Strongly Unambiguous Büchi Automata Are Polynomially Predictable With Membership Queries

Dana Angluin
Yale University, New Haven, CT, USA
dana.angluin@yale.edu

Timos Antonopoulos
Yale University, New Haven, CT, USA
timos.antonopoulos@yale.edu

Dana Fisman
Ben-Gurion University, Beer-Sheva, Israel
`http://www.cs.bgu.ac.il/~dana`
dana@cs.bgu.ac.il

Abstract

A Büchi automaton is *strongly unambiguous* if every word $w \in \Sigma^\omega$ has at most one final path. Many properties of strongly unambiguous Büchi automata (SUBAs) are known. They are fully expressive: every regular ω-language can be represented by a SUBA. Equivalence and containment of SUBAs can be decided in polynomial time. SUBAs may be exponentially smaller than deterministic Muller automata and may be exponentially bigger than deterministic Büchi automata. In this work we show that SUBAs can be learned in polynomial time using membership and certain non-proper equivalence queries, which implies that they are polynomially predictable with membership queries. In contrast, under plausible cryptographic assumptions, non-deterministic Büchi automata are not polynomially predictable with membership queries.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation

Keywords and phrases Polynomially predictable languages, Automata learning, Strongly unambiguous Büchi automata, Automata succinctness, Regular ω-languages, Grammatical inference

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.8

Funding This research was supported by grant 2016239 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel and by the Office of Naval Research (ONR) award #N00014-17-1-2787.

1 Introduction

Reactive systems – systems which interact with their environment via inputs and outputs in an ongoing manner, are ubiquitous in today's life: operating systems, hardware systems, protocols and networking systems are just a few of the classical examples; self-driving cars and autonomous robots are today's leading examples. A computation of a reactive system can be abstracted by an infinite word over an alphabet consisting of inputs and outputs. The behavior of a reactive system can thus be seen as a language of infinite words (ω-words). Thus, under the assumption that the systems are finite-state, automata that process infinite words (ω-automata), and the class of languages they recognize – the *regular ω-languages* – are a useful model for reactive systems. Indeed, this is the main model used in the design, verification and synthesis of reactive systems.

In various applications, the need to infer the language of a system at hand (or in mind) has arisen. In many settings the inference problem can assume the existence of an entity answering *membership queries* (is a certain word in the language?) and *equivalence queries* (is the current hypothesis of the identity of the language correct?). This setting, enables the use of the $\mathbf{L}^*$ algorithm [2], an algorithm for inferring a regular language using membership queries and equivalence queries. Indeed, $\mathbf{L}^*$ or its improved descendants (see in [22]) have been used for tasks including black-box checking [29], assume-guarantee reasoning [18], specification mining [1], error localization [14], learning interfaces [28], regular model checking [24], finding security bugs [13], code refactoring [27, 31], learning verification fixed-points [33], as well as analyzing botnet protocols [15] and smart card readers [13].

A disadvantage of using $\mathbf{L}^*$ in applications that model behavior using ω-words is that it limits the learned languages to the class of *safety languages*, a strict subset of the regular ω-languages, for which the complement can be described by a language of finite words. However, many interesting properties of reactive systems, in particular, *liveness* and *fairness*, require richer classes of regular ω-languages. For this reason it is desirable to obtain a learning algorithm for the full class of regular ω-languages.

Learnability results regarding the class of regular ω-languages can be summarized shortly as follows. The full class of regular ω-languages can be learned either using a non-polynomial reduction to finite words, termed $(L)_\$$ [21], or using a representation by families of DFAs (FDFA), which may be exponentially more succinct than $(L)_\$$, although the running time of the algorithm may be polynomial in $(L)_\$$ in the worst case [5]. The maximal sub-class of the regular ω-languages which is known to be polynomially learnable is the set of languages accepted by deterministic weak parity automata (DwPA) [25].

In this work we show that while under plausible crypotographic assumptions, the class of ω-regular languages is not polynomially predictable with membership queries when the target language is represented using a *non-deterministic Büchi automaton* (Theorem 1), it is polynomially predictable with membership queries when the target language is represented using a *strongly unambiguous Büchi automaton* (Corollary 15).

The result on polynomial predictability with membership queries of *strongly unambiguous Büchi automata* (SUBA) is a corollary of a result (Theorem 12) on learning SUBAs in polynomial time using membership and non-proper equivalence queries (where hypotheses are represented using mod-2-MAs for a related language of finite words).[1] This contrast in learnability results for the class of regular ω-languages arises because the running time of the learning algorithm is bounded as a function of the size of the *representation* of the target language, and NBAs (non-deterministic Büchi automata) may be exponentially more succinct than SUBAs. Thus we also focus on *succinctness* comparisons between alternative representations.

In §2, we provide the preliminaries regarding Büchi automata and strongly unambiguous Büchi automata. In §3, we discuss the framework of learning with membership queries (MQs) and equivalence queries (EQs), and discuss related learnability results for regular ω-languages. In §4, we discuss the framework of polynomial predictability with MQs; relate it to the framework of learning with MQs and EQs; and provide the negative result regarding learnability using NBAs. The positive result about learnability using SUBAs is proved in §7, after some more necessary definitions are provided.

Complexity of learning algorithms is measured with respect to the size of the representation of the unknown target language. We thus provide, in §5, size comparison results between SUBAs and other models of regular ω-languages for which learning algorithms have been obtained. We show that SUBAs may be exponentially more succinct than FDFAs and DwPAs, but the other direction is also true: FDFAs and DwPAs can be exponentially more succinct than SUBAs. We further show that SUBAs can be exponentially more succinct than the DFA representation for $(L)_\$$ or its reverse.

[1] Mod-2-MAs are explained in the body of the paper, in §6.

The learning algorithm for SUBAs uses a reduction to unambiguous finite automata on finite words (UFAs) and uses the model of *mod-2-MA*; a special type of *multiplicity automata*. In §6 we explain multiplicity automata and mod-2-MAs and study size comparisons relating mod-2-MAs, DFAs and NFAs. A UFA of size n can be represented by a mod-2-MA of size at most n, and UFAs, NFAs and mod-2-MAs may be exponentially more succinct than DFAs. We show that NFAs may be exponentially more succinct than mod-2-MAs, and that if there exist infinitely many Mersenne primes, then mod-2-MAs are exponentially more succinct than NFAs.

Finally, as mentioned, we provide the positive result for learning SUBAs in §7 and conclude with a short discussion in §8.

2 Strongly Unambiguous Büchi Automata

Carton and Michel [12] introduced the definition of a complete strongly unambiguous Büchi automaton (CUBA) and proved that every regular ω-language can be accepted by a CUBA. Bousquet and Löding [9] introduced the relaxed definition of a strongly unambiguous Büchi automaton (SUBA) and proved that the containment and equivalence problems for SUBAs are solvable in polynomial time. Their proof reduces these problems for SUBAs to the containment and equivalence problems for unambiguous finite automata, which were shown to be solvable in polynomial time by Stearns and Hunt [20]. The usual translation of a linear temporal logic (LTL) formula into a Büchi automaton produces a SUBA; see the papers of Wilke [34, 35] for more on the relationship between LTL and CUBAs.

Given a finite alphabet Σ of symbols, we consider both the set of finite words Σ^* and the set of infinite or ω-words Σ^ω, which are maps from the positive integers to Σ. The term *word* refers to a finite word or an ω-word. A subset of Σ^* is a *language* and a subset of Σ^ω is an *ω-language*. For a word w, the notation $w[j]$ refers to the symbol of w indexed by j, where indices start at 1. For words w and v and a finite word u, if $w = uv$, then u is a *prefix* of w and v is a *suffix* of w.

We consider two types of automata: nondeterministic finite automata (NFAs), which accept sets of finite words, and nondeterministic Büchi automata (NBAs), which accept sets of infinite words, that can each be represented as a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$, where Σ is the finite alphabet of input symbols, Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final states.

A *path* of an NFA (resp. an NBA) on a word w is a finite (resp. infinite) sequence of states $q_0, q_1, \ldots$ such that for all j indexing symbols of w, $(q_{j-1}, w[j], q_j) \in \Delta$. A path is *initial* if $q_0 \in Q_0$. A path of an NFA is *final* if it ends with a final state. A path of an NBA is *final* if it passes infinitely often through a final state. A path is *accepting* if it is both initial and final. For an NFA or NBA, the word w is *accepted* if there exists at least one accepting path for w. We use $[\![\mathcal{A}]\!]$ for the set of words accepted by $\mathcal{A}$.

The transition relation Δ is *deterministic* if for any $q_1 \in Q$ and $\sigma \in \Sigma$, there is at most one $q_2 \in Q$ such that $(q_1, \sigma, q_2) \in \Delta$. If an NFA (resp. NBA) has a deterministic transition relation and at most one initial state, then it is a deterministic finite automaton (DFA) (resp. a deterministic Büchi automaton (DBA).) The transition relation Δ is *reverse deterministic* if the reverse relation $\Delta^r = \{(q_2, \sigma, q_1) \mid (q_1, \sigma, q_2) \in \Delta\}$ is deterministic. A state q of an NFA or NBA $\mathcal{A}$ is *live* if it appears on an accepting path for some word w. The automaton is *trim* if it contains no non-live states. The non-live states may be removed from $\mathcal{A}$ without affecting the set of words it accepts.

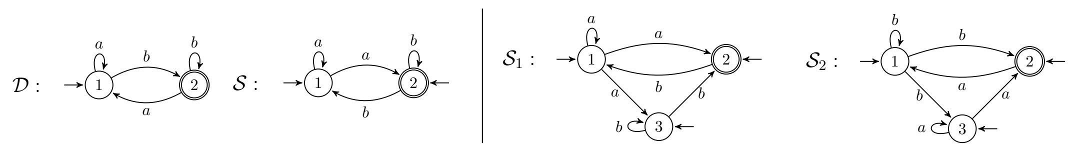


Figure 1 Left: A DBA $\mathcal{D}$ for the language $(\Sigma^* b)^\omega$ that is not a SUBA, and a SUBA $\mathcal{S}$ for the same language. Right: Two nonisomorphic minimum SUBAs $\mathcal{S}_1$ and $\mathcal{S}_2$ for $(a+b)^*(aa^*bb^*)^\omega$.

The class of languages accepted by NFAs or DFAs is the *regular languages*. The class of languages accepted by NBAs is the *regular ω-languages*. The class of languages accepted by DBAs is a proper subclass of the class of regular ω-languages.

Unambiguous Automata

- An NFA $\mathcal{A}$ is said to be *unambiguous* (UFA) if every word $w \in \Sigma^*$ has at most one accepting path.
- An NBA $\mathcal{B}$ is said to be *unambiguous* (UBA) if every word $w \in \Sigma^\omega$ has at most one accepting path.
- An NBA $\mathcal{B}$ is said to be *strongly unambiguous* (SUBA) if every word $w \in \Sigma^\omega$ has at most one final path [9].
- An NBA $\mathcal{B}$ is said to be *strongly unambiguous and complete* (CUBA) if every word $w \in \Sigma^\omega$ has exactly one final path [12].

Note that every DFA is a UFA and every UFA is an NFA, and these containments are proper. The above standard definitions for UFAs and UBAs refer to the uniqueness of accepting paths, while the definitions for SUBAs and CUBAs refer to the uniqueness of final paths, whether or not they are also initial. Every CUBA is a SUBA, and every SUBA is an UBA, and these containments are proper. Note that a DBA is not necessarily a SUBA. For instance, the DBA $\mathcal{D}$ of Fig. 1 which recognizes all words with infinitely many b's is not a SUBA, since b^ω is accepted from both states 1 and 2. The SUBA $\mathcal{S}$ of Fig. 1 accepts this language. Carton and Michel [12] proved that every regular ω-language can be accepted by a CUBA (and therefore also by a SUBA).

In this paper we focus on the class of SUBAs. It is easy to see that for any $w \in \Sigma^\omega$ if there is a final path on w, it originates at a unique state, and therefore the transition function of any trim SUBA is reverse deterministic. Unlike in the case of DFAs representing regular languages of finite words, there need not be a canonical minimum SUBA for a language. In Fig. 1 we show two non-isomorphic SUBAs $\mathcal{S}_1$ and $\mathcal{S}_2$ with the minimum possible number of states for the set of ω-words over the alphabet $\{a, b\}$ that contain an infinite number of a's and an infinite number of b's.[2]

3 Learning Regular ω-Languages With Queries

The first framework of learning that we consider is that of an algorithm learning a target language L using equivalence queries (EQs) and membership queries (MQs). In a membership query, the learning algorithm specifies a word w and receives the answer 1 if $w \in L$ and

[2] Automata $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$ and $\mathcal{A}' = (\Sigma, Q', Q_0', \Delta', F')$ are isomorphic if they are equivalent to each other under a renaming of the states. Formally, if there exists a map $h : Q \to Q'$ such that (a) $q \in Q_0$ iff $h(q) \in Q_0'$, (b) $q \in F$ iff $h(q) \in F'$ and (c) $(q_1, \sigma, q_2) \in \delta$ iff $(h(q_1), \sigma, h(q_2)) \in \delta'$.

the answer 0 if $w \notin L$, indicating whether w is a member of the target language. In an equivalence query, the learning algorithm specifies a hypothesis, that is, a set of words H, and receives either the answer “yes”, signifying that the sets H and L are equal, or the answer “no” together with an arbitrarily selected counterexample w such that w is an element of exactly one of L and H, that is, a counterexample that shows the sets H and L are not equal. If the answer to an equivalence query is “yes”, the learning algorithm has succeeded in identifying the target language L.

The above description omits a specification of how a word w is represented when it is the subject of a MQ by the learning algorithm or when it is returned as a counterexample to an EQ, as well as how a set of words H is represented by the learning algorithm in an EQ. It also omits a measure of the “size” of the target language L, which is a parameter to functions bounding the running time of the algorithm. We now address these omissions.

As is usual, a finite word w is represented by the finite sequence of symbols it contains, and its length is the length of that sequence. Arbitrary ω-words w are not represented; rather we restrict the words used in MQs and returned as counterexamples by EQs to be *ultimately periodic*, that is, words of the form $u(v)^\omega$ for finite words $u, v \in \Sigma^*$, denoting u concatenated with an infinite sequence of copies of v. This restriction is justified by the fact that two regular ω-languages that agree on the classification of all ultimately periodic words are in fact equal [10]. Given an ω-language L and a symbol \$ not in the alphabet of L, we define

$$(L)_\$ = \{u\$v \mid u(v)^\omega \in L\}.$$

Then two regular ω-languages L_1 and L_2 are equal if and only if $(L_1)_\$ = (L_2)_\$$. Consistent with this definition, we assume that the ultimately periodic word $u(v)^\omega$ is represented by the finite string $u\$v$. Calbrix et al. [11] introduced this definition and showed that if L is a regular ω-language, then $(L)_\$$ is a regular language.

To represent both the target language L and a hypothesis set H, the natural choice is an automaton of the type being considered (for example, a SUBA), with an appropriate size measure (for example, the number of states of the automaton.) If this is the situation, the EQs are termed *proper*. However, it is sometimes useful to allow a different type of representation of hypotheses H in the EQs, in which case the EQs are *non-proper*. This issue will be considered further in Section 7.

In defining the running time of a learning algorithm, each EQ or MQ is counted as an oracle call, that is, one step. The learning algorithm must be able to cope with arbitrary counterexamples, which may be arbitrarily long, so it is said to run in polynomial time if its running time is bounded by a polynomial in the size of the smallest automaton accepting the target language L and the length of the longest counterexample returned by EQs.

In this learning framework, Maler and Pnueli [25] gave a polynomial time learning algorithm using MQs and proper EQs for a strict subclass of languages accepted by DBAs, namely the class of ω-languages L such that both L and its complement $(\Sigma^\omega \setminus L)$ are accepted by DBAs. This class of languages is characterized by the deterministic weak parity automata (DwPA) and is a strict subclass of the class of all regular ω-languages.

Farzan et al. [21] applied the learning algorithm $\mathbf{L}^*$ to the problem of learning a DFA accepting the regular language $(L)_\$$, providing an algorithm using MQs and proper EQs to learn an arbitrary regular ω-language represented by NBAs.[3] One issue with this approach is

[3] $\mathbf{L}^*$ runs in time polynomial in n and ℓ where n is the size of the minimal DFA for the language, and ℓ the size of the largest counterexample, asks at most n equivalence queries and at most $O(\ell n^2)$ membership queries [2]. The complexity of the algorithm proposed by Farzan et al. is thus polynomial with respect to the size of a DFA for $(L)_\$$.

that the DFA for $(L)_\$$ may be quite large: for an NBA with m states, Calbrix et al. provide an upper bound of $2^m + 2^{2m^2+m}$ on the size of a DFA for $(L)_\$$. In Section 5 we show that the minimum DFA for $(L)_\$$ and its reverse, $(L)^r_\$$, may also be exponentially larger than a SUBA for L.

Angluin and Fisman [4] proposed a representation of regular ω-languages using families of DFAs, a representation that may be exponentially more concise than the minimum DFA for $(L)_\$$, and gave a learning algorithm $\mathbf{L}^\omega$ based on that representation. However, the intermediate hypotheses of the learning algorithm could be large, in the worst case as large as a minimum DFA for $(L)_\$$.

4 A Negative Result for Learning NBAs

NBAs may be exponentially more succinct than SUBAs (see Section 5). However, in this section we show that under plausible cryptographic assumptions, there is no polynomial time algorithm to learn NBAs using equivalence and membership queries. We now define a second, more relaxed, notion of learning, namely polynomial predictability with membership queries.

The learning framework of *polynomial predictability with membership queries* assumes that there is an upper bound n on the length of relevant example words and that there is an arbitrary unknown probability distribution D on these words. The learning algorithm has access to words randomly drawn according to D and labeled according to the target concept L, as well as MQs to L. The algorithm is also given an upper bound s on the target concept and an accuracy parameter $\epsilon > 0$. We refer to the operation of drawing a word according to D and getting the result of its membership in L as *drawing a classsified word.* The algorithm runs for some time, drawing classified words and making MQs until it requests one test word to predict. This word is also drawn according to D and, without making any further draws of classified words or MQs, the algorithm outputs 1 or 0 as a prediction of whether the word is a member of L or not. For the algorithm to be successful, this prediction must be correct with probability at least $(1 - \epsilon)$. In the running time of the algorithm, drawing a classified word, making a MQ and requesting the test word to predict count as oracle calls, that is, each counts as one step.

A class $\mathcal{C}$ of languages with a particular choice of representations is *polynomially predictable with membership queries* if there exists a (possibly randomized) learning algorithm A in the framework just described, that takes as input n, s and ϵ, such that for any positive integer n, any probability distribution D over words of length at most n, for any target language $L \in \mathcal{C}$, for any positive integer s that is an upper bound on the size of the smallest representation of L, and for any $\epsilon > 0$, the algorithm A with access to words drawn according to D and classified according to L, and access to MQs for L, runs for time bounded by a polynomial in n, s, and $1/\epsilon$ and with probability at least $(1 - \epsilon)$ correctly predicts the classification of the test word to predict.

Comparing this definition of learning to that in Section 3, the definition using EQs and MQs requires complete representations of the hypotheses in a specific form, while polynomial predictability with MQs only requires the ability to make predictions of the classifications of new examples, with no restriction on how the (implicit) hypothesis is represented. The definition of polynomial predictability with MQs is more relaxed than the definition of polynomial time learning with EQs and MQs in the sense that if a class $\mathcal{C}$ can be learned in polynomial time with EQs and MQs, where the representation used in the EQs has a polynomial time membership test, then $\mathcal{C}$ is also polynomially predictable with membership queries. Angluin [2] showed that if the hypotheses used in EQs have a polynomial time membership test, then polynomial time learnability with EQs and MQs implies PAC-learnability with MQs; this in turn implies polynomial predictability with MQs.

We now prove the following negative result for learning NBAs.

▶ **Theorem 1.** *If we assume the computational intractability of any of the following three problems: (1) testing quadratic residues modulo a composite, (2) inverting RSA encryption, or (3) factoring Blum integers, then the concept class of regular ω-languages is not polynomially predictable with membership queries, when the target language is represented by an NBA.*

This follows from the analogous result for nondeterministic finite automata proved by Angluin and Kharitonov [7, Corollary 7]. The proof below is a straightforward reduction of predicting NFAs with membership queries to predicting NBAs with membership queries.

Proof. We first describe a general transformation of an NFA $\mathcal{A}$ over the input alphabet Σ to a related NBA $\mathcal{A}'$ over an augmented input alphabet. Choose a new alphabet symbol $a \notin \Sigma$. Given an NFA $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$, we construct from it the NBA $\mathcal{A}' = (\Sigma', Q', Q_0', \Delta', F')$ as follows. Let $\Sigma' = \Sigma \cup \{a\}$. Choose a new state, say $q' \notin Q$, and let $Q' = Q \cup \{q'\}$ and $Q_0' = Q_0$. For the transition relation, let $\Delta' = \Delta \cup \{(q, a, q') \mid q \in F\} \cup \{(q', a, q')\}$. Thus, there is a new transition on a from every final state of $\mathcal{A}$ to the new state q', and a transition on a from q' to itself. Let $F' = \{q'\}$, so the new state is the only final state of $\mathcal{A}'$. It is not difficult to see that if $L \subseteq \Sigma^*$ is the language accepted by $\mathcal{A}$, then the ω-language accepted by $\mathcal{A}'$ is $L \cdot a^\omega$.

If we have a learning algorithm $\mathbf{A}'$ that polynomially predicts NBA acceptance using membership queries, we can use it to construct an algorithm $\mathbf{A}$ that polynomially predicts NFA acceptance using membership queries. To implement $\mathbf{A}$ running with a target language L accepted by the NFA $\mathcal{A}$ and the probability distribution D on Σ^*, we simulate the algorithm $\mathbf{A}'$, answering its queries as follows.

Suppose A' makes a membership query with (u, v) representing the ultimately periodic word $u(v)^\omega$. If $u(v)^\omega = xa^\omega$ for some $x \in \Sigma^*$ then we make a MQ to L with the word x and return the resulting answer. Otherwise, the answer to the MQ is 0.

Suppose $\mathbf{A}'$ requests a random classified word. Then we request a word from Σ^* chosen according to D and classified according to L, which returns a pair (x, y) where $x \in \Sigma^*$ and $y \in \{0, 1\}$ indicates whether or not $x \in L$. The element we supply to $\mathbf{A}'$ is the pair $((x, a), y)$, which indicates the choice of the ω-word xa^ω and the classification y.

Finally, when $\mathbf{A}'$ requests the test word to predict, we request the test word to predict, and receive a string $x \in \Sigma^*$, chosen according to D. The test word that we supply to $\mathbf{A}'$ to predict is (x, a), representing the ω-word xa^ω.

The queries of $\mathbf{A}'$ are answered as though it is learning the NBA $\mathcal{A}'$ derived from $\mathcal{A}$, with the distribution on ω-words giving probability $D(x)$ to xa^ω for $x \in \Sigma^*$, and probability zero to words not of this form. Thus, if $\mathbf{A}'$ predicts the correct classification of (x, a) by $\mathcal{A}'$, then $\mathbf{A}$ predicts the correct classification of x by $\mathcal{A}$. ◀

Because of the relationship between polynomial time learnability with EQs and MQs and polynomial predictability with membership queries, we have the following.

▶ **Corollary 2.** *If we assume the truth of any of the three cryptographic assumptions listed in Theorem 1 then there is no polynomial time algorithm to learn NBAs using EQs and MQs, such that the representation used in the EQs has a polynomial time membership test.*

5 Size Comparisons for SUBAs

Because the performance of a learning algorithm is bounded as a function of the size of the representation of the target language, we investigate size comparisons between SUBAs and other representations of regular ω-languages. For their conversion of an NBA of n states to an equivalent SUBA, Carton and Michel [12] give an upper bound of $(12n)^n$ on the number of states in the resulting CUBA. Bousquet and Löding [9] show that there is a family of languages L_n such that L_n is accepted by a DBA (which is also an NBA) of $n+1$ states but any SUBA to accept L_n must have at least 2^{n-1} states, showing that DBAs and NBAs may be exponentially more succinct than SUBAs. Here we focus on comparisons to representations used in learning algorithms, namely deterministic weak parity automata (DwPAs), families of DFAs (FDFAs) and DFAs for $(L)_\$$.

5.1 Comparison to FDFAs and DwPAs

A family of DFAs (FDFA) $\mathcal{F} = (\mathcal{M}, \{\mathcal{A}_q\})$ over an alphabet Σ consists of a leading deterministic automaton $\mathcal{M} = (\Sigma, Q, q_0, \delta)$ and progress DFAs $\mathcal{A}_q = (\Sigma, S_q, s_{0q}, \delta_q, F_q)$ for each $q \in Q$. Let $\mathcal{M}$ be an automaton and (u, v) a pair of finite words representing the ultimately periodic word uv^ω. The normalization of (u, v) with respect to $\mathcal{M}$ is the pair (x, y) such that $x = uv^i$, $y = v^j$ and $0 \leq i < j$ are the smallest for which uv^i and uv^{i+j} reach the same state of $\mathcal{M}$. A pair of finite words (u, v) is accepted by an FDFA $\mathcal{F} = (\mathcal{M}, \{\mathcal{A}_q\})$ if y is accepted by $\mathcal{A}_{q_x}$ where q_x is the state reached by $\mathcal{M}$ when reading x and (x, y) is the normalization of (u, v) with respect to $\mathcal{M}$. For a comprehensive discussion on FDFAs see [3].

A deterministic weak parity automaton (DwPA) is a tuple $(\Sigma, Q, q_0, \delta, \kappa)$ where the first four components are the same as for DBAs, and $\kappa : Q \to \{1, \ldots, k\}$ maps a state to a number from a finite set of naturals, referred to as *color*. A DwPA accepts an ω-word w if the maximal color *visited* along the run of the DwPA on w is even.

▶ **Theorem 3.** *There exists a family $\{L_n\}$ of ω-languages such that L_n is recognized by a SUBA of $2n+2$ states, while any FDFA and any DwPA recognizing L_n require at least 2^n states.*

Proof. The proof uses the same family of languages used by Bousquet and Löding [9] to prove an exponential translation may be required when going from SUBAs to deterministic Muller automata. The family they proposed is given by $L_n = \Sigma^* a \Sigma^{n-1} ab^\omega$ where $\Sigma = \{a, b\}$. Fig 2. in [9] shows a SUBA for L_n with $n+2$ states. To see that an FDFA requires at least n states, we note that the number of states in a leading automaton of an FDFA is at least the number of equivalence classes in the right congruence $\sim_L$ where for all finite words x and y we have that $x \sim_L y$ iff $\forall w \in \Sigma^\omega.\ xw \in L \iff yw \in L$. The proof follows since the number of equivalence classes derived from $\sim_{L_n}$ is at least 2^n [26]. The proof for DwPA follows from this as well since the number of equivalence classes of $\sim_L$ is also a lower bound for the number of states of any deterministic ω-automaton for a language L [6]. ◀

▶ **Theorem 4.** *There exists a family $\{L_n\}$ of ω-languages such that L_n is recognized by an FDFA using $n+3$ states and by a DwPA using $n+2$ states, while any SUBA recognizing L_n requires at least 2^n states.*

Proof of Thm. 4 . The proof uses the same family of languages used by Bousquet and Löding [9] to prove an exponential translation may be required when going from deterministic Büchi automata to SUBAs. The family they proposed is given by $L_n = \Sigma^{n-1} a \Sigma^\omega$ where $\Sigma = \{a, b\}$. A FDFA for L_n has the same structure as a DFA for $\Sigma^{n-1} a \Sigma^*$, namely states

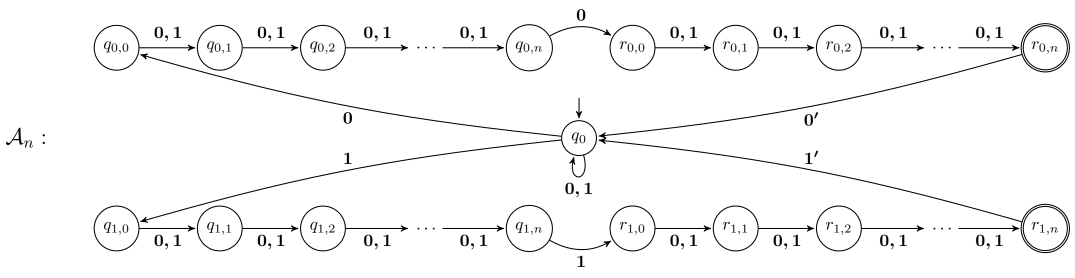


Figure 2 A SUBA with $4n + 5$ states for L_n used in the proof of Thm. 5 for which any DFA to recognize $(L_n)_\$$ or its reverse requires at least 2^n states.

$q_0, q_1, \ldots, q_n, q_{n+1}$ where q_i transits with a or b to q_{i+1} for $0 \leq i < n$, q_n transits to q_{n+1} with a and q_{n+1} transits to itself with a or b. The progress automata for all states q_i for $i \leq n$ is empty, and the progress automaton for state q_{n+1} is the one state accepting DFA. To build a DwPA for L_n we can take the same structure and the coloring $\kappa(q_{n+1}) = 2$ and $\kappa(q_i) = 1$ for any $0 \leq i \leq n$. ◀

5.2 Comparison to DFAs for $(L)_\$$ or Its Reverse

One may notice that the families of languages used in the comparisions of SUBA with DMA, DBA, FDFA and DwPA are similar to the families of languages used to show that a DFA for the reverse of a language L can be exponentially bigger or exponentially smaller than a DFA for L itself. This has to do with the fact that SUBAs are backward deterministic. We thus asked ourselves whether there exists a family of languages $\{L_n\}$ that a SUBA can characterize with polynomially in n many states, while both a DFA for $(L_n)_\$$ and a DFA for its reverse, $(L_n)^r_\$$, would require exponentially many states. The answer is positive.

▶ **Theorem 5.** *There exists a family $\{L_n\}$ of ω-languages such that L_n is recognized by a SUBA of $4n + 5$ states, while any DFA to recognize $(L_n)_\$$ or its reverse requires at least 2^n states.*

Proof. We define a family of ω-languages over the alphabet $\{0, 1, 0', 1'\}$ by

$$L_n = \Big((0+1)^* \big(0(0+1)^n 0(0+1)^n 0' + 1(0+1)^n 1(0+1)^n 1'\big)\Big)^\omega,$$

for all positive integers n. We define an NBA $\mathcal{A}_n$ of $4n + 5$ states to accept L_n (depicted in Fig. 2), and show that it is a SUBA. It is not difficult to verify that $\mathcal{A}_n$ accepts L_n. To see that $\mathcal{A}_n$ is a SUBA, note that any final run on an ω-word w must pass infinitely often through $r_{0,n}$ or $r_{1,n}$. After the former, the only possible symbol is $0'$, and after the latter, the only possible symbol is $1'$, and these symbols can occur nowhere else. Because the transition function of $\mathcal{A}_n$ is reverse deterministic, this completely determines the sequence of states traversed in a final run on w.

Next we show that a DFA to accept $(L_n)_\$$ must have at least 2^n states. Assume to the contrary that there is a DFA $\mathcal{M}$ with fewer than 2^n states that accepts this language. Then there exist two different strings $u, v \in \{0, 1\}^n$ that reach the same state q of $\mathcal{M}$ from the initial state. Without loss of generality, there exist $w, u_1, v_1 \in \{0, 1\}^*$ such that $u = w0u_1$ and $v = w1v_1$. Let x be any element of $\{0, 1\}^*$ with length $n - |u_1|$. Then $ux = w0u_1x$ and $vx = w1v_1x$ reach the same state of $\mathcal{M}$ from the initial state. However, $w0u_1x(0u_1x0'0u_1x)^\omega$ is an element of L_n while $w1v_1x(0u_1x0'0u_1x)^\omega$ is not, so $ux\$0u_1x0'0u_1x \in (L_n)_\$$, while $vx\$0u_1x0'0u_1x \notin (L_n)_\$$, a contradiction because these two strings must reach the same state of $\mathcal{M}$ from its initial state.

Finally we show that a DFA to accept $(L_n)^r_\$$, the reverse of $(L_n)_\$$ must have at least 2^n states. Assume that there is a DFA $\mathcal{M}^r$ of fewer than 2^n states that accepts the language $(L_n)^r_\$$. There exist two different strings $u, v \in \{0,1\}^n$ that reach the same state of M^r from its initial state, and strings $w, u_1, v_1 \in \{0,1\}^*$ such that $u = w0u_1$ and $v = w1v_1$. Let $x, y \in \{0,1\}^*$ be arbitrary strings of lengths $|x| = n - |u_1|$ and $|y| = n - |w|$. Consider the two strings

$$z_0 = w0u_1x00'y = ux00'y, \text{ and } z_1 = w1v_1x00'y = vx00'y,$$

which reach the same state of M^r from its initial state. Considering the reverses of these two strings, $z_0^r = y^r0'0x^ru_1^r0w^r$ is a possible period of L_n, that is,

$$0x^ru_1^r0w^r(z_0^r)^\omega \in L_n, \text{ but } 0x^ru_1^r0w^r(z_1^r)^\omega \notin L_n.$$

Thus, $z_0\$w0u_1x0 \in (L_n)^r_\$$ while $z_1\$w0u_1x0 \notin (L_n)^r_\$$, a contradiction because z_0 and z_1 reach the same state of $\mathcal{M}^r$ from its initial state. ◀

6 Multiplicity Automata

A multiplicity automaton represents a function f mapping finite strings Σ^* to elements of a field $\mathcal{K}$. The definitions are formulated using vectors and matrices and their products over $\mathcal{K}$. A *multiplicity automaton* of dimension d is specified as a tuple $\mathcal{A} = (\Sigma, v_I, \{\mu_\sigma\}_{\sigma\in\Sigma}, v_F)$, where Σ is the input alphabet, $v_I \in \mathcal{K}^d$ is the initial state, for each symbol $\sigma \in \Sigma$, $\mu_\sigma \in \mathcal{K}^{d\times d}$ is the transition map on σ, and $v_F \in \mathcal{K}^d$ is the output map. To specify the function f computed by $\mathcal{A}$ we first define a function μ from Σ^* to $\mathcal{K}^{d\times d}$ inductively as follows. For the empty string ε, $\mu(\varepsilon)$ is the $d \times d$ identity matrix. Given $\sigma \in \Sigma$ and $w \in \Sigma^*$, $\mu(\sigma w) = \mu_\sigma \cdot \mu(w)$, where $\cdot$ denotes matrix product. Thus, for a word $w = \sigma_1\sigma_2\cdots\sigma_n$, $\mu(w) = \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$. Then the value output by $\mathcal{A}$ on input word w is given by

$$f(w) = v_I^\top \mu(w) v_F,$$

where the vectors v_I and v_F from $\mathcal{K}^d$ are interpreted as $d \times 1$ column vectors.

Multiplicity automata have many useful properties. Assume that the arithmetic operations in the field $\mathcal{K}$ take one step. Then computing $f(w)$ requires computing the product of $|w|$ square matrices and two vectors of dimension d and takes time polynomial in $|w|$ and d. Thon and Jaeger [32] (who use the term *(linear) sequential systems* for multiplicity automata) give polynomial time algorithms to find a basis for the state space of a multiplicity automaton, to minimize a multiplicity automaton and to test two multiplicity automata for equivalence. The algorithm to find a basis for the states also yields a shortest string w (if any) that is not mapped to 0, and shows that such a string must have length less than the dimension of the automaton. This is because if the output is 0 on all the basis elements, the function is identically 0. Given multiplicity automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of dimensions d_1 and d_2 computing f_1 and f_2, there are multiplicity automata for the sum $f_1 + f_2$ (of dimension $d_1 + d_2$) and product $f_1 \cdot f_2$ (of dimension $d_1 \cdot d_2$) that may be constructed in polynomial time.

We consider the special case of multiplicity automata over the Galois Field $\mathcal{K} = \mathrm{GF}(2)$ of the two elements $\{0,1\}$, in which addition is defined modulo 2. These are termed *mod-2-multiplicity automata*, abbreviated as mod-2-MAs. The outputs of a mod-2-MA $\mathcal{A}$ are either 0 or 1, so we may consider them as language acceptors by defining $[\![\mathcal{A}]\!]$ to be the set of elements of Σ^* mapped to 1 by $\mathcal{A}$. We next show how to convert a mod-2-MA to an equivalent DFA.

$$\mathcal{M}_1 = (\{a,b\}, (1,0,0), \{\mu_a, \mu_b\}, (1,1,0)^\top)$$

$$\text{where } \mu_a = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \text{ and } \mu_b = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

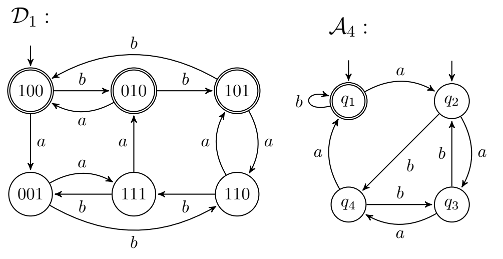


Figure 3 Left: a mod-2-MA $\mathcal{M}_1$ of dimension 3. Middle: the DFA $\mathcal{D}_1$ constructed from the mod-2-MA $\mathcal{M}_1$ according to Lemma 6. Right: the UFA $\mathcal{A}_4$ from Denis et al. [19].

▶ **Lemma 6.** *Let $\mathcal{A} = (\Sigma, v_I, \{\mu_\sigma\}_{\sigma\in\Sigma}, v_F)$ be a mod-2-MA of dimension n. There exists a DFA $\mathcal{A}'$ with at most 2^n states such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$.*

Proof. For $\mathcal{A}'$ the set of states is all row vectors $v \in \{0,1\}^n$ and the initial state is $v_I^\top$. For states v_1 and v_2 there is a transition from v_1 to v_2 on σ if and only if $v_2 = v_1\mu_\sigma$. The set of final states is all states v such that the inner product of v and $v_F^\top$ is 1. Then the definition of the output of $\mathcal{A}$ guarantees that for all $w \in \Sigma^*$, the output of $\mathcal{A}$ on w is 1 if and only if $\mathcal{A}'$ accepts w. ◀

As an example of this construction, consider the mod-2-MA $\mathcal{M}_1$ of dimension 3 given in the left of Fig. 3. The equivalent DFA $\mathcal{D}_1$ constructed from $\mathcal{M}_1$ as described in the proof of Lemma 6 is shown in the middle of Fig. 3, with unreachable states omitted. To see that the blowup in this conversion is inevitable, let's make some connection to NFAs and UFAs first.

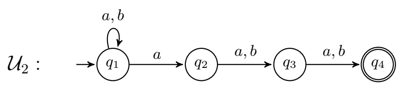
The language accepted by $\mathcal{M}_1$ and $\mathcal{D}_1$ is the same as the language accepted by the UFA called $\mathcal{A}_4$ by Denis et al. [19], shown on the right in Fig. 3, which can be verified by determinizing $\mathcal{A}_4$ and comparing with $\mathcal{D}_1$. Note that no NFA of fewer than 4 states can accept this language, because it must ensure that a^k is accepted if and only if k is 0 or 1 modulo 4. Thus, a mod-2-MA may be more concise than the smallest equivalent NFA, an issue we consider further below. For the reverse direction, we have the following, observed in Example 2.3 by Beimel et al. [8].

▶ **Lemma 7.** *Let $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$ be an NFA of n states. There exists a mod-2-MA $\mathcal{A}'$ of dimension n such that for every $w \in \Sigma^*$, the output of $\mathcal{A}'$ on input w is 1 if and only if the number of accepting paths for w in $\mathcal{A}$ is odd.*

Proof. Suppose the states of $\mathcal{A}$ are $Q = \{q_1, q_2, \ldots, q_n\}$. We define a mod-2-MA $\mathcal{A}'$ of dimension n as follows. For vectors in $\{0,1\}^n$, dimension i represents state q_i for $i = 1, 2, \ldots, n$. The vector v_I is the characteristic vector of Q_0, the vector v_F is the characteristic vector of F, and for each $\sigma \in \Sigma$, $[\mu_\sigma]_{i,j} = 1$ if and only if $(q_i, \sigma, q_j) \in \Delta$. An inductive proof shows that $[\mu(w)]_{i.j} = 1$ if and only if the number of paths on w from q_i to q_j is odd. Multiplying on the left by the transpose of v_I selects the paths starting at an initial state, and multiplying on the right by v_F adds up the results for the final states, giving 1 if and only if the total number of accepting paths on input w in $\mathcal{A}$ is odd. ◀

In the case of a UFA, for each $w \in \Sigma^*$ the number of accepting paths is either 0 or 1, immediately implying the following fact, also observed by Beimel et al.

▶ **Corollary 8.** *For any UFA $\mathcal{A}$ of n states, there exists a mod-2-MA $\mathcal{A}'$ of dimension n such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$.*



$$\mathcal{M}_2 = (\{a, b\}, (1, 0, 0, 0), \{\mu_a, \mu_b\}, (0, 0, 0, 1)),$$

$$\text{where } \mu_a = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \mu_b = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 4 The UFA $\mathcal{U}_2$ accepting $\Sigma^* a\Sigma\Sigma$ for $\Sigma = \{a, b\}$ and the mod-2-MA for the same language constructed according to Lemma 7.

As an example of this construction, consider the language $\Sigma^* a\Sigma\Sigma$, consisting of all strings over $\Sigma = \{a, b\}$ with an a as the third symbol from the end. This is accepted by the UFA $\mathcal{U}_2$ of 4 states, shown in Fig. 4 on the left. The mod-2-MA $\mathcal{M}_2$ of dimension 4 constructed from $\mathcal{U}_2$ according to Lemma 7 is given on the right of Fig. 4.

► Remark 9. In the case of mod-2-MAs, the results listed above for general multiplicity automata give polynomial time algorithms to minimize the number of dimensions, and to test equivalence, emptiness and universality, as well as polynomial time constructions for the symmetric difference (by addition), complement (by adding the constant 1), intersection (by multiplication), and union (by intersection and complement) of two given automata. If the language is nonempty, a shortest accepted word may be found in polynomial time and has length less than the number of dimensions (see [32]).

Beimel et al. [8] consider the problem of learning a multiplicity automaton computing a function f using suitably generalized equivalence and membership queries (see Section 3 for the basic definitions of learning with queries.) In particular, the answer to a membership query on word $w \in \Sigma^*$ is the value of $f(w) \in \mathcal{K}$, and the answer to an equivalence query with a multiplicity automaton $\mathcal{H}$ is either "yes" or a counterexample, that is, a word $w \in \Sigma^*$ such that the output of $\mathcal{H}$ on w is not equal to $f(w)$. Beimel et al. give an algorithm to learn a multiplicity automaton in time polynomial in the rank of the Hankel matrix of f (which is bounded above by the dimension of the multiplicity automaton) and the length of the longest counterexample. In contrast to $\mathbf{L}^*$, in which rows of the observation table that are unequal become distinct states, in the learning algorithm for multiplicity automata, a row becomes a new basis vector only if it is linearly independent of the existing basis vectors.

Given that DFAs, UFAs, NFAs and mod-2-MAs all accept the class of regular languages, an important distinction between them is succinctness, the number of states in the smallest automaton to accept a given regular language. An NFA, UFA or mod-2-MA of n states can be converted to an equivalent DFA of at most 2^n states. DFAs are UFAs, which are NFAs, so NFAs are at least as succinct as UFAs, which are at least as succinct as DFAs. By Corollary 8, each UFA can be converted to a mod-2-MA of the same size, so mod-2-MAs are at least as succinct as UFAs and DFAs. The family of languages L_n given by $(a+b)^*a(a+b)^n$ shows that NFAs, UFAs and mod-2-MAs may be exponentially more succinct than DFAs.

Schmidt [30] considers the family of complements of the languages L_n given by $\{ww \mid w \in \{0,1\}^n\}$, and shows that the complement of L_n can be accepted by an NFA of $O(n^2)$ states, but requires at least 2^n states for a UFA. His argument is that the rank of the binary matrix $F(x, y) = xy \notin L_n$ where $x, y \in \{0,1\}^n$ is at least 2^n, and therefore that a UFA must have at least 2^n states. This also shows that a mod-2-MA for the complement of L_n must have dimension at least 2^n. This leaves the question of whether mod-2-MAs may be non-polynomially more concise than UFAs or NFAs. Here we consider the comparison of the sizes of NFAs and mod-2-MAs over a unary alphabet.

▶ **Lemma 10.** *There is a family of unary languages $\{L_n\}$ such that the smallest NFA that accepts L_n has size $O(n^2)$ while the smallest mod-2-MA that accepts L_n has dimension at least $2^{\Theta(n/\log n)}$.*

Proof. Given n, let $p_1, p_2, \ldots, p_\ell$ denote the primes less than or equal to n and let $P(n)$ denote their product. Define L_n to contain all words a^t such that t is not a positive integer multiple of $P(n)$. We define an NFA $\mathcal{A}_n$ to accept L_n as follows. The states are $Q = \{q_0\} \cup \{r_{i,j} \mid 1 \leq i \leq n,\ 0 \leq j < p_i\}$, where q_0 is the only initial state and all states are final except for those in $\{r_{i,j} \mid 1 \leq i \leq k, j = p_i - 1\}$. There are transitions on a from q_0 to each $r_{i,0}$ and from each $r_{i,j}$ to $r_{i,j+1}$, where the addition is modulo p_i. Then ε is accepted, and a^t is rejected only if t is a positive integer multiple of each p_i, that is, a positive integer multiple of $P(n)$, so $\mathcal{A}_n$ accepts L_n and has $O(n^2)$ states.

Let $\mathcal{A}'$ be a mod-2-MA of dimension N accepting L_n. Then there is a mod-2-MA of dimension $N+1$ accepting the complement of L_n (Remark 9). But the shortest word in the complement of L_n has length $P(n)$, so $P(n) < N + 1$. Because the number of primes less than or equal to n is $\Theta(n/\log n)$ and each prime is at least 2, the lower bound follows. ◀

For more information on sizes of finite automata accepting unary languages, see the paper of Chrobak [16, 17], which gives more refined bounds.

In the other direction, we prove a conditional lower bound in the following lemma. A *Mersenne prime* is a prime of the form $2^d - 1$ for some positive integer d. Unfortunately, it is unknown whether there are infinitely many Mersenne primes. For this paper, a *shift register sequence* of dimension d is an infinite periodic sequence $\{a_n\}$ of bits defined by initial conditions $a_i = b_i$ for $i = 0, 1, \ldots, d-1$ and a linear recurrence

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d},$$

for all $n \geq d$, where each $c_i \in \{0, 1\}$ and the addition is modulo 2. The maximum possible minimum period of a shift register sequence is $2^d - 1$, and it is known that for every positive integer d there are shift register sequences of maximum period. These are known as *maximal length* or *pseudo noise* sequences. A maximal length sequence has 2^{d-1} ones and $2^{d-1} - 1$ zeros in the period. Golomb's book [23] is a definitive reference for shift register sequences. An example of a maximal length sequence of dimension 4 is given by $a_0 = 0$, $a_1 = 0$, $a_2 = 0$, $a_3 = 1$ and for all $n \geq 4$, $a_n = a_{n-3} + a_{n-4} \pmod 2$. This recurrence generates a sequence with period 000100110101111.

▶ **Lemma 11.** *If there are infinitely many Mersenne primes then there is a family of unary languages L_d such that for infinitely many values of d, L_d is accepted by a mod-2-MA of dimension d but no NFA of fewer than $2^d - 1$ states accepts L_d.*

Proof. Suppose the unary alphabet is $\Sigma = \{\sharp\}$. For a language $L \subseteq \Sigma^*$ we may define the infinite sequence χ_n^L where $\chi_n^L = 1$ if $\sharp^n \in L$ and 0 otherwise. Given a maximal length shift register sequence $\{a_n\}$ of dimension d, there exists a mod-2-MA $\mathcal{A}$ of dimension d such that the language L accepted by $\mathcal{A}$ has $\chi_n^L = a_n$ for all $n \geq 0$. Such a mod-2-MA $\mathcal{A}$ can be constructed as follows. Let $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d}$, with initial conditions $a_i = b_i$ for $0 \leq i < d$, be the linear recurrence used to define the given maximal length shift register sequence. Then let the mod-2-MA $\mathcal{M}$ be the tuple

$$(\{\sharp\}, (b_0, \ldots, b_{d-1}), \{\mu_\sharp\}, (1, 0, \ldots, 0)^\top),$$

where the matrix $\mu_\sharp$ is as prescribed on the right.

$$\mu_\sharp = \begin{pmatrix} 0 & 0 & \dots & 0 & c_d \\ 1 & 0 & \dots & 0 & c_{d-1} \\ 0 & 1 & \dots & 0 & c_{d-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_1 \end{pmatrix}$$

It follows that $(\chi^L_m, \chi^L_{m+1}, \dots, \chi^L_{m+d-1}) \cdot \mu_\sharp$ is equal to $(\chi^L_{m+1}, \chi^L_{m+2}, \dots, \chi^L_{m+d})$, for any $m \geq 0$, and therefore, for any $n \geq 0$, the value of χ^L_n is equal to $(b_0, b_1, \dots, b_{d-1}) \cdot (\mu_\sharp)^n \cdot (1, 0, \dots, 0)^\top$.

Let $\mathcal{A}'$ be an NFA accepting L corresponding to a maximal length shift register sequence of dimension d. Then $\mathcal{A}'$ must contain at least one reachable cycle of states with at least one final state. The length of that cycle must not be 1 or relatively prime to $2^d - 1$, or else positions of the period that should be 0's will eventually be assigned 1. More specifically, since the sequence is not constant and since the period of the shift register sequence is $2^d - 1$, there exist $m_1, m_2 < 2^d$ such that for all ℓ the value of $a_{\ell\cdot(2^d-1)+m_1}$ is 0 and the value of $a_{\ell\cdot(2^d-1)+m_2}$ is 1. On the other hand if there is a cycle of length p in $\mathcal{A}'$, then for some $k \geq 0$, the value of $a_{k+j_1\cdot p}$ is equal to $a_{k+j_2\cdot p}$, for all $j_1, j_2 \geq 0$. It can be shown that if p is coprime to $2^d - 1$, then there exist $z_1, z_2, x_1, x_2 \geq 0$, such that $x_1 \cdot (2^d - 1) + m_1$ is equal to $k + z_1 \cdot p$ and $x_2 \cdot (2^d - 1) + m_2$ is equal to $k + z_2 \cdot p$, a contradiction. Thus p must not be coprime to $2^d - 1$. Therefore, if $2^d - 1$ is prime, that is, if $2^d - 1$ is a Mersenne prime, p must be a multiple of $2^d - 1$ and as a result $\mathcal{A}$ must have at least $2^d - 1$ states. ◀

7 Learning SUBAs in Polynomial Time With MQs and Non-Proper EQs

In contrast to the negative result in Section 4 for learning NBAs, we show that SUBAs can be learned in polynomial time using MQs and non-proper EQs. Since these two classes of automata both accept all the regular ω-languages, a key difference is in the succinctness of the representation of a regular ω-language by an NBA versus a SUBA.

▶ **Theorem 12.** *There is a polynomial time algorithm to learn all regular ω-languages, when the target language is represented by a SUBA, using membership queries and non-proper equivalence queries (using mod-2-MAs to represent hypotheses).*

The existence of the learning algorithm is established by the following two results.

▶ **Lemma 13** (Bousquet and Löding [9]). *Let $\mathcal{A}$ be a SUBA of n states accepting the language $L = [\![\mathcal{A}]\!]$. Then there is a UFA of $O(n^2)$ states for the language $(L)_\$$.*

Bousquet and Löding prove that there are polynomial time algorithms to determine containment and equivalence of two regular ω-languages represented by SUBAs. Their proof gives a construction that takes a SUBA $\mathcal{A}$ of n states accepting a language L and produces a UFA of at most $n + 2n^2$ states that accepts $(L)_\$$.

▶ **Lemma 14** (Beimel et al. [8]). *There is a polynomial time algorithm to learn UFAs with membership and non-proper equivalence queries (using mod-2-MAs to represent hypotheses).*

Beimel et al. [8] give an algorithm to learn multiplicity automata over a field $\mathcal{K}$ using (generalized) MQs and EQs that runs in time polynomial in the dimension of the target automaton and the length of the longest counterexample. They remark (p. 519) that their results imply a polynomial time algorithm for learning UFAs. Specifically, combining their algorithm with the fact that UFAs can be transformed to mod-2-MAs of the same size (Corollary 8) yields the lemma.

Proof of Theorem 12. Let $\mathcal{A}$ be a SUBA of n states with input alphabet Σ accepting the language L. We assume $\$ \notin \Sigma$, and let $\Sigma_\$ = \Sigma \cup \{\$\}$. For a MQ to L, the input is a string $u\$v$ and the answer is 1 or 0 according to whether $u(v)^\omega \in L$, that is, whether $u\$v \in (L)_\$$. For a non-proper EQ to L, the input is a mod-2-MA $\mathcal{H}$ over the alphabet $\Sigma_\$$ and the answer is "yes" if the language accepted by $\mathcal{H}$ is precisely $(L)_\$$. Otherwise, the answer is a counterexample $w \in (\Sigma_\$)^*$ such that $\mathcal{H}$ accepts w and $w \notin (L)_\$$ or $\mathcal{H}$ rejects w and $w \in (L)_\$$.

The learning algorithm of Beimel et al. may use these EQs and MQs to L to learn a mod-2-MA for the language $(L)_\$$. Because $(L)_\$$ is accepted by a UFA of $O(n^2)$ states, and therefore by a mod-2-MA of dimension $O(n^2)$ (Corollary 8), the learning algorithm runs in time polynomial in n and the length of the longest counterexample. ◀

In contrast to the negative result in Section 4 for the representation of regular ω-languages by NBAs, we have the following corollary.

▶ **Corollary 15.** *The class of ω-regular languages is polynomially predictable with membership queries when the target language is represented by a SUBA.*

Proof. By the discussion preceding Theorem 1, it is sufficient to note that there is a polynomial time algorithm that takes as inputs a mod-2-MA $\mathcal{H}$ and a finite word w and decides whether $\mathcal{H}$ accepts w. ◀

8 Discussion

We have shown that there is a polynomial time algorithm to learn strongly unambiguous Büchi automata (SUBAs) using membership queries and non-proper EQs, where the EQs use mod-2-MAs to represent $(L)_\$$. This implies that SUBAs are polynomially predictable with membership queries. By contrast, we have shown that under plausible cryptographic assumptions, general NBAs are not polynomially predictable with MQs. In applications, careful thought should be given to the choice of representations, considering both succinctness and learnability.

Given that the standard translation of a linear temporal logic (LTL) formula to an NBA yields a SUBA [34, 35], it is natural to ask what our results imply about the learnability of LTL formulas. The first step of the standard translation of an LTL formula ϕ constructs a state set consisting of all subsets of subformulas of ϕ, so the constructed SUBA may be of size exponential in the size of ϕ. Thus an algorithm that runs in time polynomial in the size of the SUBA does not avoid this exponential blow up. However, the difficulty of learning LTL formulas may be unavoidable: LTL formulas are at least as expressive and succinct as Boolean formulas, and the results of Angluin and Kharitonov [7] show that under the same cryptographic assumptions in Theorem 1, Boolean formulas are not polynomially predictable with membership queries.

Several avenues of research are suggested by our results. Can the equivalence queries used by the learning algorithm of Theorem 12 be modified to use SUBAs, or at least NBAs, as hypotheses? Farzan et al. [21] show how to convert the DFA hypotheses used by their algorithm to NBAs for equivalence queries, a method that does not extend to mod-2-MAs. The question of how different two minimal SUBAs for the same language can be appears to be open; note that the SUBAs in Fig. 1 are isomorphic if we permit a permutation of the alphabet symbols. Given the useful properties of mod-2-MAs, perhaps the idea of using mod-2-MAs for $(L)_\$$ as a general representation of regular ω-languages L should be explored. On a minor point, is there a proof that mod-2-MAs may be exponentially more succinct than NFAs without the assumption that there are infinitely many Mersenne primes? Finally, none of our results bear on the open questions of whether deterministic Büchi automata, deterministic parity automata or deterministic Muller automata are learnable in polynomial time with equivalence and membership queries.

References

1 G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 4–16, 2002.

2 D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

3 D. Angluin, U. Boker, and D. Fisman. Families of DFAs as Acceptors of omega-Regular Languages. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 11:1–11:14, 2016.

4 D. Angluin and D. Fisman. Learning Regular Omega Languages. In *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, pages 125–139, 2014.

5 D. Angluin and D. Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.

6 D. Angluin and D. Fisman. Regular omega-Languages with an Informative Right Congruence. In *GandALF*, volume 277 of *EPTCS*, pages 265–279, 2018.

7 D. Angluin and M. Kharitonov. When Won't Membership Queries Help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

8 A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning Functions Represented As Multiplicity Automata. *J. ACM*, 47(3):506–530, May 2000.

9 N. Bousquet and C. Löding. Equivalence and Inclusion Problem for Strongly Unambiguous Büchi Automata. In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, pages 118–129, 2010. doi:10.1007/978-3-642-13089-2_10.

10 J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *International Congress on Logic, Methodology and Philosophy*, pages 1–11. Stanford Univ. Press, 1962.

11 Hugues C., 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.

12 O. Carton and M. Michel. Unambiguous Büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003. doi:10.1016/S0304-3975(02)00618-7.

13 G. Chalupar, S. Peherstorfer, E. Poll, and J. de Ruiter. Automated Reverse Engineering using Lego®. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA, August 2014. USENIX Association.

14 M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the Language of Error. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, pages 114–130, 2015.

15 C. Y. Cho, D. Babic, E. C. R. Shin, and D. Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 426–439, 2010. doi:10.1145/1866307.1866355.

16 M. Chrobak. Finite Automata and Unary Languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.

17 M. Chrobak. Errata to: Finite Automata and Unary Languages. *Theor. Comput. Sci.*, 47(3):149–158, 2003.

18 J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning Assumptions for Compositional Verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '03, pages 331–346, Berlin, Heidelberg, 2003. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1765871.1765903`.

19 F. Denis, A. Lemay, and A. Terlutte. Residual Finite State Automata. *Fundam. Inform.*, 51:339–368, January 2002.

20 E. E. Stearns and B. Hunt. On the Equivalence and Containment Problems for Unambiguous Regular Expressions, Regular Grammars and Finite Automata. *SIAM J. Comput.*, 14:598–611, August 1985.

21 A. Farzan, Y-F. Chen, E.M. Clarke, Y-K. Tsay, and B-Y. Wang. Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In *TACAS*, pages 2–17, 2008.

22 D. Fisman. Inferring regular languages and ω-languages. *J. Log. Algebr. Meth. Program.*, 98:27–49, 2018.

23 S. W. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills, CA, USA, 1981.

24 P. Habermehl and T. Vojnar. Regular Model Checking Using Inference of Regular Languages. *Electr. Notes Theor. Comput. Sci.*, 138(3):21–36, 2005.

25 O. Maler and A. Pnueli. On the Learnability of Infinitary Regular Sets. *Inf. Comput.*, 118(2):316–326, 1995.

26 O. Maler and L. Staiger. On Syntactic Congruences for Omega-Languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997.

27 T. Margaria, O. Niese, H. Raffelt, and B. Steffen. Efficient test-based model generation for legacy reactive systems. In *HLDVT*, pages 95–100. IEEE Computer Society, 2004.

28 W. Nam and R. Alur. Learning-Based Symbolic Assume-Guarantee Reasoning with Automatic Decomposition. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 170–185. Springer, 2006.

29 D. Peled, M. Y. Vardi, and M. Yannakakis. Black Box Checking. In *FORTE*, pages 225–240, 1999.

30 E. M. Schmidt. *Succinctness of Descriptions of Context-free, Regular and Finite Languages.* PhD thesis, Cornell University, Ithaca, NY, USA, 1978.

31 M. Schuts, J. Hooman, and F. W. Vaandrager. Refactoring of Legacy Software Using Model Learning and Equivalence Checking: An Industrial Experience Report. In *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*, pages 311–325, 2016.

32 M. R. Thon and H. Jaeger. Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *Journal of Machine Learning Research*, 16:103–147, 2015.

33 A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using Language Inference to Verify Omega-Regular Properties. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 45–60, 2005.

34 T. Wilke. Past, Present, and Infinite Future. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 95:1–95:14, 2016.

35 T. Wilke. Backward Deterministic Büchi Automata on Infinite Words. In *FSTTCS*, 2017.