Saturation Problems for Families of Automata

León Bohn
RWTH Aachen University, Germany

Yong Li
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Christof Löding
RWTH Aachen University, Germany

Sven Schewe
University of Liverpool, UK

Abstract

Families of deterministic finite automata (FDFA) represent regular ω-languages through their ultimately periodic words (UP-words). An FDFA accepts pairs of words, where the first component corresponds to a prefix of the UP-word, and the second component represents a period of that UP-word. An FDFA is termed saturated if, for each UP-word, either all or none of the pairs representing that UP-word are accepted. We demonstrate that determining whether a given FDFA is saturated can be accomplished in polynomial time, thus improving the known PSPACE upper bound by an exponential. We illustrate the application of this result by presenting the first polynomial learning algorithms for representations of the class of all regular ω-languages. Furthermore, we establish that deciding a weaker property, referred to as almost saturation, is PSPACE-complete. Since FDFAs do not necessarily define regular ω-languages when they are not saturated, we also address the regularity problem and show that it is PSPACE-complete. Finally, we explore a variant of FDFAs called families of deterministic weak automata (FDWA), where the semantics for the periodic part of the UP-word considers ω-words instead of finite words. We demonstrate that saturation for FDWAs is also decidable in polynomial time, that FDWAs always define regular ω-languages, and we compare the succinctness of these different models.

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

Keywords and phrases Families of Automata, automata learning, FDFAs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.146

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2506.13197`

Funding *León Bohn*: Supported by DFG grant LO 1174/7-1.
Yong Li: Supported by National Natural Science Foundation of China (Grant No. 62102407).
Sven Schewe: Supported by the EPSRC through projects EP/X03688X/1 and EP/X042596/1.

1 Introduction

Regular ω-languages (languages of infinite words) are a useful tool for developing decision procedures in logic that have applications in model checking and synthesis [6, 34]. There are many different formalisms for representing regular ω-languages, like regular expressions, automata, semigroups, and logic [33, 31]. In this paper, we study *families of automata* that represent regular ω-languages in terms of the ultimately periodic words that they contain. This is based on the fact that a regular ω-language is uniquely determined by the ultimately periodic words that it contains, which follows from results by Büchi [11] on the closure of regular ω-languages under Boolean operations (see also [13, Fact 1])). Ultimately periodic

© León Bohn, Yong Li, Christof Löding, and Sven Schewe;
licensed under Creative Commons License CC-BY 4.0
52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 146; pp. 146:1–146:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

words are of the form uv^ω for finite words u, v (where v is non-empty). For a regular ω-language L, [13] considers the language $L_\$$ of all finite words of the form $u\$v$ with $uv^\omega \in L$. They show that $L_\$$ is regular, and from a DFA for $L_\$$ one can construct in polynomial time a nondeterministic Büchi automaton for L. A similar approach was used independently by Klarlund in [22] who introduces the concept of *families of deterministic finite automata* (FDFA) for representing ω-languages, based on the notion of family of right congruences (FORC) introduced by Maler and Staiger [29]. Instead of using a single DFA, an FDFA consists of one so-called leading transition system, and so-called progress DFAs, one for each state of the leading transition system. A pair (u, v) of finite words (with v non-empty) is accepted if v is accepted by the progress DFA of the state reached by u in the leading transition system. As opposed to the $L_\$$ representation from [13], the FDFA model of [22, 29] only considers pairs (u, v) such that v loops on the state of the leading transition system that is reached by u, referred to as *normalized pairs*. So an FDFA for L is an FDFA that accepts all normalized pairs (u, v) with $uv^\omega \in L$.

Because there exist many learning algorithms for DFAs, these kinds of representations based on DFAs have received attention in the area of learning regular ω-languages [17, 3, 24, 9, 10, 26]. One problem is that methods for DFA learning might come up with automata that treat different pairs that define the same ultimately periodic word differently. So it might happen that a pair (u_1, v_1) is accepted while (u_2, v_2) is rejected, although $u_1v_1^\omega = u_2v_2^\omega$. In this case it is not clear anymore which ω-language is accepted. One can use the existential (nondeterministic) semantics and say that uv^ω is accepted if some pair representing uv^ω is accepted. But this does not necessarily define a regular ω-language (see [24, Example 2]). An FDFA is called *saturated* if it treats all normalized pairs representing the same ultimately periodic word in the same way (accepts all or rejects all). Additionally, we call an FDFA *fully saturated* if it treats all pairs representing the same ultimately periodic words in the same way (not only the normalized ones). The $u\$v$ representation corresponds to fully saturated FDFAs because a DFA for the language $L_\$$ can easily be turned into a fully saturated FDFA for L by taking the part before \$ as leading transition system, and the part after the \$ for defining the progress DFAs.

It has been shown in [2] that saturated FDFAs possess many good properties as formalism for defining ω-languages.[1] However, up to now the best known upper bound on the complexity for deciding if a given FDFA is (fully) saturated is PSPACE [2]. In this paper, we settle several questions concerning the complexity of saturation problems. As our first contribution we provide polynomial time algorithms for deciding saturation and full saturation of FDFAs, solving a question that was first posed more than three decades ago in [13] (see related work for existing work on the saturation problem). We also consider the property of almost saturation, which is weaker than saturation and thus allows for smaller FDFAs (Lemma 17), while ensuring that almost saturated FDFAs define regular ω-languages [25]. We show that this comes at a price, because deciding almost saturation is PSPACE-complete. Furthermore, we show that it is PSPACE-complete to decide whether a given FDFA defines a regular ω-language, solving a question that was raised in [12].

An overview of the models that appear in our paper and of our results is given in Figure 1. The different classes of FDFAs are shown on the left-hand side. The entries PTIME/PSPACE-c. indicate the complexity of deciding whether a given FDFA belongs to the respective class. As an application of the polynomial saturation algorithms, we provide

[1] The semantics of FDFAs in [2] and [22] are defined differently, however, saturated FDFAs define the same ω-languages in both models. We follow the definitions from [2].

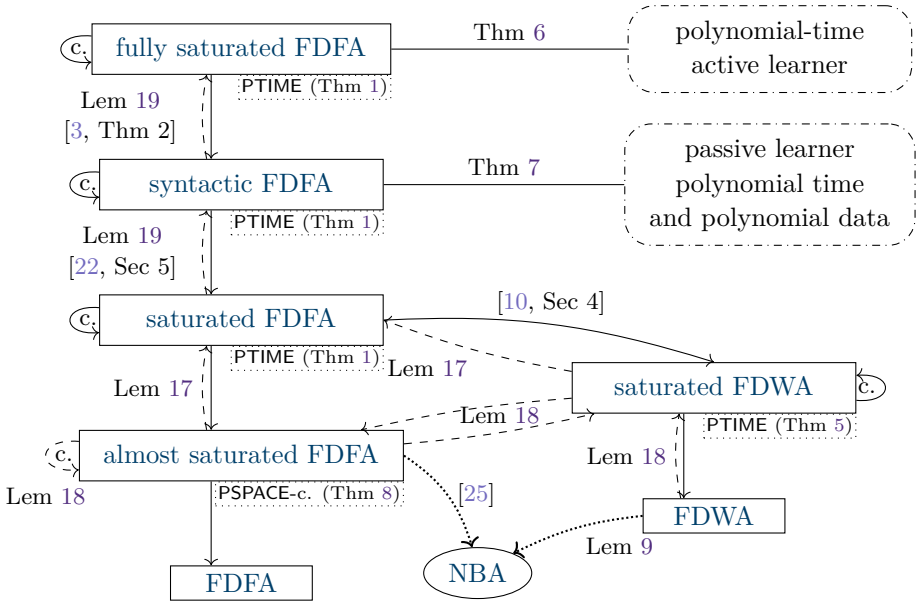

Figure 1 Overview of the properties and models considered in this paper. Solid arrows indicate translations that are possible without blowup, while dashed ones are used for translations with an exponential lower bound. Dotted arrows are constructions yielding automata, and ones labeled with "c." are complementation constructions. Solid lines are used to indicate a connection to learning. We provide more details on this diagram in the introduction.

the *first* polynomial learning algorithms for classes representing all regular ω-languages: a polynomial time active learner for fully saturated FDFAs, and a passive learner for syntactic FDFAs that is polynomial in time and data (see related work for existing work on learning ω-languages). The syntactic FDFA for L is the saturated FDFA with the least possible leading transition system (called $\mathfrak{L}^L$ in [22]). The size of the models that our algorithms can learn is, in general, incomparable to the size of deterministic parity automata (DPAs) ([2, Theorem 5.12] shows an exponential lower bound from saturated FDFAs to DPAs and the language family used for that is accepted also by small fully saturated FDFAs; an exponential lower bound from DPAs to syntactic FDFAs follows from the language family L_n used in Lemma 19, which is easily seen to be accepted by small DPAs).

For completing the picture of the FDFA landscape, we also give examples showing that complementing almost saturated FDFAs can cause an exponential blow-up (while it is trivial for saturated or fully saturated FDFAs), and for exponential size gaps between the different models. Exponential lower bounds are indicated by the dashed arrows in Figure 1, where the loops labeled "c." represent the complement. The solid arrows mean that there is a transformation that does not cause any blow-up.

Finally, we also consider the model of families of deterministic weak automata (FDWA), which is shown with two entries on the right-hand side of Figure 1. This is an automaton model for families of weak priority mappings (FWPM), which have been introduced in [10]. Syntactically, an FDWA is an FDFA in which the progress DFAs have the property that each strongly connected component (SCC) of states is either completely accepting or completely rejecting. This corresponds to the restriction made for deterministic weak (Büchi) automata (DWA); See, e.g., [27]. And indeed, the progress automata are not interpreted as DFAs accepting finite words, but as deterministic weak automata accepting periodic words: a

pair (u, v) is accepted if v^ω is accepted by the progress DWA of the state reached by u in the leading transition system, which means that v^ω ends up cycling in an accepting SCC. As for FDFAs, the property of saturation for FDWAs requires that either all normalized representations of an ultimately periodic word are accepted or none. (In order to not consider too many models, we restricted ourselves to existing ones and did not additionally introduce fully saturated and syntactic FDWAs.) We show that saturation can also be decided in polynomial time for FDWAs, which is easier to see than for FDFAs.

Concerning succinctness of FDWAs compared to FDFAs, it follows from results in [10] that saturated FDFAs can be transformed into saturated FDWAs by only adapting the acceptance mechanism and keeping the transition structure. We show that a translation in the other direction can cause an exponential blow-up, even when going to the more relaxed model of almost saturated FDFAs. Vice versa, we show that translating almost saturated FDFAs to saturated FDWAs also might cause an exponential blow-up.

In summary, our contributions are polynomial saturation algorithms for FDFAs and FDWAs, PSPACE-completeness results for deciding almost saturation and regularity of FDFAs, and some exponential lower bounds for transformations between different models.

Organization. The paper is structured as follows. We continue the introduction with a discussion of related work. In Section 2 we introduce the main definitions. In Section 3 we present the polynomial time saturation algorithms (Section 3.1), the learning algorithms (Section 3.2), and the PSPACE-completeness of almost saturation (Section 3.3). The regularity problem is shown to be PSPACE-complete in Section 4, and the exponential lower bounds for transformations between the models are presented in Section 5. We conclude in Section 6.

Related work on the saturation and regularity problems. The problem of saturation was first raised in the conclusion of [13] for the $L_\$$ representation of ultimately periodic words: "How can we decide efficiently that a rational language $K \subseteq A^*\$A^+$ is saturated by $\overset{\text{UP}}{\equiv}$?" Here, two words $u\$v$ and $u'\$v'$ are in relation $\overset{\text{UP}}{\equiv}$ if $uv^\omega = u'(v')^\omega$. A variation of the problem appears in [12] in terms of so called period languages: For an ω-language L, a finite word v is called a *period* of L if there is an ultimately periodic word of the form uv^ω in L, and $\mathsf{per}(L)$ denotes the set of all these periods of L. So $\mathsf{per}(L)$ is obtained from $L_\$$ by removing the prefixes up to (and including) the \$. It is shown that a language K of finite words is a period language iff K is closed under the operations of power, root, and conjugation, where $\mathsf{pow}(K) = \{v^k \mid v \in K,\ k \geq 1\}$, $\mathsf{root}(K) = \{v \mid \exists k > 0 :\ v^k \in L\}$, and $\mathsf{conj}(K) = \{v_2 v_1 \mid v_1 v_2 \in K\}$. Our notion of power-stable corresponds to closure under root and pow, and our notion of loopshift-stable corresponds to closure under conjugation. Given any $K \subseteq \Sigma^*$, the least period language containing K is $\mathsf{per}(L) = \mathsf{root}(\mathsf{pow}(\mathsf{conj}(K)))$. Then [12] poses three decision problems for a given regular language $K \subseteq \Sigma^*$: (1) Is K a period language? (2) Is $\mathsf{per}(K)$ regular? (3) Is $\mathsf{pow}(K)$ regular?

Problem (1) is a variation of the saturation problem for FDFAs that focuses solely on periods, independent of their prefixes. It is observed in [12] that Problem (1) is decidable without considering the complexity. Similarly, Problem (2) is a variation of the regularity problem that we study for FDFAs, again considering only periods independent of their prefixes. It is left open in [12] whether Problem (2) is decidable, but a reduction to Problem (3) is given. Notably, this reduction is exponential since it constructs a DFA for $\mathsf{root}(\mathsf{conj}(K))$. Later, Problem (3) was shown to be decidable in [18] without an analysis of the complexity. Our results in Section 4 give a direct proof that Problem (2) is PSPACE-complete.

The paper [7] defines cyclic languages as those that are closed under the operations of power, root, and conjugation, and gives a characterization of regular cyclic languages in terms of so called strongly cyclic languages. So the cyclic languages are precisely the period languages of [12]. Decidability questions are not studied in [7].

In [16], lasso automata are considered, which are automata accepting pairs of finite words. The representation basically corresponds to the $u\$v$ representation of [13] but without an explicit $ symbol, using different transition relations instead. It is easy to see that lasso automata and automata for the $u\$v$ representation are equivalent up to minor rewriting of the automata. The property of full saturation corresponds to the notion of bisimulation invariance in [16]: two lassos (u, v) and (u', v') are called bisimilar if $uv^\omega = u'(v')^\omega$, and a lasso automaton is bisimulation invariant if it accepts all pairs from a bisimulation class or none. Bisimulation invariance is characterized in [16] by the properties *circular* (power-stable in our terminology) and *coherent* (loopshift-stable in our terminology), [15, Theorem 2], [16, Section 3.3]. An Ω-automaton is defined as a circular and coherent lasso automaton, which corresponds to the property of full saturation for FDFAs. This property of a lasso automaton is shown to be decidable in [16, Theorem 18] without considering the complexity (since the decision procedure builds the monoid from a DFA, it is at least exponential). Another decision procedure is given in [14, Proposition 12] with a doubly exponential upper bound.

Finally, the saturation problem for FDFAs is explicitly considered in [2, Theorem 4.7], where it is shown to be in PSPACE by giving an upper bound (exponential in the size of the FDFA) on the size of shortest pairs violating saturation if the given FDFA is not saturated.

Related work concerning learning algorithms. There are some active and some passive learning algorithms for different representations of regular ω-languages, but all of them are either for subclasses of the regular ω-languages, or they are not polynomial time (for active learners) or not polynomial in the required data (for passive learners), where the complexity for learning a class $\mathcal{C}$ of representations is measured in a smallest representation of the target language from $\mathcal{C}$. The only known polynomial time active learner for regular ω-languages in the minimally adequate teacher model of [1] is for deterministic weak Büchi automata [28]. There are active learners for nondeterministic Büchi automata (NBA) [17, 24] and for deterministic Büchi and co-Büchi automata [26], but these are heuristics for the construction of NBA that are not polynomial in the target representations. The algorithm from [17] uses an active learner for DFAs for learning the $u\$v$ representation of [13] for regular ω-languages. Whenever the DFA learner asks an equivalence query, the DFA is turned into an NBA, and from a counter-example for the NBA a counter example for the DFA is derived. Our active learner uses the same principle but turns the DFA for the $u\$v$ representation into an FDFA and checks whether it is fully saturated.

There is a polynomial time active learner for deterministic parity automata [30], but this algorithm uses in addition to membership and equivalence queries so-called loop-index queries that provide some information on the target automaton, not just the target language. So this algorithm does not fit into the minimally adquate teacher model from [1]. The paper [3] presents a learning algorithm for FDFAs. But it turns out that this is not a learning algorithm for regular ω-languages: The authors make the assumption that the FDFAs used in equivalence queries define regular ω-languages (beginning of Section 4 in [3]) while they only provide such a semantics for saturated FDFAs. So their algorithm requires an oracle that returns concrete representations of ultimately periodic words as counter-examples for *arbitrary* FDFAs, which is not an oracle for regular ω-languages. Our algorithms solve this problem by first using the polynomial time saturation check, making sure that we only submit saturated FDFAs to the equivalence oracle.

Concerning passive learners, the only polynomial time algorithms that can learn regular ω-languages in the limit from polynomial data are for subclasses that make restrictions on the canonical Myhill/Nerode right-congruence of the potential target languages. The algorithm from [4] can only learn languages for which the minimal automaton has only one state per Myhill/Nerode equivalence class (referred to as IRC languages for "informative right congruence"). The algorithm from [8] can also learn languages beyond that class but there is no characterization of the class of languages that can be inferred beyond the IRC languages. The algorithm in [9] infers deterministic Büchi automata (DBA) in polynomial time but the upper bound on the size of characteristic samples for a DBA-recognizable language L is exponential in a smallest DBA for L, in general. The algorithm in [10] generalizes this to deterministic parity automata (DPA) and can infer a DPA for each regular ω-language, but a polynomial bound for the size of characteristic samples is only given for languages accepted by a DPA that has at most k states per Myhill/Nerode class for a fixed k.

Related work concerning the model of FDWA. The paper [19] considers a model of FDFA with so-called duo-normalized acceptance and mentions connections to the model FWPM from [10] in a few places, without going into details of the connection. Using results from [10], it is not very hard to show that the models can be considered the same in the sense that they can be easily converted into each other without changing the transition structure. We give a proof of this in the full version of the paper.

2 Preliminaries

An alphabet is a finite, non-empty set Σ of symbols. We write Σ^* and Σ^ω to denote the set of finite and the set of infinite words over Σ, respectively. For a word w, we use $|w|$ to refer to its length, which is the number of symbols if w is finite, and ∞ otherwise. The empty word, ε, is the unique word of length 0 and we use Σ^+ for the set of finite, non-empty words, i.e. $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. We assume a linear order on the alphabet and consider the standard length-lexicographic ordering on finite words, which first orders words by length, and by lexicographic order among words of same length. An ω-word $w \in \Sigma^\omega$ is called *ultimately periodic* if it can be written as ux^ω with $u \in \Sigma^*$ and $x \in \Sigma^+$. We write $\mathsf{UP}(\Sigma)$ for the set of all ultimately periodic words over the alphabet Σ. For a word $x \in \Sigma^*$, we write $\sqrt{x}$ to denote its *root*, which is the unique minimal word u such that $x^\omega = u^\omega$.[2] For set $P \subseteq \Sigma^+$, we write $\omega\text{-}\mathsf{Pow}(P)$ to denote the set $\{x^\omega \mid x \in P\}$

A *transition system* (TS) is a tuple $\mathcal{T} = (\Sigma, Q, \delta, \iota)$, consisting of an input alphabet Σ, a non-empty set of states Q, a transition function $\delta : Q \times \Sigma \to Q$, and an initial state $\iota \in Q$. We define $\delta^* : Q \times \Sigma^* \to Q$ as the natural extension of δ to finite words, and overload notation treating $\mathcal{T}$ as the function mapping a finite word u to $\delta^*(\iota, u)$. We write $Q(\mathcal{T})$ for the set of states of a TS-like object $\mathcal{T}$, and assume that $Q(\mathcal{T})$ is prefix-closed subset of Σ^* containing for each state the length-lexicographically minimal word that reaches it. The *size* of $\mathcal{T}$, denoted $|\mathcal{T}|$, is the number of states. A set $S \subseteq Q$ of states is a *strongly connected component* (SCC) if S is non-empty, and $\subseteq$-maximal with respect to the property that for every pair of states $p, q \in S$ there is a non-empty word $x \in \Sigma^+$ such that $\delta^*(p, x) = q$. An equivalence relation

[2] Usually, $\sqrt{x}$ is defined to be the minimal u with $u^i = x$ for some $i > 0$. A simple application of the Theorem of Fine and Wilf shows that the two definitions coincide: If $u^i = x$ and $v^\omega = x^\omega$, then $v^{|x|} = u^{i|v|} =: y$. Since y has periods $|u|$ and $|v|$, by the theorem of Fine and Wilf, it also has period $gcd(|u|, |v|)$.

$\sim \subseteq \Sigma^* \times \Sigma^*$ is called right congruence if it preserves right concatenation, meaning $x \sim y$ implies $xz \sim yz$ for all $x, y, z \in \Sigma^*$. A TS $\mathcal{T}$ can be viewed as a right congruence $\sim_{\mathcal{T}}$ where $x \sim_{\mathcal{T}} y$ if $\mathcal{T}(x) = \mathcal{T}(y)$, and a right congruence $\sim$ can be viewed as a TS using the classes of $\sim$ as states and adding for each class u and symbol $a \in \Sigma$ the transition $[u]_\sim \xrightarrow{a} [ua]_\sim$.

A *deterministic finite automaton* (DFA) $\mathcal{D} = (\Sigma, Q, \delta, \iota, F)$ is a TS with a set $F \subseteq Q$ of final states. By replacing δ with a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, we obtain a *nondeterministic finite automaton* (NFA), so every DFA is also an NFA. The language accepted by an NFA $\mathcal{N}$, denoted $L_*(\mathcal{N})$, consists of all words that can reach a state in F.

A *deterministic Büchi automaton* (DBA) $\mathcal{B}$ is syntactically the same as a DFA. It accepts an ω-word $w \in \Sigma^\omega$ if the run of $\mathcal{B}$ on w visits a final state from F infinitely often, and we write $L_\omega(\mathcal{B})$ for the language accepted by $\mathcal{B}$. If all states within a strongly connected component are either accepting, or all of them are rejecting, we call $\mathcal{B}$ *weak*. If we replace the transition function of a DBA with a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, we obtain a *nondeterministic Büchi automaton* (NBA), which accepts a word w if some run on w visits F infinitely often. The language of an NBA is the set of all words it accepts, and say that $L \subseteq \Sigma^\omega$ is a *regular ω-language* if it is the language of some NBA. It already follows from the results of Büchi [11] that if two ω-languages agree on the ultimately periodic words, then they must be equal. We call a set $L \subseteq \mathsf{UP}(\Sigma)$ of ultimately periodic words *UP-regular* if there exists some regular ω-language L' such that $L' \cap \mathsf{UP}(\Sigma) = L$.

A family of DFAs (*FDFA*) is a pair $\mathcal{F} = (\mathcal{T}, \mathcal{D})$ where $\mathcal{T}$ is a transition system called the *leading TS*, and $\mathcal{D}$ is an indexed family of automata, so for each state $q \in Q(\mathcal{T})$, $\mathcal{D}_q$ is a DFA. We overload notation writing $\mathcal{D}_u$ for the DFA $\mathcal{D}_{\mathcal{T}(u)}$ and call $\mathcal{D}_u$ a *progress DFA*. We always assume that if x and y lead to the same state in some $\mathcal{D}_u$, then also ux and uy lead to the same state in $\mathcal{T}$. This can always be achieved with a blow-up that is at most quadratic by taking the product of the progress DFA with the leading TS, and it makes arguments in some proofs (e.g. Lemma 4) simpler. An FDFA $\mathcal{F}$ *accepts* a pair $(u, x) \in \Sigma^* \times \Sigma^+$ if the DFA $\mathcal{D}_u$ accepts x, and we write $\mathsf{L}_{\mathsf{rep}}(\mathcal{F})$ to denote the set of all representations accepted by $\mathcal{F}$. A representation (u, x) is called *normalized (with regard to $\mathcal{F}$)* if ux and u lead to the same state in $\mathcal{T}$, and write $\mathsf{L}_{\mathsf{norm}}(\mathcal{F})$ for the set of normalized pairs accepted by $\mathcal{F}$. A family of deterministic weak automata (*FDWA*) $\mathcal{W} = (\mathcal{T}, \mathcal{B})$ is syntactically the same as an FDFA but it views the progress DFAs as DBAs, and additionally requires each DBA $\mathcal{B}_u$ to be weak. $\mathcal{W}$ accepts a pair (u, x) if x^ω is accepted by $\mathcal{B}_u$ viewed as a DBA. We write $\mathsf{L}_{\mathsf{rep}}(\mathcal{W})$ for the set of all representations accepted by $\mathcal{W}$ and denote by $\mathsf{L}_{\mathsf{norm}}(\mathcal{W})$ the restriction to normalized representations.

As FDFAs and FDWAs are syntactically the same, we say *family* $\mathcal{F} = (\mathcal{T}, \mathcal{D})$ to refer to either an FDFA or an FDWA. We call $\mathcal{D}_u$ a *progress automaton*, and write just $\mathcal{D}$ to denote $\mathcal{D}_\varepsilon$ in the case that $\mathcal{T}$ is trivial. The *size* of $\mathcal{F}$ is the pair (n, k) where n is the size of $\mathcal{T}$ and k is the maximum size of any $\mathcal{D}_u$. We call a family $\mathcal{F}$ *saturated* if for all normalized $(u, x), (v, y)$ with $ux^\omega = vy^\omega$ holds that (u, x) is accepted by $\mathcal{F}$ iff (v, y) is accepted by $\mathcal{F}$. We say that $\mathcal{F}$ is *fully saturated* if this equivalence holds for all pairs, not only normalized ones. For every saturated FDFA $\mathcal{F}$, there exists a unique regular ω-language L which contains precisely those ultimately periodic words that have a normalized representation which is accepted by $\mathcal{F}$ [2, Theorem 5.7]. In this case we say that L is the *language* accepted by $\mathcal{F}$ and denote it by $L_\omega(\mathcal{F})$.

The left quotient of the word u with regard to a regular language L of finite or infinite words, denoted $u^{-1}L$, is the set of all w such that $uw \in L$. Using it, we define the prefix congruence $\sim_L$ of L, as $u \sim_L v$ if $u^{-1}L = v^{-1}L$. We call L *prefix-independent* if $\sim_L$ has only one class. The *syntactic FDFA* $\mathcal{F}_L = (\mathcal{T}_{\sim_L}, \mathcal{D}^L)$ of a regular ω-language L is always saturated,

and it is the minimal one (with respect to the size of its progress DFAs) among all FDFAs that use $\mathcal{T}_{\sim_L}$ as leading TS. In general, one may obtain a saturated FDFA with strictly smaller progress DFAs by using a larger leading TS that refines $\sim_L$ [22, Proposition 2]. This does not hold in fully saturated FDFAs, because in those the minimal progress DFAs for all states reached on $\sim_L$-equivalent words coincide. Intuitively, saturation can be viewed as a form of semantic determinism, which guarantees the existence of unique, minimal progress DFAs (for a given leading TS). Due to their nondeterministic nature and the resulting non-uniqueness of their progress DFAs, the notion of an almost saturated syntactic FDFA does not exist.

3 Saturation Problems

In this section, we consider various saturation problems for FDFA and FDWA. We first establish PTIME-decidability of saturation for both types of families. Then, we illustrate that this can be applied for both active and passive learning of ω-regular languages. Finally, we consider a relaxed notion of saturation and we show that deciding it is PSPACE-complete.

3.1 Deciding saturation in PTIME

In the following, we state the main theorem regarding both saturation and full saturation. Subsequently, however, we opt to focus only on the former case of saturation, deferring a proof of the more general statement to the full version.

▶ **Theorem 1.** *For a given FDFA $\mathcal{F}$, one can decide in polynomial time whether $\mathcal{F}$ is (fully) saturated. If not, there are (not necessarily) normalized pairs $(u, x), (v, y)$ of polynomial length with $ux^\omega = vy^\omega$, such that $\mathcal{F}$ accepts (u, x) and rejects (v, y).*

We call $\mathcal{F}$ *loopshift-stable* if shifting the start of the looping part closer to the beginning or further from the beginning preserves acceptance. Formally, this is the case if, for all normalized (u, ax) it holds that $\mathcal{F}$ accepts (u, ax) if, and only if, $\mathcal{F}$ accepts (ua, xa). If the leading TS is trivial, then there is only one progress automaton. In this case we also say that the language of this progress automaton is loopshift-stable (it contains ax if, and only if, it contains xa for each word x and letter a). We say that $\mathcal{F}$ is *power-stable* when (de-)duplications of the loop preserves acceptance, so if for all normalized (u, x) holds that either $\mathcal{F}$ accepts all (u, x^i) or $\mathcal{F}$ rejects all (u, x^i), where $i > 0$. The following result has been established for full saturation in [15, 16] (see the subsection on related work in the introduction).

▶ **Lemma 2.** *An FDFA is saturated if, and only if, it is loopshift-stable and power-stable.*

Proof (sketch). Every normalized pair (u, x) has a canonical form that can be obtained by first shifting as much of u as possible into the looping part, and then deduplicating the obtained looping part. So we can go back and forth between any two representations of the same ω-word by a sequence of only loopshifts and (de)duplications of the looping part. Thus, $\mathcal{F}$ treats all representations of an ω-word the same if and only if acceptance is preserved under loopshifts and (de)duplications, and the claim is established. ◀

We now show that deciding the conjunction of loopshift-stable and power-stable is possible in polynomial time if done in the right order (from the proof of Theorem 8 one can deduce that deciding only power-stable alone is PSPACE-hard).

▶ **Lemma 3.** *There exists an algorithm which given an FDFA $\mathcal{F}$, decides in polynomial time whether $\mathcal{F}$ is loopshift-stable. Moreover, if $\mathcal{F}$ is not loopshift-stable, the algorithm returns a normalized representation (u, ax) of polynomial length such that $\mathcal{F}$ accepts (u, ax) if and only $\mathcal{F}$ it rejects (ua, xa).*

Proof (sketch). It suffices to check for every state u of $\mathcal{T}$ and symbol $a \in \Sigma$, whether there exists a word w such that $\big(aw$ is accepted by $\mathcal{D}_u$ and loops on u in $\mathcal{T}\big)$ if, and only if, $\big(wa$ is *not* accepted by $\mathcal{D}_{ua}$ and loops on ua in $\mathcal{T}\big)$. Finding such a word is equivalent to testing emptiness for a DFA that can be obtained by a product construction between transition systems that are of the same size, and can easily be obtained from $\mathcal{T}$ and $\mathcal{D}_u$. This guarantees polynomial runtime, and also means that a counterexample of polynomial length exists. ◀

▶ **Lemma 4.** *There exists an algorithm that given a loopshift-stable FDFA $\mathcal{F}$ in which each progress DFA is minimal, decides in polynomial time whether $\mathcal{F}$ is power-stable. Moreover, if $\mathcal{F}$ is not power-stable, the algorithm returns a counterexample (u, x) of polynomial length such that $\mathcal{F}$ accepts (u, x^i) and rejects (u, x^j) where the exponents i, j are bounded by the maximum size of a progress automaton of $\mathcal{F}$.*

Proof (sketch). Let u be a state in $\mathcal{T}$. This proof makes use of the assumption that x and y leading to the same state in $\mathcal{D}_u$ implies that ux and uy lead to the same state in $\mathcal{T}$. Specifically, this property ensures that if a state q of $\mathcal{D}_u$ is reached by a word that loops on u in $\mathcal{T}$, then all words reaching q loop on u.

The loopshift-stability of $\mathcal{F}$ guarantees that for all words $x_1, \cdots, x_n, x'_1, \cdots, x'_n$ that loop on state u in $\mathcal{T}$ and satisfy that $\mathcal{D}_u(x_i) = \mathcal{D}_u(x'_i)$ for $1 \leq i \leq n$, we have that $\mathcal{D}_u(x_1 \dots x_n) = \mathcal{D}_u(x'_1 \dots x'_n)$. This means that in an arbitrary word of the form $x_1 \dots x_n$, swapping out any x_i for some other x'_i that leads to the same state in $\mathcal{D}_u$, will not change the state reached by the resulting word. Thus, an algorithm which tests for power-stability, only has to consider at most one representative for each state q of a progress DFA $\mathcal{D}_u$.

For each progress DFA $\mathcal{D}_u$, the algorithm iterates though all states q of $\mathcal{D}_u$. If no words reaching q loop on u, it proceeds to the next state as such states are irrelevant when considering normalized acceptance. Otherwise, all words that reach q loop on u, and the algorithm picks a short representative r. The sequence of states reached by the words $r, r^2, \dots$ will repeat after at most $|\mathcal{D}_u|$ entries. This means that the algorithm only has to check if all of them are accepted, or all of them are rejected. If the algorithm finds a state in some progress DFA for which this is not the case, then $\mathcal{F}$ is not power-stable. As the length of u and r are at most $|\mathcal{T}|$ and $|\mathcal{D}_u|$, respectively, it is easy to build a counterexample with the desired properties. ◀

By first checking for loopshift-stability, and only then testing for power-stability means we can decide saturation in polynomial time. A similar result can be obtained for FDWA.

▶ **Theorem 5.** *For given FDWA $\mathcal{W}$, one can decide in polynomial time if $\mathcal{W}$ is saturated.*

Proof (sketch). As acceptance in an FDWA is naturally invariant under (de)duplication of the looping part, we only have to check whether $\mathcal{W}$ is invariant under loopshifts. This can be done through an approach similar to the one used in Lemma 3 adapted to the ω-semantics of the progress automata of an FDWA. ◀

3.2 Application to Learning

In this section we show that the polynomial-time algorithms for (full) saturation of FDFAs can be used to build polynomial time learning algorithms that can learn the regular ω-languages represented by FDFAs. More specifically, we provide a polynomial-time active

learner for fully saturated FDFAs, as well as a polynomial-time passive learner that can learn syntactic FDFAs from polynomial data. As mentioned in the introduction, these are the first polynomial algorithms for representations of the full class of regular ω-languages.

In general, in automata learning, there is a class $\mathcal{L}$ of potential target languages for which the algorithm should be able to infer a representation in a class $\mathcal{C}$ of automata that represent languages in $\mathcal{L}$. For active learning, we consider the standard setting of a minimally adequate teacher from [1], in which an *active learner* can ask membership and equivalence queries. Membership queries provide information on the membership of words in the target language. An equivalence query can be asked for automata in $\mathcal{C}$, and if the automaton does not accept the target language, then the answer is a counter-example in the symmetric difference of the target language and the language defined by the automaton. The running time of an active learner for the automaton class $\mathcal{C}$ with oracles for a language L as input is measured in the size of the minimal automaton for L (in the class $\mathcal{C}$ under consideration), and the length of the longest counter-example returned by the equivalence oracle during the execution of the algorithm. It is by now a well-known result that there are polynomial time active learners for the class of regular languages represented by DFAs [1, 32, 21]. Based on such a DFA learner and the full saturation check (Theorem 1) we can build an active learner for fully saturated FDFAs.

▶ **Theorem 6.** *There is a polynomial time active learner for the class of regular ω-languages represented by fully saturated FDFAs.*

Proof (sketch). The learning algorithm uses a polynomial time active learner for DFAs in order to learn a DFA for the language $L_\$:= \{u\$v \mid uv^\omega \in L\}$ over the alphabet $\Sigma \cup \{\$\}$. Membership queries can easily be answered using the membership oracle for ultimately periodic words. On equivalence queries of the DFA learner, our algorithm first translates the DFA over $\Sigma \cup \{\$\}$ into an FDFA $\mathcal{F}_\mathcal{A}$ with $\mathsf{L}_{\mathsf{rep}}(\mathcal{F}_\mathcal{A}) = \{(u, v) \in \Sigma^* \times \Sigma^+ \mid u\$v \in L_*(\mathcal{A})\}$, which is a straight-forward construction. Then it checks if the FDFA is fully saturated (Theorem 1), and if not returns a counter-example to the DFA learner derived from the example for failure of full saturation. If the FDFA is fully saturated but there is a counter-example (u, v) this is passed on as $u\$v$ to the DFA learner. ◀

A *passive learner* gets as input two sets S_+ of examples that are in the language and S_- of examples that are not. It outputs an automaton from $\mathcal{C}$ such that all examples from S_+ are accepted and all examples from S_- are rejected. The pair $S = (S_+, S_-)$ is referred to as sample. It is an L-sample if all examples in S_+ are in L, and all examples in S_- are outside L. For the class of regular ω-languages represented by their syntactic FDFAs, the sets S_+ and S_- contain representations of ultimately periodic words, and the passive learner outputs a syntactic FDFA $\mathcal{F}$ that is consistent with S, that is, $S_+ \subseteq \mathsf{L}_{\mathsf{rep}}(\mathcal{F})$ and $S_- \cap \mathsf{L}_{\mathsf{rep}}(\mathcal{F}) = \emptyset$. In the terminology of [20], we say that a passive learner f can learn automata from $\mathcal{C}$ for each language in $\mathcal{L}$ in the limit if for each $L \in \mathcal{L}$ there is an L-sample S_L, such that $f(S_L)$ returns an automaton $\mathcal{A}$ that accepts L, and for each L-sample S that extends S_L (contains all examples from S_L), $f(S)$ returns the same automaton $\mathcal{A}$. Such a sample S_L is called a characteristic sample for L and f. Further, f is said to learn automata from $\mathcal{C}$ for each language in $\mathcal{L}$ in the limit from polynomial data if for each L there is a characteristic sample for L and f of size polynomial in a minimal automaton for L (from the class $\mathcal{C}$).

▶ **Theorem 7.** *There is a polynomial time passive learner for regular ω-languages represented by syntactic FDFAs that can infer a syntactic FDFA for each regular ω-language in the limit from polynomial data.*

Proof (sketch). The passive learner constructs an FDFA $\mathcal{F}$ from the given sample S using techniques known from passive learning of DFAs. It then checks whether $\mathcal{F}$ is a syntactic FDFA that is consistent with S and returns $\mathcal{F}$ if yes. Otherwise, it returns a default FDFA that accepts precisely all representations of the ultimately periodic words in S_+. The test whether $\mathcal{F}$ is a syntactic FDFA can then be done by checking whether $\mathcal{F}$ is saturated (Theorem 1). We guarantee in step 1 that there is no smaller leading congruence for the given sample. ◀

3.3 Almost Saturation

Saturation is sufficient, but not necessary to guarantee ω-regular semantics. In [25], the weaker notion of *almost saturation* is shown to also be sufficient, and in this section we consider the complexity of deciding whether an FDFA has this property. We say that $\mathcal{F} = (\mathcal{T}, \mathcal{D})$ is *almost saturated* if $(u, x) \in \mathsf{L}_{\mathsf{norm}}(\mathcal{F})$ implies $(u, x^i) \in \mathsf{L}_{\mathsf{norm}}(\mathcal{F})$ for all $i > 1$. In other words, once such an FDFA accepts a loop, it must also accept all repetitions of it, which means that the languages accepted by the progress automata are closed under taking powers of the words in the language.

▶ **Theorem 8.** *It is PSPACE-complete to decide whether an FDFA is almost saturated.*

Proof (sketch). Membership in PSPACE can be established by an algorithm that guesses a word showing that the FDFA is not almost saturated. To show PSPACE-hardness, we reduce from the DFA intersection problem, which asks whether the intersection of an arbitrary sequence $\mathcal{D}_1, \ldots, \mathcal{D}_p$ of DFAs is non-empty and is known to be PSPACE-complete [23]. We assume that the number p of automata is prime, otherwise we pad the sequence with universal DFAs. We introduce a fresh symbol # and construct the FDFA $\mathcal{F} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is trivial and $\mathcal{A}$ is a DFA for the complement of $\# L_*(\mathcal{D}_1) \# L_*(\mathcal{D}_2) \# \cdots \# L_*(\mathcal{D}_p)$. Then a word x is accepted by $\mathcal{A}$ while x^i is rejected iff $x = \#u$ for u accepted by all the $\mathcal{D}_j$. ◀

4 Regularity

We only consider normalized representations in this section, following other works on FDFAs [2, 24, 25], and provide proof details in the full version. We believe that the methods that we present in this section also work for the non-normalized semantics, but this requires some extra details, so we prefer to stick to one setting.

An FDFA or FDWA $\mathcal{F}$ defines a *UP-language* (a set of ultimately periodic words) through the representations it accepts. If this UP-language L is UP-regular, that is, there is an ω-regular language L' such that $L = \mathsf{UP}(\Sigma) \cap L'$, then L' is unique and $\mathcal{F}$ defines this ω-regular language. For FDFAs it is known that they define ω-regular languages if they are saturated [2] or almost saturated [25]. But in general, the UP-language of an FDFA needs not to be UP-regular, which is witnessed e.g. by an FDFA accepting pairs of the form (u, ba^i) for which the UP-language is $\bigcup_{i \in \mathbb{N}} \left(\Sigma^*(ba^i)^\omega\right)$, which is not UP-regular [25, Example 2].

The main result in this section is that the *regularity problem for FDFAs* is decidable: given an FDFA, decide whether it defines a regular ω-language, that is, whether its UP-language is UP-regular. We show that the problem is PSPACE-complete. Note that almost saturation is *not* a necessary condition, as witnessed by a simple progress DFA accepting a^i for all odd integers i, which clearly defines a UP-regular language for the trivial leading TS but is not almost saturated.

But let us start with the observation that for FDWAs there is no decision problem because the UP-language is always UP-regular, which can be shown by the same construction to NBAs that is used for (almost) saturated FDFAs (which goes back to [13]).

▶ **Lemma 9.** *For every FDWA $\mathcal{W} = (\mathcal{T}, \mathcal{B})$, the UP-language $\{ux^\omega \mid x^\omega$ is accepted by $\mathcal{B}_u\}$ is UP-regular*

We now turn to the regularity problem for FDFAs. We first show that it is sufficient to analyze loopshift-stable FDFAs with trivial leading TS (Lemma 10, Theorem 11). However, this requires to work with *nondeterministic* progress automata because making an FDFA loopshift-stable can cause an exponential blow-up.

We call $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ a *family of NFAs* (*FNFA*), where $\mathcal{T}$ is the leading *deterministic* TS and, for each $q \in Q(\mathcal{T})$, $\mathcal{N}_q$ is an NFA. Recall, that we consider the normalized setting, so (u, v) is accepted by $\mathcal{F}$ if $\mathcal{T}(u) = \mathcal{T}(uv)$ and v is accepted by $\mathcal{N}_{\mathcal{T}(u)}$. FDFAs are simply special cases of FNFAs, where all progress automata are deterministic. The notion of loopshift-stability and other notions defined for FDFA naturally carry over to FNFAs as well.

▶ **Lemma 10.** *Given an FDFA or FNFA $\mathcal{F} = (\mathcal{T}, \mathcal{N})$, we can build a loopshift-stable FNFA $\mathcal{F}' = (\mathcal{T}, \mathcal{N}')$ in polynomial time such that $\mathcal{F}$ and $\mathcal{F}'$ define the same UP-language.*

Proof (sketch). $\mathcal{F}'$ can nondeterministically guess, for (u, v), the shift (uv_1, v_2v_1); $\mathcal{T}(uv_1)$; and the state p the accepting run of $N_{\mathcal{T}(uv_1)}$ on v_2v_1 is in after reading v_2, and use its nondeterministic power to validate the guess. ◀

We now state the main theorems of this section.

▶ **Theorem 11.** *The regularity problem for FDFAs is PSPACE-complete. More precisely, the following problems are PSPACE-complete:*

1. *Is the UP-language of a given FNFA $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ UP-regular?*
2. *Is the UP-language of a given FDFA $\mathcal{F} = (\mathcal{T}, \mathcal{D})$ with a trivial TS $\mathcal{T}$ UP-regular?*
3. *Is the UP-language of a given FNFA $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ with a trivial TS $\mathcal{T}$ and a loopshift-stable set $P = L_*(\mathcal{N})$ UP-regular?*
4. *Is the UP-language of a given FNFA $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ with a trivial TS $\mathcal{T}$ UP-regular?*

Proof outline: To show hardness, it is enough to show hardness for (2), as hardness for (3) then follows from Lemma 10, and both (2) and (3) are special cases of (1) and (4). We show hardness of (2) in Theorem 16.

Most of this section is dedicated to establish a decision procedure (which is in PSPACE) for (3). This is done in Lemmas 12 through 15. Using Lemma 10 reduces (4) to (3), and (2) is a special case of (4). Case (1) can be reduced to case (3) by considering a labeling of the words with the states of the leading TS. ◀

Prefix-independent languages. For the PSPACE upper bound, as justified by the reduction to (3) of Theorem 11, we focus on a loopshift-stable FNFA $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ with trivial $\mathcal{T}$. So there is only one progress NFA $\mathcal{N} = (\Sigma, Q, \delta, \iota, F)$, which accepts some language $P = L_*(\mathcal{N})$ satisfying $uv \in P \Leftrightarrow vu \in P$ as $\mathcal{F}$ is loopshift-stable. Since $\mathcal{T}$ is trivial, the UP-language L of $\mathcal{F}$ is prefix-independent and is of the form $L = \{u \cdot v^\omega \mid (u, v) \in \mathsf{L}_{\mathsf{norm}}(\mathcal{F})\} = \Sigma^* \cdot \omega\text{-}\mathsf{Pow}(P)$ (recall that $\omega\text{-}\mathsf{Pow}(P) = \{v^\omega : v \in P\}$). Our goal is to decide whether L is UP-regular.

As the semantics is existential, it can happen that some representations of a word $v^\omega \in \omega\text{-}\mathsf{Pow}(P)$ are not in P. This makes the problem non-trivial even in the prefix-independent case as witnessed by the language $P = \{a^iba^j \mid i + j \geq 1\}$, which consists of all words with exactly one b and at least one a over the alphabet $\Sigma = \{a, b\}$. Clearly, P is regular, but $L = \Sigma^* \cdot \{\left(a^iba^j\right)^\omega \mid i + j \geq 1\}$ is not UP-regular.

We now define a congruence relation $\approx_P$: for $x, y \in \Sigma^*$,

$$x \approx_P y \text{ iff for all } v \in \Sigma^* \text{ holds } (xv)^\omega \in \omega\text{-}\mathsf{Pow}(P) \iff (yv)^\omega \in \omega\text{-}\mathsf{Pow}(P).$$

As $L = \Sigma^* \cdot \omega\text{-}\mathsf{Pow}(P)$ is prefix-independent, $\approx_P$ is the syntactic congruence of L defined in [5], and also corresponds to the definition of the progress congruence of the syntactic FORC [29] and to the periodic progress congruence from [3] (where the latter two are only defined for regular ω-languages). It follows from results in [13, 2] that L is UP-regular if, and only if, $\approx_P$ has finite index.

▶ **Lemma 12.** *Let $\mathcal{F} = (\mathcal{T}, \mathcal{N})$ be a loopshift-stable FNFA with trivial TS $\mathcal{T}$; let $P = L_*(\mathcal{N})$ and let $L = \Sigma^* \cdot \omega\text{-}\mathsf{Pow}(P)$. Then the following claims are equivalent: (1) $\approx_P$ has finite index; and (2) the UP-language L of $\mathcal{F}$ is UP-regular.*

Transition profiles. By Lemma 12, to determine whether or not the UP-language of $\mathcal{F}$ is UP-regular, we only need to check whether $\approx_P$ has finite index. To this end, we introduce the *transition profile* of a finite word $x \in \Sigma^+$, which captures the behavior of x in a TS or automaton, in this case the progress NFA $\mathcal{N}$. Formally, τ is given as a mapping $\tau : Q \to Q$ if $\mathcal{N}$ is deterministic and $\tau : Q \to 2^Q$ if it is nondeterministic, with $q \mapsto \delta^*(q, x)$, and we write $\tau_\mathcal{N}$ for the function that assigns to a finite word its transition profile in $\mathcal{N}$. There are at most $|Q|^{|Q|}$ and $2^{|Q|^2}$ different mappings for deterministic and nondeterministic automata, respectively. We now refine $\approx_P$ using the transition profile by introducing

$$x \approx_\mathcal{N} y \text{ if, and only if, } x \approx_P y \text{ and } \tau_\mathcal{N}(x) = \tau_\mathcal{N}(y) \ .$$

Clearly, $\approx_P$ has finite index if, and only if, $\approx_\mathcal{N}$ has finite index.

Transition profiles are useful, because they offer sufficient criteria for a word x to be in the power language: we can directly check from $\tau_\mathcal{N}(x)$ whether or not there is some power $j > 0$ such that $x^j \in P$. This, of course, entails $x^\omega \in \omega\text{-}\mathsf{Pow}(P)$. We call such transition profiles *accepting* and transition profiles, where no power of x is accepted, *rejecting*. Note that $\tau_\mathcal{N}(x)$ is called accepting if *some power* of x is accepted, not necessarily x itself. Correspondingly, $\tau_\mathcal{N}(x)$ is rejecting if *all powers* of x are rejected.

Note that there can be words x such that $x^\omega \in \omega\text{-}\mathsf{Pow}(P)$ but no power of x is in P. For instance, let $P = \{b^i a b^j : i + j \geq 0\}$ be a loopshift-stable regular language: clearly $(ab \cdot ab)^\omega \in \omega\text{-}\mathsf{Pow}(P)$ but $(ab \cdot ab)^i \notin P$ for all $i > 0$.

Recall that the root $\sqrt{x}$ is the shortest word u such that $u^\omega = x^\omega$. To get a feeling for the properties of transition profiles, for a word x with $x = \sqrt{x}$, $x^\omega \in \omega\text{-}\mathsf{Pow}(P)$ can only hold if some power of x is in P, which we can read from $\tau_\mathcal{N}(x)$.

We call a transition profile τ *terminal* if it is accepting, but τ^i is rejecting for some power i. (Note that $\tau_\mathcal{N}(x^i)$ is the function obtained from iterating $\tau_\mathcal{N}(x)$ i times. However, we can take the power of a transition profile directly without knowing x, and even for transition profiles that do not correspond to a word.)

We call a word x such that $\tau_\mathcal{N}(x)$ is terminal, a *terminal word* and write $\mathsf{Ter}(P)$ for the set of terminal words in P. So x is a terminal word if some power x^j is in P, and there is $i > 1$ such that $x^{ik} \notin P$ for all $k \geq 1$. Note that if x is a terminal word, then so is $\sqrt{x}$. The following lemma expresses finite index of $\approx_P$ in terms of a property of $\mathsf{Ter}(P)$ that we use in our decision procedure.

▶ **Lemma 13.** *If the NFA $\mathcal{N}$ accepts a loopshift-stable language P, then $\approx_\mathcal{N}$ (and thus $\approx_P$) have finite index if, and only if, $\mathsf{Ter}(P)$ has finitely many roots.*

Proof (sketch). We show that, if there are infinitely many roots in $\mathsf{Ter}(P)$, then we can construct from them an arbitrary number of words that are distinguished by $\approx_P$; $\approx_P$ therefore has infinite index.

Conversely, if there are finitely many roots $r_1, \ldots, r_n$ in $\mathsf{Ter}(P)$, then two words x, y are identified equivalent by $\approx_P$ if they define the same transition profile $\tau_{\mathcal{N}}(x) = \tau_{\mathcal{N}}(y)$ and for all $z \in \Sigma^*$ $\tau_{\mathcal{N}}(xz)$ is accepting or there is a root $r_i \in \mathsf{Ter}(P)$ such that $(xz)^\omega = r_i^\omega$ iff there is a root $r_j \in \mathsf{Ter}(P)$ such that $(yz)^\omega = r_j^\omega$ where $1 \leq i, j \leq n$.

These are all regular properties, and $\approx_P$ is therefore finite. ◀

We now show that infinitely many roots of $\mathsf{Ter}(P)$ are witnessed by transition profiles with specific properties, called good witnesses, as defined below.

We say that a word x visits τ_g first if (a) $\tau_{\mathcal{N}}(x) = \tau_g$ holds and (b) no true prefix y of x satisfies $\tau_{\mathcal{N}}(y) = \tau_g$. If there is a word x that visits τ_g first, we say that u recurs to τ_g if $u \neq \varepsilon$ and (a) $\tau_{\mathcal{N}}(xu) = \tau_g$ holds and (b) no true non-empty prefix v of u satisfies $\tau_{\mathcal{N}}(xv) = \tau_g$.

We say that τ_g is a *good witness* if it is terminal and one of the following holds:

1. there are infinitely many words x that visit τ_g first, *or*
2. there is a word x that visits τ_g first and a word u that recurs on τ_g that have different roots.

▶ **Lemma 14.** *Let $P = L_*(\mathcal{N})$. There is a good witness τ_g if, and only if, $\mathsf{Ter}(P)$ has infinitely many roots.*

Proof (sketch). If $\mathsf{Ter}(P)$ has infinitely many roots, then infinitely many of them refer to one witness. We show that they cannot all be constructed from one word x that visits τ_g first and one word u that recurs on τ_g *and* has the same root as τ_g, so that one of the two cases must hold.

Reversely, if τ_g is a good witness we distinguish case (1), where we argue that the infinitely many words that visit τ_g first must have different roots (and roots of terminal words are terminal). For the other case, we build infinitely many roots with the terminal transition profile τ_g from a word x that visits τ_g first and a word u that recurs on τ_g and has a different root than x. ◀

Checking if there is a good witness can be done on the succinctly defined TS whose state space are the transition profiles that tracks the transition profiles of finite words.

▶ **Lemma 15.** *For a given NFA $\mathcal{N}$, we can check if there is a good witness in PSPACE.*

Proof (sketch). We observe that tracing the transition profiles is done by a TS $\mathcal{T}'$ of size exponential in $\mathcal{N}$, but succinctly represented by $\mathcal{N}$ (and in τ_g for checking that τ_g is terminal). Checking the conditions of a good transition profile are a number of reachability problems in this transition profile, and all of them are in NL the size of $\mathcal{T}'$. ◀

This finishes the proof for the PSPACE upper bound. We now show the lower bound for case (2) of Theorem 11.

▶ **Theorem 16.** *Deciding for a DFA $\mathcal{D}$ whether $\mathsf{Ter}(P)$ is finite for $P = L_*(\mathcal{D})$ and deciding whether $\mathsf{Ter}(P)$ has finitely many roots are PSPACE-hard. Furthermore, deciding if the UP-language of an FDFA with trivial leading TS is UP-regular, is PSPACE-hard.*

Proof. Assume that we are given p DFAs $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_p$ that do not accept the empty word; we assume that p is prime (and otherwise pad) and that $\# \notin \Sigma$. We define $L_1 = \#^+\Sigma^+\#^*$, $L_p = {L_1}^p$, and $L_\cap = \bigcap_{i=1}^p L_*(\mathcal{D}_i)$. Deciding if $L_\cap = \emptyset$ is the PSPACE-hard problem we reduce from.

We build two DFAs $\mathcal{D}$ and $\mathcal{D}'$ using the states of $\mathcal{D}_1, \ldots, \mathcal{D}_p$ plus $p+1$ extra states. $\mathcal{D}'$ accepts a word w if it is in L_p and of the form $w = \#^+ w_1 \#^+ w_2 \ldots \#^+ w_p \#^*$ s.t. $w_i \in \Sigma^+$ holds for all $i \leq p$ *and* $w_i \notin L_*(\mathcal{D}_i)$ holds for some $i \leq p$. We call its language L'.

$\mathcal{D}$ accepts a word w if $w \in {L_p}^* \cdot L' \cdot {L_p}^*$ *or* $w \in {L_1}^i$ and p does not divide i.

If $L_\cap \neq \emptyset$, then every word $v \in \#^+ L_\cap$ is terminal for $P = L_*(\mathcal{D})$, because v^p defines a rejecting transition profile. These words are all roots, and the $\#^+$ part ensures that there are infinitely many of them.

For the case $L_\cap = \emptyset$ we assume towards a contradiction that $\mathsf{Ter}(P)$ contains a root u. Then $u \in {L_1}^i \#^*$ for some i. If p divides i, then either all powers of u are accepted or no power of u is accepted (contradiction). If p does not divide i, then u^j is accepted if p divides j (because all i words in Σ^+ are rejected by some $\mathcal{D}_k$ due to $L_\cap = \emptyset$), and u^j is accepted if p does not divide j, because then p does not divide ij and $u^j \in {L_1}^{ij}$ (contradiction).

For showing the second part of the theorem, we use the DFA $\mathcal{D}$ as progress automaton for an FDFA with trivial leading TS, and then use Lemma 12 and Lemma 13. These lemmas require the language that is accepted to be loopshift-stable. While $\mathcal{D}$ does not recognize a loopshift-stable regular language P, it is not hard to see that the loopshift-stable language $P' = \{xy \mid x, y \in (\Sigma \cup \{\#\})^*, yx \in P\}$ we obtain by the construction from Lemma 10 also satisfies that $\mathsf{Ter}(P') = \emptyset$ if $L_\cap = \emptyset$. Since $\mathsf{Ter}(P) \subseteq \mathsf{Ter}(P')$, we also get that $\mathsf{Ter}(P')$ contains infinitely many roots if $L_\cap \neq \emptyset$. Furthermore P' and P define the same UP-language, and thus deciding if this UP-language is UP-regular is the same as deciding if $\mathsf{Ter}(P)$ has finitely many roots. ◀

5 Comparison

In this section, we analyze the influence that the properties introduced in Section 3 can have on the succinctness of the various models. Moreover, we also consider the transformation between models with certain properties. The results of this section are presented visually in the overview in Figure 1. We sketch the families used to obtain lower bounds only roughly, deferring formal definitions and proofs the full version for the sake of readability.

Saturated FDWAs may be exponentially more succinct than saturated FDFAs. Intuitively, this comes from the fact that by Lemma 2 FDFAs have to be loopshift-stable, which implies that their progress DFAs exhibit a monoidal structure. We have described this structure in more detail, and shown how it makes deciding power-stability tractable in Lemma 4. However it also means that if the leading TS is trivial, then in the progress automaton two words u, v may only lead to the same state if they cannot be distinguished in arbitrary products, that is, for all words x, w, either xuw and xvw are both accepted by the progress DFA, or both are rejected. This insight allows us to obtain the following lower bound with standard techniques.

▶ **Lemma 17.** *For each $n > 0$, there is a language L_n that can be recognized by a saturated FDWA of size $(1, n+2)$, and by an almost saturated FDFA of size $(1, n+2)$, but for every saturated FDFA of size $(1, m)$ that recognizes L_n, we have that $m \geq n^n$.*

Proof (sketch). We represent mappings $\{1, \ldots, n\} \to \{1, \ldots, n\}$ as words over an alphabet of constant size. The language L_n contains a word w if it has infinitely many infixes $\#u\#$ such that 1 is a fixpoint of the mapping represented by u. A saturated FDWA can wait for an occurrence of $\#$ and then keep track of the number that 1 is mapped to, which is possible with $n+2$ states. As the progress automaton of this FDWA has only a single accepting state which is a sink, it is almost saturated when viewed as FDFA. By Lemma 2, a saturated

FDFA must be loopshift-stable, and therefore cannot wait for an occurrence of # before following a single number. Thus, it has to memorize the full mapping encoded by the word read so far, for which it needs at least n^n states. ◀

The size of an almost saturated FDFA is, in general, incomparable to the size of a saturated FDWA, and there may be an exponential blowup in either direction, which the following lemma formalizes.

▶ **Lemma 18.** *For all $n > 0$, there are languages L_n and L'_n such that*

1. *L_n is accepted by a saturated FDWA of size $(1, 2n+1)$, while for every almost saturated FDFA for L_n with trivial leading TS has a progress automaton of size $2^{\frac{n}{2}}$*
2. *L'_n can be recognized by an almost saturated FDFA of size $(1, n+3)$, and by an FDWA of size $(1, n+3)$, but every saturated FDWA with trivial leading TS has at least 2^n states*
3. *in every almost saturated FDFA with trivial leading TS that accepts the complement of L'_n, the progress automaton has at least $\frac{2^n}{2n}$ states.*

Proof (sketch). The alphabet of L_n consists of non-empty subsets of $\{1, \dots, 2n\}$. A word is in L_n if some number i is member of only finitely many symbols, meaning L_n is prefix-independent. Due to its limit behavior, a saturated FDWA can test on the loop for an occurrence of each number in sequence, leading to a progress automaton with $2n+1$ states. But one can show that for all $X, Y \subseteq \{1, \dots, 2n\}$ with $X \neq Y$ and $|X| = |Y| = n$, either X and Y, or their complements must lead to different states from the initial state of the progress automaton of any almost saturated FDFA for L_n.

The language L'_n is defined over the alphabet $\{0, 1\}$, and contains all words with infinitely many infixes $0u0$ such that $|u| = n-1$, which is clearly prefix-independent. When using a trivial leading TS, an FDWA or almost saturated FDFA can simply guess the start of an infix $0u0$ and verify that $|u| = n-1$, which can be done with $n+3$ states. This is not possible in a saturated FDWA since it has to be invariant under loopshifts. Similarly, in the progress automaton of any almost saturated FDFA for the complement of L'_n, distinct words of length n must lead to different states. ◀

The following bounds were already known in the literature (c.f. [22, Section 5] and [3, Theorem 2]). We include them for the purpose of completeness, and provide families of languages achieving them in a systematic way.

▶ **Lemma 19.** *For all $n > 0$ there are languages L_n, L'_n such that*

1. *L_n is recognized by a saturated FDFA of size $(n, 2n)$, but the syntactic FDFA for L_n is of size $(1, n^n)$, and*
2. *the syntactic FDFA for L'_n is of size $(n+1, 2n+1)$ and every fully saturated FDFA for L'_n has a progress DFA of size at least n^n.*

Proof (sketch). We consider words of the form $u_1\#u_2\#\dots$ where each u_i encodes a mapping m_{u_i} from $\{1, \dots, n\}$ to itself, akin to Lemma 17. Such a word is in L_n if infinitely many u_i represent a mapping such that 1 is a fixpoint. A small saturated FDFA $(\mathcal{T}, \mathcal{D})$ tracks in its leading TS $\mathcal{T}$ the number that 1 is sent to under the mapping encoded by the word read since the last #. This leads to n progress DFAs of size $2n$ each, and $\mathcal{D}_i$ accepts words of the form $x\#y$ where m_x maps i to 1 and m_y maps 1 to i. The syntactic FDFA on the other hand has a trivial leading TS and in its progress DFA, any words encoding distinct mappings have to lead to different states. For the second claim, we slightly modify L_n to force the leading TS of the syntactic FDFA to be roughly of the shape of $\mathcal{T}$, which means the each progress DFA is of linear size. A fully saturated FDFA for L', however, cannot simply ignore words if they do not loop on a state of the leading TS, which forces it to track all n^n mappings. ◀

6 Conclusion

We have determined the complexity of several decision problems related to the saturation of FDFAs and FDWAs. We showed that this has practical implications using polynomial-time (full) saturation check to provide a polynomial-time active learner for fully saturated FDFAs, as well as a polynomial-time passive learner capable of learning in the limit the syntactic FDFA for each regular ω-language from polynomial data. These are the first algorithms of their kind for representations of the entire class of ω-regular languages. Beyond these direct applications, we believe that our proofs offer deeper insights into the structure of FDFAs, particularly regarding the reasons for non-saturation and non-regularity, which may prove useful for further results and algorithms for ω-languages represented by families of automata.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.
2. Dana Angluin, Udi Boker, and Dana Fisman. Families of dfas as acceptors of ω-regular languages. *Log. Methods Comput. Sci.*, 14(1), 2018. doi:10.23638/LMCS-14(1:15)2018.
3. Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/J.TCS.2016.07.031.
4. Dana Angluin, Dana Fisman, and Yaara Shoval. Polynomial identification of ω-automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020*, volume 12079 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2020. doi:10.1007/978-3-030-45237-7_20.
5. André Arnold. A syntactic congruence for rational omega-language. *Theor. Comput. Sci.*, 39:333–335, 1985. doi:10.1016/0304-3975(85)90148-3.
6. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
7. Marie-Pierre Béal, Olivier Carton, and Christophe Reutenauer. Cyclic languages and strongly cyclic languages. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 49–59. Springer, 1996. doi:10.1007/3-540-60922-9.
8. León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.MFCS.2021.20.
9. León Bohn and Christof Löding. Passive learning of deterministic Büchi automata by combinations of DFAs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 114:1–114:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.114.
10. León Bohn and Christof Löding. Constructing deterministic parity automata from positive and negative examples. *TheoretiCS*, 3, 2024. doi:10.46298/THEORETICS.24.17.
11. J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
12. Hugues Calbrix and Maurice Nivat. Prefix and period languages of rational *omega*-languages. In *Developments in Language Theory II, At the Crossroads of Mathematics, Computer Science and Biology, Magdeburg, Germany, 17-21 July 1995*, pages 341–349. World Scientific, Singapore, 1996.

13 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational w-languages. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993. doi:10.1007/3-540-58027-1_27.

14 Anton Chernev, Helle Hvid Hansen, and Clemens Kupke. Dual adjunction between Ω-automata and wilke algebra quotients. *CoRR*, abs/2407.14115, 2024. doi:10.48550/arXiv.2407.14115.

15 Vincenzo Ciancia and Yde Venema. Stream automata are coalgebras. In *Coalgebraic Methods in Computer Science - 11th International Workshop, CMCS 2012, Colocated with ETAPS 2012, Tallinn, Estonia, March 31 - April 1, 2012, Revised Selected Papers*, volume 7399 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2012. doi:10.1007/978-3-642-32784-1.

16 Vincenzo Ciancia and Yde Venema. Omega-automata: A coalgebraic perspective on regular omega-languages. In *8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom*, volume 139 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CALCO.2019.5.

17 Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw 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, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008. doi:10.1007/978-3-540-78800-3_2.

18 Szilárd Zsolt Fazekas. Powers of regular languages. *Int. J. Found. Comput. Sci.*, 22(2):323–330, 2011. doi:10.1142/S0129054111008064.

19 Dana Fisman, Emmanuel Goldberg, and Oded Zimerman. A robust measure on fdfas following duo-normalized acceptance. In *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024*, volume 306 of *LIPIcs*, pages 53:1–53:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.53.

20 E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.

21 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

22 Nils Klarlund. A homomorphism concepts for omega-regularity. In *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.

23 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

24 Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for büchi automata based on family of dfas and classification trees. *Inf. Comput.*, 281:104678, 2021. doi:10.1016/J.IC.2020.104678.

25 Yong Li, Sven Schewe, and Qiyi Tang. A novel family of finite automata for recognizing and learning ømega-regular languages. In *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I*, volume 14215 of *Lecture Notes in Computer Science*, pages 53–73. Springer, 2023. doi:10.1007/978-3-031-45329-8_3.

26 Yong Li, Sven Schewe, and Qiyi Tang. Angluin-style learning of deterministic büchi and co-büchi automata. In *Proceedings of the Thirty-Third International Joint Conference on*

Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024, pages 4506–4514. ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/498.

27 Christof Löding. Efficient minimization of deterministic weak omega-automata. *Inf. Process. Lett.*, 79(3):105–109, 2001. doi:10.1016/S0020-0190(00)00183-6.

28 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi:10.1006/inco.1995.1070.

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

30 Jakub Michaliszyn and Jan Otop. Learning infinite-word automata with loop-index queries. *Artif. Intell.*, 307:103710, 2022. doi:10.1016/J.ARTINT.2022.103710.

31 Dominique Perrin and Jean-Eric Pin. *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*. Elsevier Morgan Kaufmann, 2004.

32 Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In Stephen Jose Hanson, Werner Remmele, and Ronald L. Rivest, editors, *Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT*, volume 661 of *Lecture Notes in Computer Science*, pages 51–73. Springer, 1993. doi:10.1007/3-540-56483-7_22.

33 Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–191. Elsevier and MIT Press, 1990. doi:10.1016/B978-0-444-88074-1.50009-3.

34 Wolfgang Thomas. Facets of synthesis: Revisiting church's problem. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009. doi:10.1007/978-3-642-00596-1_1.