs (FDFAs)

It is well known that two \( \omega \)-regular languages are equivalent if they agree on the set of ultimately periodic words (this is a consequence of McNaughton’s theorem [11]). An ultimately periodic word \( uv^\omega \), where \( u \in \Sigma^* \) and \( v \in \Sigma^+ \), is usually represented by the pair \( (u,v) \). A canonical representation of an \( \omega \)-regular language can thus consider only ultimately periodic words, namely define a language of pairs \( (u,v) \in \Sigma^* \times \Sigma^+ \). Such a representation \( F \) should satisfy the saturation property: considering two pairs \( (u,v) \) and \( (u',v') \), if \( uv^\omega = u'v'^\omega \) then either both \( (u,v) \) and \( (u',v') \) are accepted by \( F \) or both are rejected by \( F \).

A family of DFAs (F DFA) accepts such pairs \( (u,v) \) of finite words. Intuitively, it consists of a leading automaton \( Q \) with no acceptance condition that runs on the prefix-word \( u \), and for

---

2 There are other acceptance conditions in the literature, the most known of which are weak, Rabin, Streett, and Müller. The three conditions that we concentrate on are the most used ones; Büchi and co-Büchi due to their simplicity, and parity due to being the simplest for which the deterministic variant is strong enough to express all \( \omega \)-regular languages.
Dana Angluin, Udi Boker, and Dana Fisman

The main differences between the two definitions are: i) In [9], a pair \((u, v)\) is accepted by an FＤFＡ \(F\) if there is some factorization \((x, y)\) of \((u, v)\), such that \(Q(x) = q\) and \(P_q(y)\) accepts \(v\); and ii) in [9], the FＤFＡ \(F\) should also satisfy the constraint that for all words \(u \in \Sigma^\omega\) and \(v, v' \in \Sigma^\omega\), if \(P_{Q(u)}(v) = P_{Q(u)}(v')\) then \(Q(uv) = Q(uv')\).

3 The FＤFＡs defined here follow the definition in [1], which is a little different from the definition of FＤFＡs in [9]; the latter provide a machine model for the families of right congruences introduced in [10]. The main differences between the two definitions are: i) In [9], a pair \((u, v)\) is accepted by an FＤFＡ \(F = (Q, P)\) if there is some factorization \((x, y)\) of \((u, v)\), such that \(Q(x) = q\) and \(P_q(y)\) accepts \(v\); and ii) in [9], the FＤFＡ \(F\) should also satisfy the constraint that for all words \(u \in \Sigma^\omega\) and \(v, v' \in \Sigma^\omega\), if \(P_{Q(u)}(v) = P_{Q(u)}(v')\) then \(Q(uv) = Q(uv')\).
Families of DFAs as Acceptors of $\omega$-Regular Languages

We define $P(\Sigma^\omega)$, which there are eventually only $4$ Boolean operations as a consequence of Theorem 10, which shows that they characterize exactly the set of $\omega$-regular languages. We provide below explicit algorithms for these operations, showing that they can be done effectively.

Complementation of an DFA is simply done by switching between accepting and non-accepting states in the progress automata, as is done with DFAs.

**Theorem 2.** Let $F$ be an DFA. There is a constant-space algorithm to obtain an DFA $F^c$, such that $[F^c] = \Sigma^+ \setminus [F]$, $|F^c| = |F|$, and $F^c$ is saturated iff $F$ is.

**Proof.** Let $F = (Q, P)$, where for each state $q$ of $Q$, $P$ has the DFA $P_q = (\Sigma, P_q, q, \delta_q, F_q)$. We define $F^c$ to be the DFA $(Q, P^c)$, where for each state $q$ of $Q$, $P^c$ has the DFA $P^c_q = (\Sigma, P_q, q, \delta_q, P_q \setminus F_q)$. We claim that $F^c$ recognizes the complement language of $F$. Indeed, let $(u, v) \in \Sigma^+$ and $(x, y)$ its normalization with respect to $Q$. Then $(u, v) \in [F]$ iff $y \in [P_{Q(x)}]$ if $x \in [P_{Q(x)}]$ if $(u, v) \in [F^c]$.

---

4 Another model that lies between deterministic and nondeterministic automata are “semi-deterministic Büchi automata” [25], which are Büchi automata that are deterministic in the limit: from every accepting state onward, their behavior is deterministic. Yet, as opposed to DFAs, complementation of semi-deterministic Büchi automata might involve an exponential state blowup [2].
Since $F$ is saturated, so is $F^c$, as for all pairs $(u, v)$ and $(u', v')$ such that $uv^{i\omega} = u'v'^{i\omega}$, $F$ either accepts or rejects them both, implying that $F^c$ either rejects or accepts them both, respectively.

Union and intersection of saturated DFDFAs also resemble the case of DFAs, and are done by taking the product of the leading automata and each pair of progress automata. Yet, the correctness proof is a bit subtle, and relies on the following lemma, which shows that for a normalized pair $(x, y)$, the period-word $y$ can be manipulated in a certain way, while retaining normalization.

**Lemma 3.** Let $Q$ be an automaton, and let $(x, y)$ be the normalization of some $(u, v) \in \Sigma^{*+}$ w.r.t. $Q$. Then for every $i \geq 0$, $j \geq 1$ and finite words $y', y''$ such that $y = y'y''$, we have that $(xy'y', (y''y')^j)$ is the normalization of itself w.r.t. $Q$.

**Proof.** Let $x_1 = xy'y'$ and $y_1 = (y''y')^j$. Since $(x, y)$ is normalized w.r.t. $Q$, we know that the run of $Q$ on $xy''$ is of the form $q_0, q_1, \ldots, q_{k-1}, (q_k, q_{k+1}, q_{k+2}, \ldots, q_m)^\omega$, where $|x| = k$ and $|y| = (m - k) + 1$. As $xy'' = x_1 y''_1$, the run of $Q$ on $x_1 y''_1$ is identical. Since $x$ is a prefix of $x_1$, the position $|x_1|$ lies within the repeated period, implying that $|x_1|$ is the first position, from $|x_1|$ onwards, that is repeated along the aforementioned run. Since $y_1$ is a cyclic repetition of $y$, and $Q$ loops back over $y$, it also loops back over $y_1$. Thus $(x_1, y_1)$ is the normalization of itself w.r.t. $Q$. ▶

We continue with the union and intersection of saturated DFDFAs.

**Theorem 4.** Let $F_1$ and $F_2$ be saturated DFDFAs of size $(n_1, k_1)$ and $(n_2, k_2)$, respectively. There exist logarithmic-space algorithms to obtain saturated DFDFAs $H$ and $H'$ of size $(n_1 n_2, k_1 k_2)$, such that $[H] = [F_1] \cap [F_2]$ and $[H'] = [F_1] \cup [F_2]$.

**Proof.** The constructions of the union and intersection DFDFAs are similar, only differing by the accepting states. We shall thus describe them together.

**Construction:**

Given two automata $A_1$ and $A_2$, where $A_i = (\Sigma, A_i, \iota_i, \delta_i)$, we denote by $A_1 \times A_2$ the product automaton $(\Sigma, A_1 \times A_2, (\iota_1, \iota_2), \delta_\times)$, where for every $\sigma \in \Sigma$, $\delta_\times((q_1, q_2), \sigma) = (\delta(q_1, \sigma), \delta(q_2, \sigma))$.

Given two DFDFAs $D_1 = (A_1, F_1)$ and $D_2 = (A_2, F_2)$, over the automata $A_1$ and $A_2$, and with the accepting states $F_1$ and $F_2$, respectively, we define the DFDFAs $D_1 \otimes D_2$ and $D_1 \oplus D_2$ as follows:

- $D_1 \otimes D_2 = (A_1 \times A_2, F_1 \times F_2)$
- $D_1 \oplus D_2 = (A_1 \times A_2, F_1 \times A_2 \cup A_1 \times F_2)$

Given two sets of DFDFAs $P_1$ and $P_2$, we define the sets of DFDFAs $P_1 \otimes P_2$ and $P_1 \oplus P_2$ as follows:

- $P_1 \otimes P_2 = \{ D_1 \otimes D_2 \mid D_1 \in P_1 \text{ and } D_2 \in P_2 \}$
- $P_1 \oplus P_2 = \{ D_1 \oplus D_2 \mid D_1 \in P_1 \text{ and } D_2 \in P_2 \}$

Given saturated DFDFAs $F_1 = (Q_1, P_1)$ and $F_2 = (Q_2, P_2)$, we claim that $H = (Q_1 \times Q_2, P_1 \oplus P_2)$ and $H' = (Q_1 \times Q_2, P_1 \otimes P_2)$ are saturated DFDFAs that recognize the intersection and union of $[F_1]$ and $[F_2]$, respectively.

Notice that the number of states in $H$ and $H'$ is quadratic in the number of states in $F_1$ and $F_2$, yet their representations is very similar to the representations of $F_1$ and $F_2$, for which reason the construction can be done in logarithmic space.
Correctness:

Consider a pair \((u, v)\) in \(\Sigma^+\). Let \((x_1, y_1)\) and \((x_2, y_2)\) be its normalization with respect to \(Q_1\) and \(Q_2\), respectively, where \(x_1 = uv^{i_1}, y_1 = v^{j_1}\), \(x_2 = uv^{i_2}, y_2 = v^{j_2}\). Let \(i = \max(i_1, i_2)\) and \(j = \min(j_1, j_2)\) be the least common multiple of \((i_1, j_2)\). Define \(x = uv^i\) and \(y = v^j\).

One can verify that the normalization of \((u, v)\) with respect to \(Q_1\) \(\times\) \(Q_2\) is \((x, y)\).

We have \(Q_1 \times Q_2(xy) = Q_1 \times Q_2(x) = (Q_1(x), Q_2(x))\). Since \(xy^ω = x_1y_1^ω\) and \(F_1\) is saturated, we get that \((x, y) \in [F_1]\) iff \((x_1, y_1) \in [F_1]\). Since the pair \((x, y)\) satisfies the requirements of Lemma 3 w.r.t. \((x_1, y_1)\) and \(Q_1\), it follows that \((x, y)\) is a normalization of itself w.r.t. \(Q_1\). Thus, \(y \in P_{Q_1(x)}\) iff \(y_1 \in P_{Q_1(x_1)}\). Analogously, \(y \in P_{Q_2(x)}\) iff \(y_2 \in P_{Q_2(x_2)}\).

Hence, \((u, v) \in [F_1] \cap [F_2]\) iff \((y_1 \in P_{Q_1(x_1)}\) and \(y_2 \in P_{Q_2(x_2)}\) if \(y \in P_{Q_1(x)} \cup P_{Q_2(x)}\) iff \((u, v) \in H\). Similarly, \((u, v) \in [F_1] \cup [F_2]\) iff \((y_1 \in P_{Q_1(x_1)}\) or \(y_2 \in P_{Q_2(x_2)}\) iff \(y \in P_{Q_1(x)} \cup P_{Q_2(x)}\) iff \((u, v) \in H\).

The saturation of \(H\) and \(H'\) directly follows from the above proof of the languages they recognize: consider two pairs \((u, v)\) and \((u', v')\), such that \(uv^ω = u'v'^ω\). Then, by the saturation of \(F_1\) and \(F_2\), both pairs either belong, or not, to each of \([F_1]\) and \([F_2]\). Hence, both pairs belong, or not, to each of \([H] = [F_1] \cap [F_2]\) and \([H'] = [F_1] \cup [F_2]\). ▶

Decision procedures

All of the basic decision problems can be resolved in nondeterministic logarithmic space, using the Boolean operations above and corresponding decision algorithms for DFAs.

The first decision question to consider is that of membership: given a pair \((u, v)\) and an FDFA \(F = (Q, P)\), does \(F\) accept \((u, v)\)? The question is answered by normalizing \((u, v)\) into a pair \((x, y)\) and evaluating the runs of \(Q\) over \(x\) and \(P_{Q(x)}\) over \(y\). A normalized pair is determined by traversing along \(Q\), making up to \(|Q|\) repetitions of \(v\). Notice that memory wise, \(x\) and \(y\) only require a logarithmic amount of space, as they are of the form \(x = uv^i\) and \(y = v^j\), where the representation of \(i\) and \(j\) is bounded by \(\log |Q|\). The overall logarithmic-space solution follows from the complexity of algorithms for deterministically traversing along an automaton.

▶ Proposition 5. Given a pair \((u, v) \in \Sigma^+\) and an FDFA \(F\) of size \((n, k)\), the membership question, of whether \((u, v) \in [F]\), can be resolved in deterministic space of \(O(\log n + \log k)\).

The next questions to consider are those of emptiness and universality, namely given an FDFA \(F = (Q, P)\), whether \([F] = \emptyset\), and whether \([F] = \Sigma^+\), respectively. Notice that the universality problem is equivalent to the emptiness problem over the complement of \(F\). For nondeterministic automata, the complement automaton might be exponentially larger than the original one, making the universality problem much harder than the emptiness problem. Luckily, FDFA complementation is done in constant space, as is the case with deterministic automata, making the emptiness and universality problems equally easy.

The emptiness problem for an FDFA \((Q, P)\) cannot be resolved by only checking whether there is a nonempty progress automaton in \(P\), since it might be that the accepted period \(v\) is not part of any normalized pair. Yet, the existence of a prefix-word \(x\) and a period-word \(y\), such that \(Q(x) = Q(xy)\) and \(P_{Q(x)}\) accepts \(y\) is a sufficient and necessary criterion for the nonemptiness of \(F\). This can be tested in NLOGSPACE. Hardness in NLOGSPACE follows by a reduction from graph reachability [8].

▶ Theorem 6. Emptiness and universality for FDAs are NLOGSPACE-complete.

Proof. An FDFA \(F = (Q, P)\) is not empty iff there exists a pair \((u, v) \in \Sigma^+\), whose normalization is some pair \((x, y)\), such that \(P_{Q(x)}\) accepts \(y\). As a normalized pair is a
normalization of itself, a sufficient and necessary criterion for the nonemptiness of \( F \) is the existence of a pair \((x, y)\), such that \( Q(x) = Q(xy) \) and \( P_{Q(x)} \) accepts \( y \).

We can nondeterministically find such a pair \((x, y)\) in logarithmic space by guessing \( x \) and \( y \) (a single letter at each step), and traversing along \( Q \) and \( P_{Q(x)} \) [7].

Hardness in NLOGSPACE follows by a reduction from graph reachability [8].

As FDFAs complementation is done in constant space (Theorem 2), the universality problem has the same space complexity.

The last decision questions we handle are those of equality and containment, namely given saturated FDFAs \( F \) and \( F' \), whether \([F] = [F']\) and whether \([F] \subseteq [F']\), respectively. Equality reduces to containment, as \([F] = [F']\) iff \([F] \subseteq [F']\) and \([F'] \subseteq [F]\). Containment can be resolved by intersection, complementation, and emptiness check, as \([F] \subseteq [F']\) iff \([F] \cap [F']'' = \emptyset\). Hence, by Theorems 2, 4, and 6, these problems are NLOGSPACE-complete. Note that NLOGSPACE hardness immediately follows by reduction from the emptiness problem, which asks whether \([F] = \emptyset\). The complexity for equality and containment is easily derived from that of emptiness, intersection and complementation.

**Proposition 7.** Equality and containment for saturated FDFAs are NLOGSPACE-complete.

**Saturation check**

All of the operations and decision problems above assumed that the given FDFAs are saturated. This is indeed the case when learning FDFAs via the algorithm of [1], and when translating \( \omega \)-automata to FDFAs (see Section 5). We show below that the decision problem of whether an arbitrary FDFAs is saturated is in PSPACE. We leave the question of whether it is PSPACE-complete open.

**Theorem 8.** The problem of deciding whether a given FDFAs is saturated is in PSPACE.

**Proof.** Let \( F = (Q, P) \) be an FDFAs of size \((n, k)\). We first show that if \( F \) is unsaturated then there exist words \( u, v', v'' \) such that \(|u| \leq n\) and \(|v'|, |v''| \leq n^nk^{2k}\), and non-negative integers \( l, r \leq k\) such that \((u, (v'v'')) \in F\) while \((u, (v'v'')) \notin F\).

If \( F \) is unsaturated then there exists some ultimately periodic word \( w \in \Sigma^* \) that has two different decompositions to prefix and periodic words on which \( F \) provides different answers. Let \( P \) and \( P' \) be the respective progress automata, corresponding to states \( q \) and \( q' \) of \( Q \). Let the run of \( Q \) on \( w \) be \( q_0, q_1, q_2, \ldots \). Since \( w \) is ultimately periodic, there exist \( i, j \in \mathbb{N} \) such that \( q_{h+j} = q_h \) for all \( h > i \). That is, eventually the run cycles through a certain sequence of states. Then \( q \) and \( q' \) must be two states on the cycle where \( w \) settles. Let \( v' \) and \( v'' \) be the subwords of \( w \) that are read on the part of the shortest such cycle from \( q \) to \( q' \) and from \( q' \) back to \( q \), respectively. Then the different decompositions are of the form \((u, (v'v''))\) and \((u', (v'v''))\) where \( u \) is a string that takes \( Q \) to \( q \). Let \( l \) and \( r \) be the shortest such, then since \( P \) and \( P' \) have at most \( k \) states, we can assume \( l, r \leq k \). We can also assume \( u \) is a shortest such string and thus \(|u| \leq n \).

For a DFA \( A = (\Sigma, Q, \iota, \delta, F) \) and a word \( v \in \Sigma^* \), we use \( \chi^A_v \) to denote the function from \( Q \) to \( Q \) defined as \( \chi^A_v(q) = \delta(q, v) \). Note that given \(|Q| = n\) there are at most \( n^k \) different such functions. Let \( \mathcal{X} = \{x_v \mid x_v = \langle v, \chi^Q_v, \chi^P_v \rangle, v \in \Sigma^* \} \). Then \( \mathcal{X} \) is the set of congruence classes of the relation \( v_1 \approx_{\mathcal{X}} v_2 \) iff \( \chi^Q_{v_1} = \chi^Q_{v_2}, \chi^P_{v_1} = \chi^P_{v_2}, \) and \( \chi^P_{v_1} = \chi^P_{v_2} \). We can build an automaton such that each state corresponds to a class in \( \mathcal{X} \), the initial state is \( \chi_v \), and the transition relation is \( \delta_{\mathcal{X}}(x_v, \sigma) = x_{w\sigma} \). The cardinality of \( \mathcal{X} \) is at most \( n^{nk^{2k}} \). Thus, every state has a representative word of length at most \( n^{nk^{2k}} \) taking the initial state to that state.
Therefore, there exist words \( y', y'' \) such that \( y' \approx_X v' \) and \( y'' \approx_X v'' \) and \( |y'|, |y''| \leq n^k k^{2k} \). Thus \( u, y', y'' \) and \( l, r \) satisfy the promised bounds.

Now, to see that FDFAs saturation is in PSPACE note we can construct an algorithm that guesses integers \( l, r \leq k \) and words \( u, v', v'' \) such that \( |u| \leq n \) and \( |v'|, |v''| \leq n k^{2k} \). It guesses the words letter by letter and constructs on the way \( \chi_{v',v''} \) and \( \chi_{v''} \). It also constructs along the way the states \( q' \) and \( q \) such that \( q = \delta(u) \) and \( q' = \delta(uv') \). It then computes the \( l \) and \( r \) powers of \( \chi_{v',v'} \) and \( \chi_{v''} \), respectively. Finally, it checks whether one of \( (\chi_{v',v'})^l(q) \) and \( (\chi_{v''}^r)'(q') \) is accepting and the other is not, and if so returns that \( \calF \) is unsaturated. The required space is \( O(nk^2 \log nk^2) \). This shows that saturation is in coNPSPACE, and by Savitch’s and Immerman–Szelepcsényi’s theorems, in PSPACE. ▶

5 Translating To and From \( \omega \)-Automata

As two \( \omega \)-regular languages are equivalent iff they agree on the set of ultimately periodic words [11], an \( \omega \)-regular language can be characterized by a language of pairs of finite words, and in particular by a saturated FDFAs. We shall write \( L \equiv L' \) to denote that a language \( L \subseteq \Sigma^* \) characterizes an \( \omega \)-regular language \( L' \). Formally:

▶ Definition 9. A language \( L \subseteq \Sigma^{*+} \) characterizes an \( \omega \)-regular language \( L' \subseteq \Sigma^\omega \), denoted by \( L \equiv L' \), if for every pair \( (u, v) \in L \), we have \( uv'' \in L' \), and for every ultimately periodic word \( uw^\omega \in L' \), we have \( (u, v) \in L \).

The families of DFAs defined in [9], as well as the analogous families of right congruences of [10], are known to characterize exactly the set of \( \omega \)-regular languages [9, 10]. This is also the case with our definition of saturated FDFAs.

▶ Theorem 10. Every saturated FDFAs characterizes an \( \omega \)-regular language, and for every \( \omega \)-regular language, there is a saturated FDFAs characterizing it.

Proof. The two directions are proved in Theorems 12 and 17, below. ▶

In this section, we analyze the state blowup involved in translating deterministic and nondeterministic \( \omega \)-automata into equivalent saturated FDFAs, and vice versa. For nondeterministic automata, we consider the Büchi acceptance condition, since it is the simplest and most commonly used among all acceptance conditions. For deterministic automata, we consider the parity acceptance condition since it is the simplest among all acceptance conditions whose deterministic version is equi-expressible to the \( \omega \)-regular languages. We also consider deterministic Büchi and co-Büchi, for the simple sub-classes they recognize.

5.1 From \( \omega \)-Automata to FDFAs

We show that DBA, DCA, and DPA have polynomial translations to saturated FDFAs, whereas translation of NBAs to FDFAs may involve an inevitable exponential blowup.

From deterministic \( \omega \)-automata

The constructions of a saturated FDFAs that characterize a given DBA, DCA, or DPA \( \calD \) share the same idea: The leading automaton is equivalent to \( \calD \), except for ignoring the acceptance condition, and each progress automaton consists of several copies of \( \calD \), memorizing the acceptance level of the period-word. For a DBA or a DCA, two such copies are enough, memorizing whether or not a Büchi (co-Büchi) accepting state was visited. For a DPA with \( k \) colors, \( k \) such copies are required.
We start with the constructions of an FDFA for a given DBA or DCA, which are almost the same.

**Theorem 11.** Let $\mathcal{D}$ be a DBA or a DCA with $n$ states. There exists a saturated FDFA $\mathcal{F}$ of size $(n, 2n)$, such that $[\mathcal{F}] \equiv [\mathcal{D}]$.

**Proof.**

**Construction:** Let $\mathcal{D} = (\Sigma, Q, \iota, \delta, \alpha)$ be a DBA or a DCA. We define the FDFA $\mathcal{F} = (Q, P)$, where $Q$ is the same as $\mathcal{D}$ (without acceptance), and each progress automaton $P_q$ has two copies of $\mathcal{D}$, having $(q, 0)$ as its initial state, and moving from the first to the second copy upon visiting a $\mathcal{D}$-accepting state. Formally: $Q = (\Sigma, Q, \iota, \delta)$, and for each state $q \in Q$, $P$ has the DFA $P_q = (\Sigma, Q \times \{0, 1\}, (q, 0), \delta', F)$, where for every $\sigma \in \Sigma$, $\delta'((q, 0), \sigma) = (\delta(q, \sigma), 0)$ if $\delta(q, \sigma) \notin \alpha$ and $(\delta(q, \sigma), 1)$ otherwise; and $\delta'((q, 1), \sigma) = (\delta(q, \sigma), 1)$. The set $F$ of accepting states is $Q \times \{1\}$ if $\mathcal{D}$ is a DBA and $Q \times \{0\}$ if $\mathcal{D}$ is a DCA.

**Correctness:** We show the correctness for the case that $\mathcal{D}$ is a DBA. The case that $\mathcal{D}$ is a DCA is analogous.

Consider a word $uv^\omega \in [\mathcal{D}]$, and let $(x, y)$ be the normalization of $(u, v)$ w.r.t. $Q$. Since $xy^\omega = uv^\omega \in [\mathcal{D}]$, it follows that $\mathcal{D}$ visits an accepting state when running on $y$ from the state $D(x)$, implying that $P_{Q(x)}(y)$ is an accepting state. Hence, $(u, v) \in [\mathcal{F}]$.

As for the other direction, consider a pair $(u, v) \in [\mathcal{F}]$, and let $(x, y)$ be the normalization of $(u, v)$ w.r.t. $Q$. Since $P_{Q(x)}(y)$ is an accepting state, it follows that $\mathcal{D}$ visits an accepting state when running on $y$ from the state $D(x)$, implying that $xy^\omega = uv^\omega \in [\mathcal{D}]$.

Note that $\mathcal{F}$ is saturated as a direct consequence of the proof that it characterizes an $\omega$-regular language.

We continue with the construction of an FDFA for a given DPA.

**Theorem 12.** Let $\mathcal{D}$ be a DPA with $n$ states and $k$ colors. There exists a saturated FDFA $\mathcal{F}$ of size $(n, kn)$, such that $[\mathcal{F}] \equiv [\mathcal{D}]$.

**Proof.**

**Construction:** Let $\mathcal{D} = (\Sigma, Q, \iota, \delta, \kappa)$ be a DPA, where $\kappa : Q \rightarrow [1..k]$. We define the FDFA $\mathcal{F} = (Q, P)$, where $Q$ is the same as $\mathcal{D}$ (without acceptance), and each progress automaton $P_q$ has $k$ copies of $\mathcal{D}$, having $(q, \kappa(q))$ as its initial state, and moving to a $j$-th copy upon visiting a state with color $j$, provided that $j$ is lower than the index of the current copy. The accepting states are those of the odd copies.

Formally: $Q = (\Sigma, Q, \iota, \delta)$, and for each state $q \in Q$, $P$ has the DFA $P_q = (\Sigma, Q \times [1..k], (q, \kappa(q)), \delta', F)$, where for every $\sigma \in \Sigma$ and $i \in [1..k]$, $\delta'((q, i), \sigma) = (\delta(q, \sigma), \min(i, \kappa(\delta(q, \sigma))))$. The set $F$ of accepting states is $\{Q \times \{i\} \mid i \text{ is odd}\}$.

**Correctness:** Analogous to the arguments in the proof of Theorem 11.

**From nondeterministic $\omega$-automata**

An NBA $\mathcal{A}$ can be translated into a saturated FDFA $\mathcal{F}$, by first determinizing $\mathcal{A}$ into an equivalent DPA $\mathcal{A}'$ [18, 7] (which might involve a $2^{O(n \log n)}$ state blowup and $O(n)$ colors [23]), and then polynomially translating $\mathcal{A}'$ into an equivalent FDFA (Theorem 12).

**Proposition 13.** Let $\mathcal{B}$ be an NBA with $n$ states. There is a saturated FDFA that characterizes $[\mathcal{B}]$ with a leading automaton and progress automata of at most $2^{O(n \log n)}$ states each.
A $2^{O(n \log n)}$ state blowup in this case is inevitable, based on the lower bound for complementing NBAs [12, 28, 22], the constant complementation of FDFAs, and the polynomial translation of a saturated FDFA to an NBA:

\textbf{Theorem 14.} There exists a family of NBAs $B_1, B_2, \ldots$, such that for every $n \in \mathbb{N}$, $B_n$ is of size $n$, while a saturated F DFA that characterizes $[B_n]$ must be of size $(m, k)$, such that $\max(m, k) \geq 2^{\Omega(n \log n)}$.

\textbf{Proof.} Michel [12] has shown that there exists a family of languages $\{L_n\}_{n \geq 1}$, such that for every $n$, there exists an NBA of size $n$ for $L_n$, but an NBA for $L_n^c$, the complement of $L_n$, must have at least $2^{n \log n}$ states.

Assume, towards a contradiction, that exist $n \in \mathbb{N}$ and a saturated F DFA $F$ of size $(m, k)$ that characterizes $L_n$, such that $\max(m, k) < 2^{\Omega(n \log n)}$. Then, by Theorem 2, there is a saturated F DFA $F^c$ of size $(m, k)$ that characterizes $L_n^c$. Thus, by Theorem 17, we have an NBA of size smaller than $(2^{\Omega(n \log n)})^5 = 2^{\Omega(n \log n)}$ for $L_n^c$. Contradiction. \hfill \Box

\subsection{From FDFAs to \(\omega\)-automata}

We show that saturated FDFAs can be polynomially translated into NBAs, yet translations of saturated FDFAs to DFAs may involve an inevitable exponential blowup.

\textbf{To nondeterministic \(\omega\)-automata}

We show below that every saturated F DFA can be polynomially translated to an equivalent NBA. Since an NBA can be viewed as a special case of an NPA, a translation of saturated FDFAs to NPAs follows. Translating saturated FDFAs to NCAs is not always possible, as the latter are not expressible enough.

The translation goes along the lines of the construction given in [4] for translating an $L_5$-automaton into an equivalent NBA. We prove below that the construction is correct for saturated FDFAs, despite the fact that saturated FDFAs can be exponentially smaller than $L_5$-automata.

We start with a lemma from [4], which will serve us for one direction of the proof.

\textbf{Lemma 15 ([4])}. Let $M, N \subseteq \Sigma^*$ such that $M \cdot N^* = M$ and $N^+ = N$. Then for every ultimately periodic word $w \in \Sigma^\omega$ we have that $w \in M \cdot N^\omega$ iff there exist words $u \in M$ and $v \in N$ such that $uv^\omega = w$.

\textbf{Lemma 16}. Let $F = (Q, P)$ be a saturated F DFA, and let $M_q, N_q, f$ and $L$ be as defined in Theorem 17. There exists an NBA $B$ with up to $O(n^2 k^3)$ states that recognizes $L$.

We continue with the translation and its correctness.

\textbf{Theorem 17}. For every saturated F DFA $F$ of size $(n, k)$, there exists an NBA $B$ with $O(n^2 k^3)$ states, such that $[F] \equiv [B]$.

\textbf{Proof.}

\textit{Construction:} Consider a saturated F DFA $F = (Q, P)$, where $Q = (\Sigma, Q_0, \iota, \delta)$, and for each state $q \in Q$, $P$ has the progress DFA $P_q = (\Sigma, P_q, \iota_q, \delta_q, F_q)$.

For every $q \in Q$, let $M_q$ be the language of finite words on which $Q$ reaches $q$, namely $M_q = \{u \in \Sigma^* \mid Q(u) = q\}$. For every $q \in Q$ and for every accepting state $f \in F_q$, let $N_{q,f}$ be the language of finite words on which $Q$ makes a self-loop on $q$, $P_q$ reaches $f$, and $P_q$
Correctness: Consider an ultimately periodic word \( uv^\omega \in [B] \). By the construction of \( B \), \( uv^\omega \in L \), where \( L \) is defined by Equation (1). Hence, \( uv^\omega \in M_q \cdot N_{q,f}^\omega \), for some \( q \in Q \) and \( f \in F_q \). By the definitions of \( M_q \) and \( N_{q,f} \), we get that \( M_q \cdot N_{q,f}^\omega \) consists of the intersection of three NBAs, for the languages \{ \( v \in \Sigma^* \mid \delta(q,v) = q \) \}, \{ \( v \in \Sigma^* \mid \mathcal{P}_q(v) = f \) \}, and \{ \( v \in \Sigma^* \mid \delta(f,v) = f \) \}, each of which can be obtained by small modifications to either \( Q \) or \( \mathcal{P}_q \), resulting in \( nk^2 \) states. Finally, the automaton \( N_{q,f}^\omega \) is obtained by adding \( \epsilon \)-transitions in the automaton of \( N_{q,f} \) from its accepting states to its initial state. Thus, each subautomaton is of size \( n + nk^2 \), and \( B \) is of size \( nk(n + nk^2) \in O(n^2k^3) \).

As for the other direction, consider a pair \((u,v) \in [F]\), and let \((x,y)\) be the normalization of \((u,v)\) w.r.t. \( Q \). We will show that \( xy^\omega \in L \), where \( L \) is defined by Equation (1), implying that \( uv^\omega \in [B] \). Let \( q = Q(x) \), so we have that \( \mathcal{P}_q(y) \) reaches some accepting state \( f \) of \( \mathcal{P}_q \). Note, however, that it still does not guarantee that \( y \in N_{q,f} \), since it might be that \( \delta_q(f,y) \neq f \).

To prove that \( xy^\omega \in L \), we will show that there is a pair \((x',y') \in \Sigma^+ \) and an accepting state \( f' \in \mathcal{P}_q \), such that \( y' = y^t \) for some positive integer \( t \), and \( y' \in N_{q,f}; \) namely \( \delta(q,y') = q \), \( \mathcal{P}_q(y') = f' \), and \( \delta(f',y') = f' \). Note first that since \( F \) is saturated, it follows that for every positive integer \( i \), \( (x,y') \in [F] \), as \( x(y')^i = xy^\omega \).

Now, for every positive integer \( i \), \( \mathcal{P}_q \) reaches some accepting state \( f_i \) when running on \( y^i \). Since \( \mathcal{P}_q \) has finitely many states, for a large enough \( i \), \( \mathcal{P}_q \) must reach the same accepting state \( f \) twice when running on \( y^i \). Let \( h \) be the smallest positive integer such that \( \mathcal{P}_q(y^h) = f \), and \( r \) the smallest positive integer such that \( \delta_q(f,y^r) = f \). Now, one can verify that choosing \( t \) to be an integer that is bigger than or equal to \( h \) and is divisible by \( r \) guarantees that \( \delta_q(f,y^t) = q \) and \( \delta_q(f',y^t) = f' \), where \( f' = \mathcal{P}_q(y^t) \).

To deterministic \( \omega \)-automata

Deterministic Büchi and co-Büchi automata are not expressive enough for recognizing every \( \omega \)-regular language. We thus consider the translation of saturated FDFAs to deterministic

---

5 The \( \epsilon \)-transitions can be removed from an NBA with no state blowup.
Families of DFAs as Acceptors of $\omega$-Regular Languages

parity automata. A translation is possible by first polynomially translating the FDFA into an NBA (Theorem 17) and then determinizing the latter into a DPA (which might involve a $2^{O(n \log n)}$ state blowup [12]).

**Proposition 18.** Let $\mathcal{F}$ be a saturated FDFA of size $(n,k)$. There exists a DPA $\mathcal{D}$ of size $2^{O(n^k \log n^k)}$, such that $\mathcal{F} \equiv \mathcal{D}$.

We show below that an exponential state blowup is inevitable. The family of languages $\{L_n\}_{n \geq 1}$ below demonstrates the inherent gap between FDFA and DPAs; an FDFA for $L_n$ may only "remember" the smallest and biggest read numbers among $\{1, 2, \ldots, n\}$, using $n^2$ states, while a DPA for it must have at least $2^{n-1}$ states.

We define the family of languages $\{L_n\}_{n \geq 1}$ as follows. The alphabet of $L_n$ is $\{1, 2, \ldots, n\}$, and a word belongs to it if and only if the following two conditions are met:

- A letter $i$ is always followed by a letter $j$, such that $j \leq i + 1$. For example, 533245 is a bad prefix, since 2 was followed by 4, while 55234122 is a good prefix.
- The number of letters that appear infinitely often is odd. For example, 2331(22343233)$^\omega$ is in $L_n$, while 1(233)$^\omega$ is not.

We show below how to construct, for every $n \geq 1$, a saturated FDFA of size polynomial in $n$ that characterizes $L_n$. Intuitively, the leading automaton handles the safety condition of $L_n$, having $n+1$ states, and ensuring that a letter $i$ is always followed by a letter $j$, such that $j \leq i + 1$. The progress automatas, which are identical, maintain the smallest and biggest number-letters that appeared, denoted by $s$ and $b$, respectively. Since a number-letter $i$ cannot be followed by a number-letter $j$, such that $j > i + 1$, it follows that the total number of letters that appeared is equal to $b - s + 1$. Then, a state is accepting if $b - s + 1$ is odd.

**Lemma 19.** Let $n \geq 1$. There is a saturated FDFA of size $(n+1,n^2)$ characterizing $L_n$.

**Proof.** We formally define an FDFA $\mathcal{F} = (\mathcal{Q}, \mathcal{P})$ for $L_n$ over $\Sigma = \{1, 2, \ldots, n\}$, as follows.

The leading automaton is $\mathcal{Q} = (\Sigma, Q, i, \delta)$, where $Q = \{\bot, q_1, q_2, \ldots, q_n\}$; $i = q_0$; and for every $i, j \in [1..n], \delta(q_i, j) = q_j$ if $j \leq i + 1$, and $\bot$ otherwise, and $\delta(\bot, j) = \bot$.

The progress automaton for the state $\bot$ consists of a single non-accepting state with a self-loop over all letters.

For every $i \in [1..n]$, the progress automaton for $q_i$ is $\mathcal{P}_i = (\Sigma, P_i, i, \delta_i, F_i)$, where:

- $P_i = [1..n] \times [1..n]$
- $i = (n, 1)$
- $\delta_i$: For every $\sigma \in \Sigma$ and $s,b \in [1..n]$, $\delta_i((s,b), \sigma) = (\min(s,\sigma), \max(b,\sigma))$.
- $F_i = \{(s,b) \mid b - s \text{ is even}\}$

Notice that the progress automaton need not handle the safety requirement, as the leading automaton ensures it, due to the normalization in the acceptance criterion of an FDFA.

A DPA for $L_n$ cannot just remember the smallest and largest letters that were read, as these letters might not appear infinitely often. Furthermore, we prove below that the DPA must be of size exponential in $n$, by showing that its state space must be doubled when moving from $L_n$ to $L_{n+1}$.

**Lemma 20.** Every DPA that recognizes $L_n$ must have at least $2^{n-1}$ states.

---

6 This is also the case when translating FDFA to deterministic Rabin [19], Streett [24], and Muller [13] automata, as explained in Remark 22.
Proof. The basic idea behind the proof is that the DPA cannot mix between 2 cycles of \( n \) different letters each. This is because a mixed cycle in a parity automaton is accepting/rejecting if its two sub-cycles are, while according to the definition of \( L_n \), the mixed cycle should reject if both its sub-cycles accept, and vice versa. Hence, whenever adding a letter, the state space must be doubled.

In the formal proof below, we dub a reachable state from which the automaton can accept some word a live state. Consider a DPA \( D_n \) that recognizes \( L_n \), and let \( q \) be some live state of \( D_n \). Observe that \([D_n^q]\), namely the language of the automaton that we get from \( D_n \) by changing the initial state to \( q \), is the same as \( L_n \) except for having some restriction on the word prefixes. More formally, if a word \( w \in [D_n^q] \) then \( w \in L_n \), and if \( w \notin L_n \) then there is a finite word \( u \), such that \( uw \in [D_n^q] \). For every \( n \in \mathbb{N} \), and every \( u \in \Sigma^* \), let \( L_{n,u} = \{w \mid uw \in L_n\} \) and let \( L_n \) denote the set of languages \( \{L_{n,u} \mid u \in \Sigma^*\} \). Note that there is actually only a finite number of prefixes \( u \) to consider (this follows e.g. from [10, Thm. 22]). Moreover, for every state \( q \) of \( D_n \) there is a corresponding word \( u_q \) such that \([D_n^q] = L_{n,u_q}\).

We prove by induction on \( n \) the following claim, from which the statement of the lemma immediately follows: Let \( D_n \) be a DPA over \( \Sigma = \{1,2,\ldots,n\} \) that recognizes some language in \( L_n \). Then there are finite words \( u,v \in \Sigma^* \), such that:

i) \( v \) contains all the letters in \( \Sigma \);

ii) the run of \( D_n \) on \( u \) reaches some live state \( p \); and

iii) the run of \( D_n \) on \( v \) returns to \( p \), while visiting at least \( 2^{n-1} \) different states.

The base cases, for \( n \in \{1,2\} \), are trivial, as they mean a cycle of size at least 1 over \( v \), for \( n = 1 \), and a cycle of size 2 for \( n = 2 \).

In the induction step, for \( n \geq 2 \), we consider a DPA \( D_{n+1} \) that recognizes some language \( L \in L_{n+1} \). We shall define \( D' \) and \( D'' \) to be the DPAs that result from \( D_{n+1} \) by removing all the transitions over the letter \( n+1 \) and by removing all the transitions over the letter 1, respectively.

Observe that for every state \( q \) that is live w.r.t. \( D_{n+1} \), we have that \([D''^q]\) is in \( L_n \), namely the language of the DPA that results from \( D_{n+1} \) by removing all the transitions over the letter \( n+1 \) and setting the initial state to \( q \) in \( L_n \). (Note that \( q \) might only be reachable via the letter \( n+1 \), yet it must have outgoing transitions over letters in \([2..n]\).) Analogously, \([D'^q]\) is isomorphic to a language in \( L_n \) via the alphabet mapping of \( i \mapsto (i-1) \). Hence, for every state \( q \) that is live w.r.t. \( D_{n+1} \), the induction hypothesis holds for \( D'' \) and \( D''' \).

We shall prove the induction claim by describing words \( u,v \in \Sigma^* \), and showing that they satisfy the requirements above w.r.t. \( D_{n+1} \). We construct \( u \) by iteratively concatenating the words \( u_1', v_1' \), \( u_2', v_2' \), and \( v_3'' \), which we define below, until the starting and ending states in some iteration \( k \) are the same. We then define the word \( v \) to be the last iteration, namely \( u_k' v_k' u_k'' v_k'' \). Let \( q_1 \) be the initial state of \( D_{n+1} \). We define for every \( i \in [1..k] \):

- \( u_i' \) and \( v_i' \) are the words that follow from the induction hypothesis on \( D'' \), where \( q_i \) is the state that \( D_{n+1} \) reaches when reading \( u_i' v_i' u_i'' v_i'' \ldots u_i'_{i-1} v_i'_{i-1} u_i''_{i-1} v_i''_{i-1} \).
- \( u_i'' \) and \( v_i'' \) are the words that follow from the induction hypothesis on \( D'' \), where \( q_i'' \) is the state that \( D_{n+1} \) reaches when reading \( u_i'' v_i'' u_i'' v_i'' \ldots u_i''_{i-1} v_i''_{i-1} u_i''_{i-1} v_i''_{i-1} \).

The word \( v \) obviously contains all the letters in \( \Sigma \), as it is composed of subwords that contain all the letters in \( \Sigma \setminus \{1\} \) and in \( \Sigma \setminus \{n+1\} \). By the definition of \( u \) and \( v \), we also have that the run of \( D_{n+1} \) on \( u \) reaches some live state \( p \), and the run of \( D_{n+1} \) on \( v \) from \( p \) returns to \( p \). Now, we need to prove that the run of \( D_{n+1} \) on \( v \) from \( p \) visits at least \( 2^n \) states.
We claim that when $D_{n+1}$ runs on $v$ from $p$, it visits disjoint set of states when reading $v'_k$ and $v''_k$. This will provide the required result, as $D_{n+1}$ visits at least $2^n-1$ states when reading each of $v'_k$ and $v''_k$.

Assume, by way of contradiction, that $D_{n+1}$ visits some state $s$ both when reading $v'_k$ and when reading $v''_k$. Let $l'$ and $r'$ be the parts of $v'_k$ that $D_{n+1}$ reads before and after reaching $s$, respectively, and $l''$ and $r''$ the analogous parts of $v''_k$. Now, define the infinite words $m' = u u'_k (l' r')^\omega$, $m'' = u u'_k l'' (r'' l''')^\omega$, and $m = u u'_k (l' r'' l''' r')^\omega$.

Observe that $m'$ and $m''$ both belong or both do not belong to $L$, since there is the same number of letters ($n$) that appear infinitely often in each of them. The word $m$, on the other hand, belongs to $L$ if $m'$ and $m''$ do not belong to $L$, and vice versa, since $n+1$ letters appear infinitely often in it. However, the set of states that are visited infinitely often in the run of $D_{n+1}$ on $m$ is the union of the sets of states that appear infinitely often in the runs of $D_{n+1}$ on $m'$ and $m''$. Thus, if $D_{n+1}$ accepts both $m'$ and $m''$ it also accepts $m$, and if it rejects both $m'$ and $m''$ it rejects $m$. (This follows from the fact that the minimal number in a union of two sets is even/odd if the minimum within both sets is even/odd.) Contradiction.

\[\blacktriangleleft\]

**Theorem 21.** There is a family of languages $\{L_n\}_{n \geq 1}$ over the alphabet $\{1, 2, \ldots, n\}$, such that for every $n \geq 1$, there is a saturated FFA of size $(n+1, n^2)$ that characterizes $L_n$, while a DPA for $L_n$ must be of size at least $2^{n^2}$.

**Proof.** By Lemmas 19 and 20. \[\blacktriangleleft\]

**Remark 22.** A small adaptation to the proof of Lemma 20 shows an inevitable exponential blowup also when translating a saturated FFA to a deterministic $\omega$-automaton with a stronger acceptance condition of Rabin [19] or Streett [24]: A mixed cycle in a Rabin automaton is rejecting if its two sub-cycles are, and a mixed cycle in a Streett automaton is accepting if its two sub-cycles are. Hence, the proof of Lemma 20 holds for both Rabin and Streett automata if proceeding in the induction step from an alphabet of size $n$ to an alphabet of size $n+2$, yielding a Rabin/Streett automaton of size at least $2^{2^n}$.

As for translating a saturated FFA to a deterministic Muller automaton [13], it is known that translating a DPA of size $n$ into a deterministic Muller automaton might require the latter to have an accepting set of size exponential in $n$ [21]. (The family of languages in [21] uses an alphabet of size exponential in the number of states of the DPA, however it can easily be changed to use an alphabet of linear size.) Hence, by Theorem 12, which shows a polynomial translation of DPAs to FFA, we get that translating an FFA to a deterministic Muller automaton entails an accepting set of exponential size, in the worst case.

6 Discussion

The interest in FFA as a representation for $\omega$-regular languages stems from the fact that they possess a correlation between the automaton states and the language right congruences, a property that traditional $\omega$-automata lack. This property is beneficial in the context of learning, and indeed an algorithm for learning $\omega$-regular languages by means of saturated FFA was recently provided [1]. Analyzing the succinctness of saturated FFA and the complexity of their Boolean operations and decision problems, we believe that they provide an interesting formalism for representing $\omega$-regular languages. Indeed, Boolean operations and decision problems can be performed in nondeterministic logarithmic space and their succinctness lies between deterministic and nondeterministic $\omega$-automata.
References


Families of DFAs as Acceptors of $\omega$-Regular Languages


