Families of DFAs as Acceptors of ω-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**

Abstract

Families of DFAs (FDFAs) provide an alternative formalism for recognizing ω-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 ω-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 ω-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 ω-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.

1998 ACM Subject Classification F.1.1 Automata, D.2.4 Formal methods

Keywords and phrases Finite automata, Omega Regular Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.11

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 ω-automata, among which are Muller, Rabin, Streett and parity automata.

[*] Research of the third author was supported by US NSF grant CCF-1138996

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 [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 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 [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 ω-regular language were proposed, both using non-conventional acceptors. One uses a reduction due to [4] named $L_\$$-automata of ω-regular languages to regular languages [6], and the other uses a representation termed *families of* DFA*s* [1]. 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^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$, can be represented as a pair of finite words (u, v). Both $L_\$$-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 $L_\$$-automata [1].

A family of DFAs (FDFA) 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^ω, 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 FDFAs, 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 ω-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 FDFAs 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 FDFAs that result from the learning algorithm of [1] are guaranteed to be saturated.

Saturated FDFAs 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 FDFAs, and how "good" are they for representing ω-regular languages. These abstract questions translate to concrete questions that concern the succinctness of FDFAs 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 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 ω-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 ω-automata. We provide polynomial translations from deterministic ω-automata to FDFAs and from FDFAs 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. As for the other direction, transforming an NBA with n states to an equivalent FDFA 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 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 FDFAs do, a transformation from an FDFA to a DBA or DCA need not exist. Comparing FDFAs to DPAs, which do recognize all ω-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 FDFA 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.)

Due to lack of space, some proofs are omitted and can be found in the full version, on the authors' home pages.

2 Preliminaries

An *alphabet* Σ is a finite set of symbols. The set of finite words over Σ is denoted by Σ^*, and the set of infinite words, termed ω-words, over Σ is denoted by Σ^ω. As usual, we use x^*, x^+, and x^ω to denote finite, non-empty finite, and infinite concatenations of x, respectively, where x can be a symbol, a finite word, or a langugae. We use ϵ for the empty word and Σ^+ for $\Sigma^* \setminus \{\epsilon\}$. 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 Σ^*, while an ω-language is a set of ω-words, that is, a subset of Σ^ω. 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 $\mathcal{A} = \langle \Sigma, Q, \iota, \delta \rangle$ consisting of an alphabet Σ, 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 $v = a_1a_2 \dots a_n$ is a sequence of states $q_0, q_1, \dots, q_n$ 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 $\mathcal{A}(v)$ as a shorthand for $\delta(\iota, v)$ and $|\mathcal{A}|$ for the number of states in Q. We use $\mathcal{A}^q$ to denote the automaton $\langle\Sigma, Q, q, \delta\rangle$ obtained from $\mathcal{A}$ by replacing the initial state with q. We say that $\mathcal{A}$ 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 α, thereby obtaining a tuple $\langle\Sigma, Q, \iota, \delta, \alpha\rangle$, 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 $[\![\mathcal{A}]\!]$ to denote the set of words accepted by a given acceptor $\mathcal{A}$, and say that $\mathcal{A}$ *accepts* or *recognizes* $[\![\mathcal{A}]\!]$. Two acceptors $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$.

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

3 Families of DFAs (FDFAs)

It is well known that two ω-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^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$, is usually represented by the pair (u, v). A canonical representation of an ω-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 *leading automaton* $\mathcal{Q}$ with no acceptance condition that runs on the prefix-word u, and for each state q of $\mathcal{Q}$, a *progress automaton* $\mathcal{P}_q$, 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 $\mathcal{Q}$ on u ends at some state q, and the run of the progress automaton $\mathcal{P}_q$ on v is accepting. This goes along the lines of $L_\$$-automata [4]. However, such an acceptance definition does not fit well the saturation requirement, and might enforce very large automata [1]. 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 $\mathcal{Q}$ 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 ω-automata, such as the Büchi condition, and allows saturated FDFAs to be up to exponentially more succinct than $L_\$$-automata [1].

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

▶ **Definition 1** (A family of DFAs (FDFA)). [1]

- A *family of* DFA*s* (FDFA) is a pair $(\mathcal{Q}, \mathbf{P})$, where $\mathcal{Q} = (\Sigma, Q, \iota, \delta)$ is a deterministic *leading* automaton, and $\mathbf{P}$ is a set of $|\mathcal{Q}|$ DFAs, including for each state $q \in Q$, a *progress* DFA $\mathcal{P}_q = (\Sigma, P_q, \iota_q, \delta_q, F_q)$.
- Given a pair $(u, v) \in \Sigma^{*+}$ and an automaton $\mathcal{A}$, the *normalization* of (u, v) w.r.t $\mathcal{A}$ 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 $\mathcal{A}(uv^i) = \mathcal{A}(uv^{i+j})$. (Since we consider complete automata, such a unique pair (x, y) is guaranteed.)
- Let $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$ be an FDFA, $(u, v) \in \Sigma^{*+}$, and $(x, y) \in \Sigma^{*+}$ the normalization of (u, v) w.r.t $\mathcal{Q}$. We say that (u, v) is *accepted* by $\mathcal{F}$ iff $\mathcal{Q}(x) = q$ for some state q of $\mathcal{Q}$ and $\mathcal{P}_q(y)$ is an accepting state of $\mathcal{P}_q$.
- We use $[\![\mathcal{F}]\!]$ to denote the set of pairs accepted by $\mathcal{F}$.
- We define the *size* of $\mathcal{F}$, denoted $|\mathcal{F}|$, as the pair $(|\mathcal{Q}|, \max\{|\mathcal{P}_q|\}_{q \in \mathcal{Q}})$.
- An FDFA $\mathcal{F}$ is saturated if for every two pairs (u, v) and (u', v') such that $uv^\omega = u'v'^\omega$, either both (u, v) and (u', v') are in $[\![\mathcal{F}]\!]$ or both are not in $[\![\mathcal{F}]\!]$.

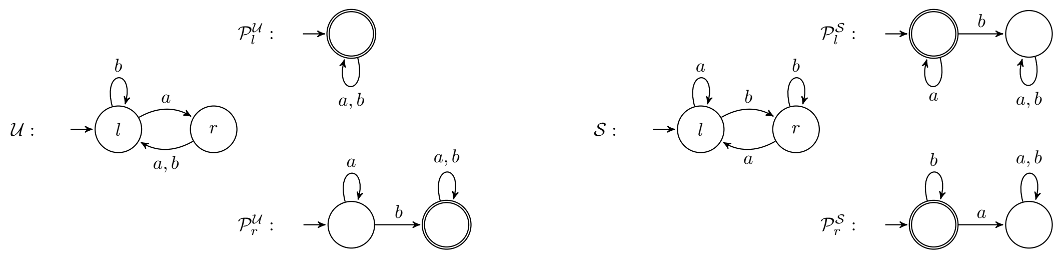
A saturated FDFA can be used to characterize an ω-regular language (see Theorem 10), while an unsaturated FDFA cannot.

An unsaturated FDFA is depicted in Figure 1 on the left. Consider the pairs (b, a) and (ba, aa). Though $b(a)^\omega = ba(aa)^\omega$, (b, a) is normalized to (b, aa) and $\mathcal{P}_l^U$ accepts aa but (ba, aa) is normalized to itself and $\mathcal{P}_r^U$ rejects aa. A saturated FDFA is depicted in Figure 1 on the right. It accepts pairs of the forms (Σ^*, a^+) and (Σ^*, b^+), and characterizes the ω-regular language $(a + b)^*(a^\omega + b^\omega)$.

4 Boolean Operations and Decision Procedures

We provide below algorithms for performing the Boolean operations of complementation, union, and intersection on saturated FDFAs, and deciding the basic questions on them, such as emptiness, universality, and language containment. All of these algorithms can be done in

[1] 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 $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$ if there is some factorization (x, y) of (u, v), such that $\mathcal{Q}(u) = q$ and $\mathcal{P}_q$ accepts v; and ii) in [9], the FDFA $\mathcal{F}$ should also satisfy the constraint that for all words $u \in \Sigma^*$ and $v, v' \in \Sigma^+$, if $\mathcal{P}_{\mathcal{Q}(u)}(v) = \mathcal{P}_{\mathcal{Q}(u)}(v')$ then $\mathcal{Q}(uv) = \mathcal{Q}(uv')$.



$\mathcal{P}_l^{\mathcal{U}}$:
a, b
$\mathcal{U}$:
b
a
l
r
a, b
a
a, b
$\mathcal{P}_r^{\mathcal{U}}$:
b
$\mathcal{P}_l^{\mathcal{S}}$:
b
a
a, b
$\mathcal{S}$:
a
b
b
l
r
a
b
a, b
$\mathcal{P}_r^{\mathcal{S}}$:
a

Figure 1 Left: an unsaturated FDFA with the leading automaton $\mathcal{U}$ and progress DFAs $\mathcal{P}_l^{\mathcal{U}}$ and $\mathcal{P}_r^{\mathcal{U}}$. Right: a saturated FDFA with the leading automaton $\mathcal{S}$ and progress DFAs $\mathcal{P}_l^{\mathcal{S}}$ and $\mathcal{P}_r^{\mathcal{S}}$.

nondeterministic logarithmic space, taking advantage of the partial deterministic nature of FDFAs.[2] 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 FDFAs are closed under Boolean operations as a consequence of Theorem 10, which shows that they characterize exactly the set of ω-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 2.** *Let $\mathcal{F}$ be an* FDFA*. There is a constant-space algorithm to obtain an* FDFA *$\mathcal{F}^c$, such that $[\![\mathcal{F}^c]\!] = \Sigma^{*+} \setminus [\![\mathcal{F}]\!]$, $|\mathcal{F}^c| = |\mathcal{F}|$, and $\mathcal{F}^c$ is saturated iff $\mathcal{F}$ is.*

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

▶ **Theorem 4.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be saturated* FDFA*s of size (n_1, k_1) and (n_2, k_2), respectively. There exist logarithmic-space algorithms to obtain saturated* FDFA*s $\mathcal{H}$ and $\mathcal{H}'$ of size $(n_1 n_2,\ k_1 k_2)$, such that $[\![\mathcal{H}]\!] = [\![\mathcal{F}_1]\!] \cap [\![\mathcal{F}_2]\!]$ and $[\![\mathcal{H}']\!] = [\![\mathcal{F}_1]\!] \cup [\![\mathcal{F}_2]\!]$.*

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 $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, does $\mathcal{F}$ accept (u, v)? The question is answered by normalizing (u, v)

[2] Another model that lies in 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 behaviour is deterministic. Yet, as opposed to FDFAs, complementation of semi-deterministic Büchi automata might involve an exponential state blowup [2].

into a pair (x, y) and evaluating the runs of $\mathcal{Q}$ over x and of $\mathcal{P}_{\mathcal{Q}(x)}$ over y. A normalized pair is determined by traversing along $\mathcal{Q}$, 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 *$\mathcal{F}$ of size (n, k), the membership question, of whether $(u, v) \in [\![\mathcal{F}]\!]$, 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 $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, whether $[\![\mathcal{F}]\!] = \emptyset$, and whether $[\![\mathcal{F}]\!] = \Sigma^{*+}$, respectively. Notice that the universality problem is equivalent to the emptiness problem over the complement of $\mathcal{F}$. 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 $(\mathcal{Q}, \mathbf{P})$ cannot be resolved by only checking whether there is a nonempty progress automaton in $\mathbf{P}$, 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 $\mathcal{Q}(x) = \mathcal{Q}(xy)$ and $\mathcal{P}_{\mathcal{Q}(x)}$ accepts y is a sufficient and necessary criterion for the nonemptiness of $\mathcal{F}$. This can be tested in NLOGSPACE. Hardness in NLOGSPACE follows by a reduction from graph reachability [8].

▶ **Theorem 6.** *Emptiness and universality for* FDFA*s are NLOGSPACE-complete.*

The complexity for *equality* and *containment* is easily derived from that of emptiness, intersection and complementation.

▶ **Proposition 7.** *Equality and containment for saturated* FDFA*s 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 ω-automata to FDFAs (see Section 5). We show below that the decision problem of whether an arbitrary FDFA 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* FDFA *is saturated is in PSPACE.*

Proof Sketch. We first show that if an FDFA $\mathcal{F}$ of size (n, k) is unsaturated then there exist words u, v', v'' such that $|u| \leq n$ and $|v'|, |v''| \leq n^n k^{2k}$, and non-negative integers $l, r \leq k$ such that $(u, (v'v'')^l) \in \mathcal{F}$ while $(uv', (v''v')^r) \notin \mathcal{F}$.

Let $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{P}'$ be the leading automaton and two relevant progress automata, with state spaces Q, P and P', respectively. We achieve the bound on the length of v' and v'', by considering for every word $v \in \Sigma^*$, the function χ_v from (Q, P, P') to (Q, P, P') defined as $\chi_v(q, p, p') = (\delta_{\mathcal{Q}}(q, v), \delta_{\mathcal{P}}(p, v), \delta_{\mathcal{P}'}(p', v))$. Note that there are up to $n^n k^{2k}$ different such functions. Hence, if v' and v'' are longer than $n^n k^{2k}$, we can replace them with shorter words that are completely equivalent w.r.t. $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{P}'$.

A coNPSPACE algorithm guesses integers $l, r \leq k$ and, letter by letter, some words u, v', v'' such that $|u| \leq n$ and $|v'|, |v''| \leq n^n k^{2k}$. Along the way, it constructs $\chi_{v'v''}$ and $\chi_{v''v'}$.

It then verifies whether one of $(\chi_{v'v''})^l$ and $(\chi_{v''v'})^r$, applied to the relevant initial states, is accepting and the other is not. By Savitch's and Immerman–Szelepcsényi's theorems, the problem is in PSPACE. ◀

5 Translating To and From ω-Automata

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

▶ **Definition 9.** A language $L \subseteq \Sigma^{*+}$ *characterizes* an ω-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 [9], as well as the analogous families of right congruences of [10], are known to characterize exactly the set of ω-regular languages [9, 10]. This is also the case with our definition of saturated FDFAs.

▶ **Theorem 10.** *Every saturated* FDFA *characterizes an ω-regular language, and for every ω-regular language, there is a saturated* FDFA *characterizing it.*

Proof. The two directions are proved in Theorems 12 and 16, below. ◀

5.1 From ω-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 ω-automata. The constructions of a saturated FDFA that characterize a given DBA, DCA, or DPA $\mathcal{D}$ share the same idea: The leading automaton is equivalent to $\mathcal{D}$, except for ignoring the acceptance condition, and each progress automaton consists of several copies of $\mathcal{D}$, 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.

▶ **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 $[\![\mathcal{F}]\!] \equiv [\![\mathcal{D}]\!]$.*

▶ **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 $[\![\mathcal{F}]\!] \equiv [\![\mathcal{D}]\!]$.*

From nondeterministic ω-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, 26, 22], the constant complementation of FDFAs, and the polynomial translation of a saturated FDFA to an NBA:

▶ **Theorem 14.** *There exists a family of* NBAs *$\mathcal{B}_1, \mathcal{B}_2, \ldots$, such that for every $n \in \mathbb{N}$, $\mathcal{B}_n$ is of size n, while a saturated* FDFA *that characterizes $[\![\mathcal{B}_n]\!]$ must be of size (m,k), such that* $\max(m,k) \geq 2^{\Omega(n \log n)}$.

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 FDFA $\mathcal{F}$ 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 FDFA $\mathcal{F}^c$ of size (m,k) that characterizes L_n^c. Thus, by Theorem 16, 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 ω-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 ω-automata. We show below that every saturated FDFA 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 [4] for translating an $L_\$$-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 $L_\$$-automata.

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

▶ **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 $uv^\omega = w$.*

We continue with the translation and its correctness.

▶ **Theorem 16.** *For every saturated* FDFA *$\mathcal{F}$ of size (n,k), there exists an* NBA *$\mathcal{B}$ with $O(n^2k^3)$ states, such that $[\![\mathcal{F}]\!] \equiv [\![\mathcal{B}]\!]$.*

Proof. *Construction:* Consider a saturated FDFA $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, where $\mathcal{Q} = \langle \Sigma, Q, \iota, \delta \rangle$, and for each state $q \in Q$, $\mathbf{P}$ has the progress DFA $\mathcal{P}_q = \langle \Sigma, P_q, \iota_q, \delta_q, F_q \rangle$.

For every $q \in Q$, let M_q be the language of finite words on which $\mathcal{Q}$ reaches q, namely $M_q = \{u \in \Sigma^* \mid \mathcal{Q}(u) = q\}$. 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 $\mathcal{Q}$ makes a self-loop on q, $\mathcal{P}_q$ reaches f, and $\mathcal{P}_q$ makes a self-loop on f, namely $N_{q,f} = \{v \in \Sigma^* \mid (\delta(q,v) = q) \wedge (\mathcal{P}_q(v) = f) \wedge (\delta_q(f,v) = f)\}$. We define the ω-regular language

$$L = \bigcup_{\{(q,f) \ \mid \ (q \in Q) \wedge (f \in F_q)\}} M_q \cdot N_{q,f}^\omega \tag{1}$$

One can construct an NBA $\mathcal{B}$ that recognizes L and has up to $O(n^2k^3)$ states.

Correctness: Consider an ultimately periodic word $uv^\omega \in [\![\mathcal{B}]\!]$. By the construction of $\mathcal{B}$, $uv^\omega \in L$, 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 and $N_{q,f}$ satisfy the hypothesis of Lemma 15, namely $N_{q,f}^+ = N_{q,f}$ and $M_q \cdot N_{q,f}^* = M_q$. Therefore, by Lemma 15, 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 $\mathcal{Q}$ on u' ends in the state q, and $\mathcal{P}_q$ 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, $(u', v') \in [\![\mathcal{F}]\!]$. Since $\mathcal{F}$ is saturated and $u'v'^\omega = uv^\omega$, it follows that $(u, v) \in [\![\mathcal{F}]\!]$, as required.

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. $\mathcal{Q}$. We will show that $xy^\omega \in L$, where L is defined by Equation (1), implying that $uv^\omega \in [\![\mathcal{B}]\!]$. Let $q = \mathcal{Q}(x)$, 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 $\delta_q(f', y') = f'$. Note first that since $\mathcal{F}$ is saturated, it follows that for every positive integer i, $(x, y^i) \in [\![\mathcal{F}]\!]$, as $x(y^i)^\omega = xy^\omega$.

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 $\hat{f}$ twice when running on y^i. Let h be the smallest positive integer such that $\mathcal{P}_q(y^h) = \hat{f}$, and r the smallest positive integer such that $\delta_q(\hat{f}, y^r) = \hat{f}$. 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 $\delta(q, y^t) = q$ and $\delta_q(f', y^t) = f'$, where $f' = \mathcal{P}_q(y^t)$. ◀

To deterministic ω-automata. Deterministic Büchi and co-Büchi automata are not expressive enough for recognizing every ω-regular language. We thus consider the translation of saturated FDFAs to deterministic parity automata. A translation is possible by first polynomially translating the FDFA into an NBA (Theorem 16) and then determinizing the latter into a DPA (which might involve a $2^{O(n \log n)}$ state blowup [12]).

▶ **Proposition 17.** *Let $\mathcal{F}$ be a saturated* FDFA*of size (n, k). There exists a* DPA *$\mathcal{D}$ of size $2^{O(n^2k^3 \log n^2k^3)}$, 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 DPAs; an FDFA for L_n may only "remember" the smallest and biggest read numbers among $\{1, 2, ..., n\}$, 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 $\{1, 2, ..., n\}$, and a word belongs to it iff 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, $2331(22343233)^\omega$ 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.

▶ **Lemma 18.** *Let $n \geq 1$. There is a saturated* FDFA *of size $(n + 1, n^2)$ characterizing L_n.*

Proof Sketch. 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. ◀

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 19.** *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, 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 $\mathcal{D}_n$ 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, if a word $w \in [\![\mathcal{D}_n^q]\!]$ then $w \in L_n$, and if $w \in L_n$ then there is a finite word u, such that $uw \in [\![\mathcal{D}_n^q]\!]$. For every $n \in \mathbb{N}$, and every $u \in \Sigma^*$, let $L_{n,u} = \{w \mid uw \in L_n\}$ and let $\mathfrak{L}_n$ 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 $\mathcal{D}_n$ there is a corresponding word u_q such that $[\![\mathcal{D}_n^q]\!] = L_{n,u_q}$.

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

(i) v contains all the letters in Σ;
(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 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 $\mathcal{D}_{n+1}$ that recognizes some language $L \in \mathfrak{L}_{n+1}$. 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 $[\![\mathcal{D}'^q]\!] \in \mathfrak{L}_n$, 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 $\mathfrak{L}_n$. (Note that q might only be reachable via the letter $n+1$, yet it must have outgoing transitions over letters in $[2..n]$.) Analogously, $[\![\mathcal{D}''^q]\!]$ is isomorphic to a language in $\mathfrak{L}_n$ via the alphabet mapping of $i \mapsto (i-1)$. Hence, for every state q that is live w.r.t. $\mathcal{D}_{n+1}$, the induction hypothesis holds for $\mathcal{D}'^q$ and $\mathcal{D}''^q$.

We shall prove the induction claim by describing words $u, v \in \Sigma^*$, and showing that they satisfy the requirements above w.r.t. $\mathcal{D}_{n+1}$. We construct u by iteratively concatenating the words u'_i, v'_i, u''_i, and v''_i, 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 $\mathcal{D}_{n+1}$. We define for every $i \in [1..k]$:

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

The word v obviously contains all the letters in Σ, 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 $\mathcal{D}_{n+1}$ on u reaches some live state p, and the run of $\mathcal{D}_{n+1}$ on v from p returns to p. Now, we need to prove that the run of $\mathcal{D}_{n+1}$ on v from p visits at least 2^n states.

We claim that when $\mathcal{D}_{n+1}$ 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 $\mathcal{D}_{n+1}$ visits at least 2^{n-1} states when reading each of v'_k and v''_k.

Assume, by way of contradiction, that $\mathcal{D}_{n+1}$ visits some state s both when reading v'_k and when reading v''_i. Let l' and r' be the parts of v'_k that $\mathcal{D}_{n+1}$ 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 $\mathcal{D}_{n+1}$ on m is the union of the sets of states that appear infinitely often in the runs of $\mathcal{D}_{n+1}$ on m' and m''. Thus, if $\mathcal{D}_{n+1}$ 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.) Contradiction. ◀

▶ **Theorem 20.** [3] *There is a family of languages $\{L_n\}_{n\geq 1}$ over the alphabet $\{1, 2, \dots, n\}$, such that for every $n \geq 1$, there is a saturated* FDFA *of size $(n+1, n^2)$ that characterizes L_n, while a* DPA *for L_n must be of size at least 2^{n-1}.*

Proof. By Lemmas 18 and 19. ◀

6 Discussion

The interest in FDFAs as a representation for ω-regular languages stems from the fact that they possess a correlation between the automaton states and the language right congruences, a property that traditional ω-automata lack. This property is beneficial in the context of

[3] A small adaptation to the proof of Lemma 19 shows an inevitable exponential blowup also when translating a saturated FDFA to a deterministic ω-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 19 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 $2^{\frac{n}{2}}$.

As for translating a saturated FDFA 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 FDFAs, we get that translating an FDFA to a deterministic Muller automaton entails an accepting set of exponential size, in the worst case.

learning, and indeed an algorithm for learning ω-regular languages by means of saturated FDFAs was recently provided [1]. Analyzing the succinctness of saturated FDFAs and the complexity of their Boolean operations and decision problems, we believe that they provide an interesting formalism for representing ω-regular languages. Indeed, Boolean operations and decision problems can be performed in nondeterministic logarithmic space and their succinctness lies between deterministic and nondeterministic ω-automata.

References

1 D. Angluin and D. Fisman. Learning regular omega languages. In Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles, editors, *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8776 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2014. doi:10.1007/978-3-319-11662-4_10.

2 F. Blahoudek, M. Heizmann, S. Schewe, J. Strejcek, and M.H. Tsai. Complementing semi-deterministic büchi automata. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9636 of *LNCS*, pages 770–787. Springer, 2016.

3 J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

4 H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational w-languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 554–566, London, UK, 1994. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645738.666444.

5 M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the language of error. In *13th Int. Symp. on Automated Technology for Verification and Analysis*, pages 114–130, 2015.

6 A. Farzan, Y. Chenand E.M. Clarke, Y. Tsay, and B. Wang. Extending automated compositional verification to the full class of omega-regular languages. In C.R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-78800-3_2.

7 D. Fisman and Y. Lustig. A modular approach for Büchi determinization. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 368–382. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-91-0, doi:10.4230/LIPIcs.CONCUR.2015.368.

8 N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 1975.

9 N. Klarlund. A homomorphism concept for omega-regularity. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1994. doi:10.1007/BFb0022276.

10 O. Maler and L. Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997. doi:10.1016/S0304-3975(96)00312-X.

11 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

12 M. Michel. Complementation is much more difficult with automata on infinite words. In *Manuscript, CNET*, 1988.

13 D.E. Muller. Infinite sequences and finite machines. In *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical design*, pages 3–16, 1963.

14 W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006.*, pages 170–185, 2006.

15 D. Neider and N. Jansen. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, pages 16–31, 2013.

16 A. Nerode. Linear automaton transformations. *Proc. of the American Mathematical Society*, 9(4):541–544, 1858.

17 D. Peled, M. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.

18 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 255–264. IEEE Computer Society, 2006. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=11132`, `doi:10.1109/LICS.2006.28`.

19 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

20 B. L. Saëc. Saturating right congruences. *ITA*, 24:545–560, 1990.

21 S. Safra. *Complexity of automata on infinite objects.* PhD thesis, Weizmann Institute of Science, 1989.

22 S. Schewe. Büchi complementation made tight. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *LIPIcs*, pages 661–672, 2009.

23 S. Schewe. Tighter bounds for the determinization of Büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 167–181, 2009.

24 R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

25 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

26 Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd Int. Colloq. on Automata, Languages, and Programming (ICALP)*, volume 4052 of *LNCS*, pages 589–600, 2006.