FAMILIES OF DFAS AS ACCEPTORS OF $\omega$-REGULAR LANGUAGES

DANA ANGLUIN, UDI BOKER, AND DANA FISMAN

Yale University, New Haven, CT, USA
Interdisciplinary Center (IDC), Herzliya, Israel
Ben-Gurion University, Beer-Sheva, Israel

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 fd 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 fd may 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 [Büch60] and Muller [Mul63], 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.

2012 ACM CCS: [Theory of computation]: Formal languages and automata theory—Automata over infinite objects.

Key words and phrases: Finite automata, Omega-Regular Languages.

* The present article extends [ABF16].

This research was supported by the United States - Israel Binational Science Foundation (BSF) grant 2016239 and the Israel Science Foundation (ISF) grant 1373/16.
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 ω-automata, among which are Muller, Rabin, Streett and parity automata.

The theory of ω-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 ω-automata the nondeterministic and deterministic variants have the same expressive power, none of them possesses all the nice qualities of acceptors for finite words. In particular, none has a corresponding Myhill-Nerode theorem [Ner58], 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 ω-automata has been a major drawback in obtaining learning algorithms for ω-regular languages, a question that has received much attention lately due to applications in verification and synthesis, such as black-box checking [PVY02], assume-guarantee reasoning [AvMN05, PGB+08, CCF+10, FKP11], error localization [WN05, CCK+15], regular model checking [VSVA05, NJ13], finding security bugs [RMSM09, CPPdR14, FJIV14], error localization [WN05, CCK+15], regular model checking [VSVA05, NJ13], finding security bugs [RMSM09, CPPdR14, FJIV14], programming networks [RMSM09, YAL14] and more. The reason is that learning algorithms typically build on this correspondence between the automaton and the right congruence.

Recently, several algorithms for learning an unknown ω-regular language were proposed, all using non-conventional acceptors. One uses a reduction due to [CNP93] named $L_\omega$-automata of ω-regular languages to regular languages [CCTW08], and the others use a representation termed families of DFAs [AF14, LCZL16]. Both representations are founded on the following well known property of ω-regular languages: two ω-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 [AF14].

A family of DFAs (F DFA) is composed of a leading automaton $\mathcal{Q}$ with no accepting states and for each state $q$ of $\mathcal{Q}$, a progress DFA $\mathcal{P}_q$. Intuitively, the leading automaton is responsible for processing the non-periodic part $u$, and depending on the state $q$ reached when $\mathcal{Q}$ terminated processing $u$, the respective progress DFA $\mathcal{P}_q$ 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 F DFA is of size $(n, k)$.

An earlier definition of FDFAs, given in [Kla94], provided a machine model for the families of right congruences of [MS97]. They were redefined in [AF14], where their acceptance criterion was adjusted, and their size was reduced by up to a quadratic factor. We follow the definition of [AF14].

1 Another related formalism is of Wilke algebras [Wil91, Wil93], which are two-sorted algebras equipped with several operations. An ω-language over $\Sigma^\omega$ is ω-regular if and only if there exists a two-sorted morphism from $\Sigma^\omega$ into a finite Wilke structure [Wil91]. A central difference between the F DFA theory and the algebraic theory of recognition by monoids, semigroups, ω-semigroups, and Wilke structures is that the former relates to right-congruences, while the latter is based on two-sided congruences.
In order for an FDFA to properly characterize an ω-regular language, it must satisfy the saturation property: considering two pairs \((u, v)\) and \((u', v')\), if \(uvω = u'v'ω\) then either both \((u, v)\) and \((u', v')\) are accepted or both are rejected (cf. [CNP93, Sa90]). Saturated FDFA are shown to exactly characterize the set of ω-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 [AF14] 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 ω-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 usefulness of applications built on top of them.

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

We show that saturated FDFA 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 FDFA 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 FDFA properly lie in between deterministic and nondeterministic ω-automata. We provide polynomial translations from deterministic ω-automata to FDFA and from FDFA to nondeterministic ω-automata, and show that an exponential state blowup in the opposite directions is inevitable in the worst case.

Specifically, a saturated FDFA of size \((n, k)\) can always be transformed into an equivalent nondeterministic Büchi automaton (NBA) with \(O(n^2k^3)\) states. (Recall that an FDFA of size \((n, k)\) can have up to \(n + nk\) states in total, having \(n\) states in the leading automaton and up to \(k\) states in each of the \(n\) progress automata.) As for the other direction, transforming an NBA with \(n\) states to an equivalent FDFA is shown to be in \(2^{Θ(n\log n)}\). This is not surprising since, as shown by Michel [Mic88], complementing an NBA involves a \(2^{Θ(n\log n)}\) state blowup, while FDFA complementation requires no state blowup.

Considering deterministic ω-automata, a Büchi or co-Büchi automaton (DBA or DCA) with \(n\) states can be transformed into an equivalent FDFA of size \((n, 2n)\), and a deterministic parity automaton (DPA) with \(n\) states and \(k\) colors can be transformed into an equivalent FDFA of size \((n, kn)\). As for the other direction, since DBA and DCA do not recognize all the ω-regular languages, while saturated FDFA do, a transformation from an FDFA to a DBA or DCA need not exist. Comparing FDFA to DPAs, which do recognize all ω-regular languages, we get that FDFA can be exponentially more succinct: We show a family of
languages \( \{L_n\}_{n \geq 1} \), such that for every \( n \), there exists an FDDA 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 \( 2^\frac{n}{2} \) 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 or a finite word. We use \( \epsilon \) for the empty word. An infinite word \( w \) is ultimately periodic if there are two finite words \( u \in \Sigma^* \) and \( v \in \Sigma^+ \), such that \( w = u v^\omega \). 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 \( i \) and \( j \) 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 \( A = (\Sigma, Q, \iota, \delta) \) consisting of an alphabet \( \Sigma \), a finite set \( Q \) of states, an initial state \( \iota \in Q \), 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 \( q_0 = \iota \), 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 \( \delta(q, a v) = \bigcup_{p \in \delta(q, a)} \delta(p, v) \) 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 \( (\Sigma, Q, q, \delta) \) 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, with respect to the relevant equivalence notion, as defined below.)

By augmenting an automaton with an acceptance condition \( \alpha \), thereby obtaining a tuple \( (\Sigma, Q, \iota, \delta, \alpha) \), 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\(^2\). 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 acronym 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

\(^2\)There are other acceptance conditions in the literature, the most known of which are weak, Rabin, Streett, and Muller. 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.
it is an acceptor over infinite words with Büchi, co-Büchi, or parity acceptance condition, respectively. The third letter is always $A$ for an acceptor (or automaton).

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 NPAs, DFAs, and NBAs have the same expressive power, DBAs, NCAs, and DCAs are strictly weaker than NBAs. An $\omega$-language is said to be $\omega$-regular if it is accepted by an NBA.

### 3. Families of DFAs (FDFA)

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 [McN66]). 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 $\mathcal{F}$ 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 $\mathcal{F}$ or both are rejected by $\mathcal{F}$.

A family of DFAs (FDFA) accepts such pairs $(u, v)$ of finite words. Intuitively, it consists of a deterministic leading automaton $Q$ with no acceptance condition that runs on the prefix-word $u$, and for each state $q$ of $Q$, a progress automaton $P_q$, which is a DFA that runs on the period-word $v$.

A straightforward definition of acceptance for a pair $(u, v)$, could have been that the run of the leading automaton $Q$ on $u$ ends at some state $q$, and the run of the progress automaton $P_q$ on $v$ is accepting. This goes along the lines of $L_S$-automata [CNP93]. However, such an acceptance definition does not fit well the saturation requirement, and might enforce very large automata [AFL]. The intuitive reason is that every progress automaton might need to handle the period-words of all prefix-words.

To better fit the saturation requirement, the acceptance condition of an FDFA is defined with respect to a normalization of the input pair $(u, v)$. The normalization is a new pair $(x, y)$, such that $xy^\omega = uv^\omega$, and in addition, the run of the leading automaton $Q$ on $xy^i$ ends at the same state for every natural number $i$. Over the normalized pair $(x, y)$, the acceptance condition follows the straightforward approach discussed above. This normalization resembles the implicit flexibility in the acceptance conditions of $\omega$-automata, such as the Büchi condition, and allows saturated FDFA to be up to exponentially more succinct than $L_S$-automata [AFL].

Below, we formally define an FDFA, the normalization of an input pair $(u, v)$, and the acceptance condition. We shall use $\Sigma^{*+}$ as a shorthand for $\Sigma^* \times \Sigma^+$, whereby the input to an FDFA is a pair $(u, v) \in \Sigma^{*+}$.

**Definition 3.1 (A family of DFAs (FDFA)).**

- A family of DFAs (FDFA) is a pair $(Q, P)$, where $Q = (\Sigma, Q, I, \delta)$ is a deterministic leading automaton, and $P$ is a set of $|Q|$ DFAs, including for each state $q \in Q$, a progress DFA $P_q = (\Sigma, P_q, I_q, \delta_q, F_q)$.

---

The FDFA defined here follow the definition in [AFL], which is a little different from the definition of FDFA in [Kla94], the latter provide a machine model for the families of right congruences introduced in [MS97]. The main differences between the two definitions are: i) In [Kla94], a pair $(u, v)$ is accepted by an FDFA $\mathcal{F} = (Q, P)$ if there is some factorization $(x, y)$ of $(u, v)$, such that $Q(x) = q$ and $P_q$ accepts $y$; and ii) in [Kla94], the FDFA $\mathcal{F}$ should also satisfy the constraint that for all words $u \in \Sigma^*$ and $v, v' \in \Sigma^+$, if $P_{Q(u)}(v) = P_{Q(u)}(v')$ then $Q(uv) = Q(uv')$.
We provide below algorithms for performing the Boolean operations of complementation, union, and intersection on saturated DFAs, and deciding the basic questions on them, such as emptiness, universality, and language containment. All of these algorithms can be done

4. Boolean Operations and Decision Procedures

Figure 1: Left: an unsaturated \( \text{U} \) and progress DFAs \( P^U_l \) and \( P^U_r \). Right: a saturated DFA with the leading automaton \( S \) and progress DFAs \( P^S_l \) and \( P^S_r \).

- Given a pair \((u, v) \in \Sigma^+\) and an automaton \( A \), the normalization of \((u, v)\) w.r.t \( A \) is the pair \((x, y) \in \Sigma^+\), such that \( x = uv^i \), \( y = v^j \), and \( i \geq 0 \), \( j \geq 1 \) are the smallest numbers for which \( A(uv^i) = A(uv^{i+j}) \). (By “smallest numbers” \((i, j)\), one can consider lexicographic order, having the smallest \( i \), and for it the smallest \( j \). By Fine and Wilf’s theorem, however, there is no ambiguity by just requiring “smallest numbers”. Notice that since we consider complete automata, such a unique pair \((x, y)\) is indeed guaranteed.)
- Let \( F = (Q, P) \) be an DFA, \((u, v) \in \Sigma^+\), and \((x, y) \in \Sigma^+\) the normalization of \((u, v)\) w.r.t \( Q \). We say that \((u, v)\) is accepted by \( F \) if \( Q(x) = q \) for some state \( q \) of \( Q \) and \( P_q(y) \) is an accepting state of \( P_q \).

A saturated DFA can be used to characterize an \( \omega \)-regular language (see Theorem 5.2), while an unsaturated DFA cannot.

An unsaturated DFA is depicted in Figure 1 on the left. Consider the pairs \((b, a)\) and \((ba, aa)\). The former is normalized into \((b, aa)\), as the run of the leading automaton \( U \) on “b” reaches the state \( l \), and then when \( U \) iterates on “a”, the first state to be revisited is \( l \), which happens after two iterations. The latter is normalized into \((ba, aa)\), namely it was already normalized. Now, \((b, a)\) is accepted since in the run on its normalization \((b, aa)\), \( U \) reaches the state \( l \) when running on “b”, and the progress automaton \( P^U_l \) accepts “aa”. On the other hand, \((ba, aa)\) is not accepted since the run of \( U \) on “ba” reaches the state \( r \), and the progress automaton \( P^U_r \) does not accept “aa”. Yet, \( ba^\omega = ba(aa)^\omega \), so the DFA should have either accepted or rejected both \((b, a)\) and \((ba, aa)\), were it saturated, which is not the case.

A saturated DFA is depicted in Figure 1 on the right. It accepts pairs of the forms \((\Sigma^+, a^+)\) and \((\Sigma^+, b^+)\), and characterizes the \( \omega \)-regular language \((a+b)^*(a^\omega + b^\omega)\) of words in which there are eventually only \( a \)’s or only \( b \)’s.
in nondeterministic logarithmic space, taking advantage of the partial deterministic nature of FDFAS. We conclude the section with the decision problem of whether an arbitrary FDFA is saturated, showing that it can be resolved in polynomial space.

Boolean operations. Saturated DFAs are closed under Boolean operations as a consequence of Theorem 5.2, 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 FDFA is simply done by switching between accepting and non-accepting states in the progress automata, as is done with DFAs.

**Theorem 4.1.** Let \( \mathcal{F} \) be an FDFA. There is a constant-space algorithm to obtain an FDFA \( \mathcal{F}^c \), such that \( |\mathcal{F}^c| = |\mathcal{F}|, |\mathcal{F}^c| = |\mathcal{F}|, \) and \( \mathcal{F}^c \) is saturated iff \( \mathcal{F} \) is.

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

Since \( \mathcal{F} \) is saturated, so is \( \mathcal{F}^c \), as for all pairs \( (u, v) \) and \( (u', v') \) such that \( u_1 \omega = u' v' \), \( \mathcal{F} \) either accepts or rejects them both, implying that \( \mathcal{F}^c \) either rejects or accepts them both, respectively.

Union and intersection of saturated DFAs 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 4.2.** 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 \( (xy^iy', (y''y')^j) \) is the normalization of itself w.r.t. \( Q \).

**Proof.** Let \( x_1 = xy^iy' \) and \( y_1 = (y''y')^j \). Since \( (x, y) \) is normalized w.r.t. \( Q \), we know that the run of \( Q \) on \( xy^i \) 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 \( xy^i = x_1 y_1^i \), the run of \( Q \) on \( x_1 y_1^i \) 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. \( Q \).

\(^4\) Another model that lies in between deterministic and nondeterministic automata are “semi-deterministic B"uchi automata” [VWS89], which are B"uchi automata that are deterministic in the limit: from every accepting state onward, their behaviour is deterministic. Yet, as opposed to DFAs, complementation of semi-deterministic B"uchi automata might involve an exponential state blowup [BHS+16].
We continue with the union and intersection of saturated FDFAs.

**Theorem 4.3.** Let $F_1$ and $F_2$ be saturated FDFAs of size $(n_1, k_1)$ and $(n_2, k_2)$, respectively. There exist logarithmic-space algorithms to obtain saturated FDFAs $H$ and $H'$ of size $(n_1n_2, k_1k_2)$, such that $[H] = [F_1] \cap [F_2]$ and $[H'] = [F_1] \cup [F_2]$.

**Proof.** The constructions of the union and intersection of FDFAs 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 DFAs $D_1 = (A_1, F_1)$ 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 DFAs $D_1 \otimes D_2$ 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 DFAs $P_1$ and $P_2$, we define the sets of DFAs $P_1 \otimes P_2$ and $P_1 \oplus P_2$ as follows:

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

Given saturated DFAs $F_1 = (Q_1, P_1)$ and $F_2 = (Q_2, P_2)$, we claim that $H = (Q_1 \times Q_2, P_1 \otimes P_2)$ and $H' = (Q_1 \times Q_2, P_1 \oplus P_2)$ are saturated DFAs 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 the algorithm can generate the representation of $H$ and $H'$ in space logarithmic in the size of $F_1$ and $F_2$: It sequentially traverses the states of $Q_1$, and for each of its states, it sequentially traverses the states of $Q_2$. Thus, it should only store the currently traversed states in $Q_1$ and $Q_2$, where each of them requires a storage space of size logarithmic in the number of states in $Q_1$ and $Q_2$, respectively. The same holds for generating the product of $P_1$ and $P_2$.

**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 $x_1 = u^{i_1}, y_1 = v^{j_1}, x_2 = u^{i_2},$ and $y_2 = v^{j_2}$. Let $i = \max(i_1, i_2)$ and $j$ be the least common multiple of $(j_1, j_2)$. Define $x = uv^i$ and $y = v^j$.

Observe that the normalization of $(u, v)$ with respect to $Q_1 \times Q_2$ is $(x, y)$: i) The equality $Q_1 \times Q_2(x) = Q_1 \times Q_2(xy)$ follows from taking $i$ to be bigger than both $i_1$ and $i_2$, which guarantees that further concatenations of $v$ will be along a cycle w.r.t. both $Q_1$ and $Q_2$, and taking $j$ to be multiple of both $j_1$ and $j_2$, which guarantees that both $Q_1$ and $Q_2$ complete a cycle along $y$. ii) The minimality of $i$ follows from the fact that it is equal to either $i_1$ or $i_2$, as a smaller number will contradict the minimality of either $i_1$ or $i_2$, and the minimality of $j$ follows from the fact that it is the minimal number divided by both $j_1$ and $j_2$, as a number not divided by one of them will not allow either $Q_1$ or $Q_2$ to complete a cycle.

We have $Q_1 \times Q_2(xy) = Q_1 \times Q_2(x) = (Q_1(x), Q_2(x))$. Since $xy^{i_2} = x_{1y_2^i}$ and $F_1$ is saturated, we get that $(x, y) \in [F_1]$ iff $(x_1, y_1) \in [F_1]$. Thus, the pair $(x, y)$ satisfies the requirements of Lemma 4.2 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 \((y_1 \in P_{Q_{1}(x_1)} \text{ and } y_2 \in P_{Q_{2}(x_2)})\) iff \(y \in P_{Q_{1}(x)} \otimes P_{Q_{2}(x)}\) iff \((u, v) \in \mathcal{H}\). Similarly, \((u, v) \in [F_1] \cup [F_2]\) iff \((y_1 \in P_{Q_{1}(x_1)} \text{ or } y_2 \in P_{Q_{2}(x_2)})\) iff \(y \in P_{Q_{1}(x)} \oplus P_{Q_{2}(x)}\) iff \((u, v) \in \mathcal{H}^\prime\).

The saturation of \(\mathcal{H}\) and \(\mathcal{H}^\prime\) directly follows from the above proof of the languages they recognize: consider two pairs \((u, v)\) and \((u', v')\), such that \(uv^\omega = u'v^\omega v^\omega\). 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 \([\mathcal{H}] = [F_1] \cap [F_2]\) and \([\mathcal{H}^\prime] = [F_1] \cup [F_2]\). \(\square\)

**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 F DFA \(F = (Q, P)\), 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 of \(P_{Q(x)}\) 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 4.4.** Given a pair \((u, v) \in \Sigma^*\) and an F DFA \(F\) 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 F DFA \(F = (Q, P)\), 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, F DFA 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 F DFA \((Q, P)\) 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 \([Jon75]\).

**Theorem 4.5.** Emptiness and universality for FDFAs are NLOGSPACE-complete.

**Proof.** An F DFA \(F = (Q, P)\) 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 \(y\). By Lemma \([4,2]\), a normalized pair is a normalization of itself, implying that 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 \(P_{Q(x)}\) \([GJ79]\).

Hardness in NLOGSPACE follows by a reduction from graph reachability \([Jon75]\), taking an F DFA whose leading automaton has a single state.
As F DFA complementation is done in constant space (Theorem 4.1), the universality problem has the same space complexity.

The last decision questions we handle are those of equality and containment, namely given saturated FDFAs \( \mathcal{F} \) and \( \mathcal{F}' \), whether \([\mathcal{F}] = [\mathcal{F}']\) and whether \([\mathcal{F}] \subseteq [\mathcal{F}']\), respectively. Equality reduces to containment, as \([\mathcal{F}] = [\mathcal{F}']\) iff \([\mathcal{F}] \subseteq [\mathcal{F}']\) and \([\mathcal{F}'] \subseteq [\mathcal{F}]\). Containment can be resolved by intersection, complementation, and emptiness check, as \([\mathcal{F}] \subseteq [\mathcal{F}']\) iff \([\mathcal{F}] \cap [\mathcal{F}']^c = \emptyset\). Hence, by Theorems 4.1, 4.3, and 4.5, these problems are NLOGSPACE-complete. Note that NLOGSPACE hardness immediately follows by reduction from the emptiness problem, which asks whether \([\mathcal{F}] = \emptyset\). The complexity for equality and containment is easily derived from that of emptiness, intersection and complementation.

**Proposition 4.6.** 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 [AFL14], and when translating \( \omega \)-automata to FDFAs (see Section 5). We show below that the decision problem of whether an arbitrary F DFA is saturated is in PSPACE. We leave the question of whether it is PSPACE-complete open.

**Theorem 4.7.** The problem of deciding whether a given F DFA is saturated is in PSPACE.

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

If \( \mathcal{F} \) is unsaturated then there exists some ultimately periodic word \( w \in \Sigma^\omega \) that has two different decompositions to prefix and periodic words on which \( \mathcal{F} \) provides different answers. Let \( \mathcal{P} \) and \( \mathcal{P}' \) be the respective progress automata, corresponding to some 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 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. (In case that \( q = q' \), we let \( v' \) and \( v'' \) be the whole cycle.) Then the different decompositions are of the form \((u,(v'v''))) \) and \((uv', (v'v'')^r)\) where \( u \) is a string that takes \( Q \) to \( q \). Let \( l \) and \( r \) be the shortest such, then since \( \mathcal{P} \) and \( \mathcal{P}' \) 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 \( \mathcal{A} = (\Sigma, Q, \iota, \delta, F) \) 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 \( n^n \) different such functions. Let \( \mathcal{X} = \{ \chi_v, \chi_v^Q, \chi_v^P, \chi_v^{P'} \mid v \in \Sigma^* \} \). Then \( \mathcal{X} \) is the set of congruence classes of the relation \( v_1 \approx_{\mathcal{X}} v_2 \) iff \( \chi_{v_1}^Q = \chi_{v_2}^Q, \chi_{v_1}^P = \chi_{v_2}^P, \) and \( \chi_{v_1}^{P'} = \chi_{v_2}^{P'} \). We can build an automaton such that each state corresponds to a class in \( \mathcal{X} \), the initial state is \( \chi_{\iota} \), and the transition relation is \( \delta(\chi_w, \sigma) = \chi_{w \sigma} \). The cardinality of \( \mathcal{X} \) is at most \( n^n k^{2k} \). Thus, every state has a representative word of length at most \( n^n k^{2k} \) taking the initial state to that state. Therefore, there exist words \( y', y'' \) such that \( y' \approx_{\mathcal{X}} v' \) and \( y'' \approx_{\mathcal{X}} v'' \) and \(|y'|, |y''| \leq n^n k^{2k} \). Thus \( u, y', y'' \) and \( l, r \) satisfy the promised bounds.

Now, to see that F DFA 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 \(|v'|, |v''| \leq n^n k^{2k} \).
It guesses the words letter by letter and constructs on the way $\chi^{v'}v''$ and $\chi^{v''}v'$. 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 $\chi^{v''}v'$, respectively. Finally, it checks whether one of $(\chi^{v'}v'')^l(q)$ and $(\chi^{v''}v')^r(q')$ is accepting and the other is not, and if so returns that $F$ is unsaturated. The required space is $O\left(n^2\log nk^2\right)$. 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 [McN66], an \(\omega\)-regular language can be characterized by a language of pairs of finite words, and in particular by a saturated F DFA. We shall write \(L \equiv L'\) to denote that a language \(L \subseteq \Sigma^*\) characterizes an \(\omega\)-regular language \(L' \subseteq \Sigma^\omega\). Formally:

**Definition 5.1.** 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 \(uv^\omega \in L'\), and for every ultimately periodic word \(uv^\omega \in L'\), we have \((u, v) \in L\).

The families of DFAs defined in [Kla94], as well as the analogous families of right congruences of [MS97], are known to characterize exactly the set of \(\omega\)-regular languages [Kla94, MS97]. This is also the case with our definition of saturated FDFAs.

**Theorem 5.2.** Every saturated F DFA characterizes an \(\omega\)-regular language, and for every \(\omega\)-regular language, there is a saturated F DFA characterizing it.

**Proof.** The two directions are proved in Theorems 5.4 and 5.8 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 F DFA that characterizes a given DBA, DCA, or DPA \(D\) share the same idea: The leading automaton is equivalent to \(D\), except for ignoring the acceptance condition, and each progress automaton consists of several copies of \(D\), 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 F DFA for a given DBA or DCA, which are almost the same. (Büchi and co-Büchi automata are special cases of parity automata, and therefore their translations to an F DFA, as described in Theorem 5.3, are special cases of the translation...
given in Theorem 5.3. Nevertheless, for clarity reasons, we do give their explicit translations below.

**Theorem 5.3.** 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 $\mathcal{D} = \langle \Sigma, Q, \iota, \delta, \alpha \rangle$ 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: $Q = \langle \Sigma, Q, \iota, \delta \rangle$, and for each state $q \in Q$, $P$ has the dfa $P_q = \langle \Sigma, Q \times \{0,1\}, (q,0), \delta', F \rangle$, 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 $\mathcal{D}(x)$, 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, $\mathcal{D}$ has the same structure as $Q$, and $Q(x) = Q(xy)$, it follows that $\mathcal{D}$ visits an accepting state when running on $y$ from the state $\mathcal{D}(x)$, 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 5.4.** 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 $\mathcal{D} = \langle \Sigma, Q, \iota, \delta, \kappa \rangle$ 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: $Q = \langle \Sigma, Q, \iota, \delta \rangle$, and for each state $q \in Q$, $P$ has the dfa $P_q = \langle \Sigma, Q \times [1..k], (q,\kappa(q)), \delta', F \rangle$, 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 5.3.
From nondeterministic \(\omega\)-automata. An NBA \(A\) can be translated into a saturated F DFA \(F\), by first determinizing \(A\) into an equivalent DPA \(A'\) [Pit06, FL15] (which might involve a \(2^{O(n \log n)}\) state blowup and \(O(n)\) colors [Sch09b]), and then polynomially translating \(A'\) into an equivalent F DFA (Theorem 5.4).

**Proposition 5.5.** Let \(B\) be an NBA with \(n\) states. There is a saturated F DFA that characterizes \([B]\) 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 [Mic88, Yan06, Sch09a], the constant complementation of FDFAs, and the polynomial translation of a saturated F DFA to an NBA:

**Theorem 5.6.** 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 \(\max(m, k) \geq 2^{\Omega(n \log n)}\).

**Proof.** Michel [Mic88] 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 4.1 there is a saturated F DFA \(F^c\) of size \((m, k)\) that characterizes \(L_n^c\). Thus, by Theorem 5.8 we have an NBA of size smaller than \((2^{\Omega(n \log n)})^5 = 2^{\Omega(n \log n)}\) for \(L_n^c\). Contradiction.

5.2. From FDFAs to \(\omega\)-automata. We show that saturated FDFAs can be polynomially translated into NBAs, yet translations of saturated FDFAs to DPAs may involve an inevitable exponential blowup.

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 expressive enough.

The translation goes along the lines of the construction given in [CNP93] for translating an \(L_\infty\)-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 \(L_\infty\)-automata.

We start with a lemma from [CNP93], which will serve us for one direction of the proof.

**Lemma 5.7 ([CNP93]).** 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 \(uv^n = w\).

We continue with the translation and its correctness.

**Theorem 5.8.** For every saturated F DFA \(F\) of size \((n, k)\), there exists an NBA \(B\) with \(O(n^2k^3)\) states, such that \([F] \equiv [B]\).

**Proof.**
Consider a saturated dfa $F = (Q, P)$, where $Q = \langle \Sigma, Q, \iota, \delta \rangle$, and for each state $q \in Q$, $P$ has the progress dfa $P_q = (\Sigma, P_q, t_q, \delta_q, F_q)$.

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 $P_q$ makes a self-loop on $f$, namely $N_{q,f} = \{v \in \Sigma^* \mid (\delta(q, v) = q) \land (P_q(v) = f) \land (\delta(f, v) = f)\}$. We define the $\omega$-regular language \[ L = \bigcup \{ (q,f) \mid q \in Q \land (f \in F_q) \} \]

One can construct an nba $B$ that recognizes $L$ and has up to $O(n^2k^3)$ states: $L$ is the union of $nk$ sublanguages; $B$ will have $nk$ corresponding subautomata, and will nondeterministically start in one of them. In each subautomaton, recognizing the language $M_q \cdot N_{q,f}$, a component of size $n$ for $M_q$ is obtained by a small modification to $Q$, in which $q$ can nondeterministically continue with an $\epsilon$-transition to a component realizing $N_{q,f}$. An nba for the language $N_{q,f}$ consists of the intersection of three nbas, for the languages $\{v \in \Sigma^* \mid \delta(q, v) = q\}$, $\{v \in \Sigma^* \mid P_q(v) = f\}$, and $\{v \in \Sigma^* \mid \delta(f, v) = f\}$, each of which can be obtained by small modifications to either $Q$ or $P_q$, resulting in $nk^2$ states. Finally, the automaton for $N_{q,f}$ 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 $nk(n + nk^2) \in O(n^2k^3)$.

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 (5.1). 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 $M_q$ and $N_{q,f}$ satisfy the hypothesis of Lemma 5.7, namely $N^+_q = N_{q,f}$ and $M_q \cdot N^+_q = M_q$. Therefore, by Lemma 5.7, there exist finite words $u' \in M_q$ and $v' \in N_{q,f}$ such that $u'v'^\omega = uv^\omega$. From the definitions of $M_q$ and $N_{q,f}$, it follows that the run of $Q$ on $u'$ ends in the state $q$, and $P_q$ accepts $v'$. Furthermore, by the definition of $N_{q,f}$, we have $\delta(q, v') = q$, implying that $(u', v')$ is the normalization of itself. Hence, $(u', v') \in [F]$. Since $F$ is saturated and $u'v'^\omega = uv^\omega$, it follows that $(u, v) \in [F]$, as required.

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 (5.1), implying that $uv^\omega \in [B]$. Let $q = Q(x)$, so we have that $P_q(y)$ reaches some accepting state $f$ of $P_q$. 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 $(x, y') \in \Sigma^+ \times \Sigma^*$ and an accepting state $f' \in P_q$, such that $y' = y^t$ for some positive integer $t$, and $f' \in N_{q,f'}$; namely $\delta(q, y') = q$, $P_q(y') = f'$, and $\delta_q(f', y') = f'$. Note first that since $F$ is saturated, it follows that for every positive integer $i$, $(x, y^i) \in [F]$, as $x(y^i)^\omega = xy^\omega$.

Now, for every positive integer $i$, $P_q$ reaches some accepting state $f_i$ when running on $y^i$. Since $P_q$ has finitely many states, for a large enough $i$, $P_q$ must reach the same accepting state $f_i$ twice when running on $y^i$. Let $h$ be the smallest positive integer such that $P_q(y^h) = f$, and $r$ the smallest positive integer such that $\delta_q(f, y^r) = f$. Now, one can

\[ \text{The } \epsilon\text{-transitions can be removed from an NBA with no state blowup.} \]
verify that choosing \( t \) to be an integer that is bigger than or equal to \( h \) and is divisible by \( r \) guarantees that \( \delta(q, y^t) = q \) and \( \delta_q(f', y^t) = f' \), where \( f' = \mathcal{P}_q(y^t) \).

\[ \blacksquare \]

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 parity automata. A translation is possible by first polynomially translating the F DFA into an NBA (Theorem 5.8) and then determinizing the latter into a DPA (which might involve a \( 2^{O(n \log n)} \) state blowup [Mic88]).

**Proposition 5.9.** Let \( \mathcal{F} \) be a saturated F DFA of size \( (n,k) \). There exists a DPA \( \mathcal{D} \) of size \( 2^{O(n^2 k^3 \log n^2 k^3)} \), 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 FDFAs and DFAs; an F DFA 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 all 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 \ldots is a bad prefix, since 2 was followed by 4, while 55234122 \ldots is a good prefix.
- The number of letters that appear infinitely often is odd. For example, 2331(23343233)\( \omega \) is in \( L_n \), while 1(233)\( \omega \) is not.

We show below how to construct, for every \( n \geq 1 \), a saturated F DFA 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 automata, 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 iff \( b - s + 1 \) is odd.

**Lemma 5.10.** Let \( n \geq 1 \). There is a saturated F DFA of size \( (n + 1, n^2) \) characterizing \( L_n \).

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

The leading automaton is \( \mathcal{Q} = (\Sigma, Q, \iota, \delta) \), where \( Q = \{ \bot, q_1, q_2, \ldots, q_n \} \); \( \iota = q_n \); 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 \( \mathcal{P}_i = (\Sigma, P_i, \iota_i, \delta_i, F_i) \), where:

- \( P_i = [1..n] \times [1..n] \)
- \( \iota_i = (n, 1) \)
- \( \delta_i: \) 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 F DFA.

\[ \blacksquare \]

\[^6\]This is also the case when translating FDFAs to deterministic Rabin [Rab69] and Streett [Str82] automata, as explained in Remark 5.13.
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 5.11.** Every DPA that recognizes $L_n$ must have at least $2^{n-1}$ states.

**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 $L_n$, a mixed cycle might reject even though both of 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, and for every $n \in \mathbb{N} \setminus \{0\}$, define the alphabet $\Sigma_n = \{1, 2, \ldots, n\}$. Consider a DPA $\mathcal{D}_n$ over $\Sigma_n$ that recognizes $L_n$, and let $q$ be some live state of $\mathcal{D}_n$. Observe that $[\mathcal{D}_n^q]$, namely the language of the automaton that we get from $\mathcal{D}_n$ by changing the initial state to $q$, is the same as $L_n$ except for having some restriction on the word prefixes. More formally, for every $n \in \mathbb{N}$ and $u \in \Sigma_n^*$, we define the language $L_{n,u} = \{w \mid uw \in L_n\}$, and let $\mathcal{L}_n$ denote the set of languages $\{L_{n,u} \neq \emptyset \mid u \in \Sigma_n^*\}$. Given some $L_{n,u}$, since $L_{n,u} \neq \emptyset$, there is a live state $q$ that $\mathcal{D}_n$ reaches when reading $u$, and we have $L_{n,u} = [\mathcal{D}_n^q]$. Conversely, given a DPA $\mathcal{D}_n$ for $L_n$ and a live state $q$ of $\mathcal{D}_n$, let $u$ be a finite word on which $\mathcal{D}_n$ reaches $q$, then $[\mathcal{D}_n^u] = L_{n,u}$.

We prove by induction on $n$ the following claim, from which the statement of the lemma immediately follows: Let $\mathcal{D}_n$ be a DPA over $\Sigma_n$ that recognizes some language in $\mathcal{L}_n$. Then there are finite words $u, v \in \Sigma_n^*$, such that:

i) $v$ contains all the letters in $\Sigma_n$;

ii) the run of $\mathcal{D}_n$ on $u$ reaches some live state $p$; and

iii) the run of $\mathcal{D}_n$ on $v$ from $p$ returns to $p$, while visiting at least $2^{n-1}$ different states.

The base cases, for $n \in \{1, 2\}$, are simple, as they mean a cycle of size at least 1 over $v$, for $n = 1$, and at least 2 for $n = 2$. Formally, for $n = 1$, $L_1 = \{1^\omega\}$ and $\mathcal{L}_1 = \{L_1\}$. A DPA $\mathcal{D}_1$ for $L_1$ must have a live state $p$ that is visited infinitely often when $\mathcal{D}_1$ reads $1^\omega$. Thus, there is a cycle from $p$ back to $p$ along a word $v$ of length at least 1, containing the letter ‘1’. For $n = 2$, it is still the case that the restriction on the next letter (to be up to 1-bigger than the current letter) does not influence. Hence, $L_2 = \{u1^\omega \mid u \in \Sigma_2^*\} \cup \{u2^\omega \mid u \in \Sigma_2^*\}$ and $\mathcal{L}_2 = \{L_2\}$. Consider a DPA $\mathcal{D}_2$ for $L_2$, having $k$ states. The run of $\mathcal{D}_2$ over the finite word $(12)^k$ must reach some live state $p$ twice. Thus, there is a cycle from $p$ back to itself along a word $v$ that contains both ‘1’ and ‘2’. Observe that $v$ must be of length at least 2, as otherwise there are only self loops from $p$, implying that $\mathcal{D}_2^v$ either accepts or rejects all words, in contradiction to the definition of $\mathcal{D}_2$.

In the induction step, we consider a DPA $\mathcal{D}_{n+1}$, for $n \geq 2$, that recognizes some language $L \in \mathcal{L}_{n+1}$. We shall define $\mathcal{D}'$ and $\mathcal{D}''$ to be the DPAs that result from $\mathcal{D}_{n+1}$ 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. $\mathcal{D}_{n+1}$, we have that $[\mathcal{D}'^q] \in \mathcal{L}_n$, namely the language of the DPA that results from $\mathcal{D}_{n+1}$ by removing all the transitions over the letter $n+1$ and setting the initial state to $q$ is in $\mathcal{L}_n$. This is the case since even if $q$ is only reachable via the letter $n+1$, it must have outgoing transitions over letters in $[2..n]$. Analogously, $[\mathcal{D}''^q]$ is isomorphic to a language in $\mathcal{L}_n$ 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 $D^{q_1}$ and $D^{q_2}$.

We shall prove the induction claim by describing words $u, v \in \Sigma_{n+1}^*$, and showing that they satisfy the requirements above w.r.t. $D_{n+1}$. We iteratively construct words $u'_1, u''_1, v'_1, v''_1$ until, roughly speaking, the run of $D_{n+1}$ on the concatenation of these words closes a loop. More precisely, upon the first iteration $k$ for which there exists $j < k$ such that $D_{n+1}(u'_1 u''_1 u'_2 u''_2 \ldots u'_j u''_j) = D_{n+1}(u'_1 u''_1 u'_2 u''_2 \ldots u'_k u''_k)$, we stop the iteration and define $u$ to be the word $u'_1 u''_1 u'_2 u''_2 \ldots u'_j u''_j$ and $v$ to be the word $u''_{j+1} v'_{j+1} u''_{j+2} v''_{j+2} \ldots u''_{k-1} v''_{k-1} v''_k$.

For the first iteration, we define:

- $u'_1$ and $v'_1$ are the words that follow from the induction hypothesis on $D^{q_1}$, where $q_1$ is the initial state of $D_{n+1}$.
- $u''_1$ and $v''_1$ are the words that follow from the induction hypothesis on $D^{q_2}$, where $q'_1$ is the state that $D_{n+1}$ reaches when reading $u'_1$.

For the next iterations, we define for every $i > 1$:

- $u'_i$ and $v'_i$ are the words that follow from the induction hypothesis on $D^{q_i}$, where $q_i$ is the state that $D_{n+1}$ reaches when reading $u'_1 u''_1 u'_2 u''_2 \ldots u''_{i-1} u''_{i-1}$.
- $u''_i$ and $v''_i$ are the words that follow from the induction hypothesis on $D^{q'_i}$, where $q'_i$ is the state that $D_{n+1}$ reaches when reading $u'_1 u''_1 u'_2 u''_2 \ldots u''_{i-1} u''_{i-1} u'_i$.

Note that, for all $i$, by the induction hypothesis $v'_i$ contains all letters of $\Sigma_n$ and $v''_i$ contains all the letters in $\Sigma_{n+1} \setminus \{1\}$. Since $v$ is composed of $v'_i$'s and $v''_i$'s it follows that $v$ contains all the letters in $\Sigma_{n+1}$. 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 $p$. Moreover, for every prefix $v_1$ of $v$ that reaches a state $s$, we have that $s$ is a live state, and for $v_2$ such that $v = v_1 v_2$, the run of $D_{n+1}$ on $v_2 v_1$ returns to $s$. Now, we need to prove that the run of $D_{n+1}$ on $v$ from $p$ visits at least $2^n$ states.

Let $v'$ be some $v'_i$ subword of $v$ and likewise let $v''$ be some $v''_i$ subword of $v$. We claim that the run of $D_{n+1}$ on $uw^\omega$ traverses disjoint sets of states while reading the subwords $v'$ and $v''$. This will provide the required result, since when traversing either $v'$ or $v''$ we know that $D_{n+1}$ visits at least $2^{n-1}$ different states, by the induction hypothesis.

Assume, by way of contradiction, a state $s$ that is visited by $D_{n+1}$ when traversing both $v'$ and $v''$. Let $l'$ and $r'$ be the parts of $v'$ that $D_{n+1}$ reads before and after reaching
s, respectively, and \( l'' \) and \( r'' \) the analogous parts of \( v'' \), as shown in Fig. 2. Let \( x \) be the subword of \( uv\omega \) between \( u \) and \( r' \) and let \( y \) be the subword of \( uv\omega \) between \( u \) and \( r'' \). Now, define the \( \omega \)-words \( m' = ux(r'l')\omega \), \( m'' = uy(r''l'')\omega \), and \( m = uy(r''l'r'l')\omega \).

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 iff the minimum within both sets is even/odd.)

**Theorem 5.12.** 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 \( \text{fdfa} \) of size \((n + 1, n^2)\) that characterizes \( L_n \), while a \( \text{dpa} \) for \( L_n \) must be of size at least \( 2^n - 1 \).

**Proof.** By Lemmas 5.10 and 5.11.

**Remark 5.13.** A small adaptation to the proof of Lemma 5.11 shows an inevitable exponential state blowup also when translating a saturated \( \text{fdfa} \) to a deterministic \( \omega \)-automaton with a stronger acceptance condition of Rabin [Rab69] or Streett [Str82]: 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 5.11 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 \( 2^{n+2} \).

As for translating a saturated \( \text{fdfa} \) to a deterministic Muller automaton [Mul63], it is known that translating a \( \text{dpa} \) of size \( n \) into a deterministic Muller automaton might require the latter to have an accepting set of size exponential in \( n \) [Sal89, Bok17]. Hence, by Theorem 5.4, which shows a polynomial translation of \( \text{dpas} \) to \( \text{fdfas} \), we get that translating an \( \text{fdfa} \) to a deterministic Muller automaton entails an accepting set of exponential size, in the worst case.

6. Discussion

The interest in \( \text{fdfas} \) 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 algorithms for learning \( \omega \)-regular languages by means of saturated \( \text{fdfas} \) were recently provided [AF14, LCZL16]. Analyzing the succinctness of saturated \( \text{fdfas} \) 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


