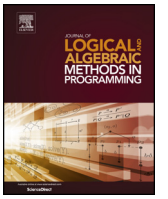
Journal of Logical and Algebraic Methods in Programming 98 (2018) 27–49

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Inferring regular languages and ω-languages

Dana Fisman

Ben-Gurion University, Israel

ARTICLE INFO

Article history:
Received 6 April 2017
Received in revised form 19 December 2017
Accepted 19 March 2018

Keywords:
Grammatical inference
Model learning
Regular languages
ω-regular languages
Right congruence
Myhill–Nerode theorem

ABSTRACT

In 1987 Angluin proposed an algorithm, termed $\mathbf{L}^*$ for inferring an unknown regular language using membership and equivalence queries. This algorithm has found many applications, amongst which in the area of system design and verification. These applications challenge the state-of-the art solutions in various directions, in particular, scaling or working with more succinct representations, and dealing with ω-languages, the main model for reasoning about reactive systems.

Both extensions confront a similar difficulty. Inference algorithms typically rely on the correspondence between the automata states and the right congruence, henceforth, the *residuality property*. DFAs enjoy the residuality property (as stated by the Myhill–Nerode Theorem) but more succinct representations such as non-deterministic and alternating finite automata (NFAs and AFAs) in general do not. The situation in the ω-languages realm is even worse, since none of the traditional automata that can express all regular ω-languages enjoys the residuality property.

This paper surveys residual models for regular languages and ω-languages and the learning algorithms that can infer these models.

1. Introduction

Grammatical inference or *automata learning* is the problem of inferring a finite concise representation (usually an automaton) of a fixed yet unknown language L. A prominent approach is that of *query learning* proposed by Angluin [3] in which the learning algorithm can pose queries to an *oracle* or *teacher* that is familiar with the language. The learning framework defines the different types of queries at the learner's disposal. For instance, a membership query (MQ, henceforth) asks whether a word w is a member of L, and is answered either by "yes" or by "no". An equivalence query asks whether a finite concise representation of an hypothesis language L' is equal to L. In the case of an equivalence query (EQ, henceforth), the answer may be "yes", in which case the learning algorithm has succeeded in exact identification of the target concept, or it may be "no", accompanied by a counterexample, that is, a word w that is in L but not in L' or vice versa. Angluin has shown that there is neither a polynomial-time learning algorithm to learn a regular language from only membership queries [3] nor there is a polynomial-time learning algorithm to learn a regular language from only equivalence queries [5]. On the positive side, she proposed an algorithm termed $\mathbf{L}^*$ that learns regular languages using membership and equivalence queries in polynomial time [3].

While the existence of an oracle answering equivalence queries, may seem unrealistic, the $\mathbf{L}^*$ algorithm has found applications in many areas. In some of them, the equivalence oracle is replaced by a large number of membership queries, in which case the resulting algorithm would be *probably approximately correct* (PAC). In others, equivalence queries are answered

E-mail addresses: dana@cs.bgu.ac.il, fisman@seas.upenn.edu.

https://doi.org/10.1016/j.jlamp.2018.03.002
2352-2208/

by reductions to questions posed to a model checker (this is the case for instance, in assume-guarantee reasoning [2,64,25, 32]). The idea of using $\mathbf{L}^*$ for *black-box* checking was postulated in [65]. In a recent CACM review paper, Vaandrager [71] explains *Model Learning*, which takes a black box approach to learning a finite state model of a given hardware or software system using membership queries (implemented by giving the system a sequence of inputs and observing the sequence of outputs) and equivalence queries (implemented using a set of test sequences in which the outputs of the hypothesis are compared with the outputs of the given system). The learned models may then be analyzed to find discrepancies between a specification and its implementation, or between different implementations. He cites applications in telecommunications, the automotive industry, online conference systems, as well as analyzing botnet protocols, smart card readers, bank card protocols, network protocols and legacy software. Other applications of $\mathbf{L}^*$ in model design and verification include regular model-checking [73,59], learning verification fixpoints [74] and error localization [24].

Coping with real life applications calls for algorithms that scale better. One direction to explore for achieving scaling is considering more succinct representations. The use of model learning of reactive (non-terminating) systems calls for obtaining learning algorithms that deal with ω-languages.

Learning regular languages. The $\mathbf{L}^*$ algorithm, as well other learning algorithms, are founded on the *residuality property*. This property relates the state of an automaton model for a language L with the so called *residual languages* of L. Given a language L and a word u, the *residual of u with respect to L*, denoted $u^{-1}L$ is the set of words $\{v \mid uv \in L\}$, i.e. the words v that if read after u take the automaton to an accepting state. The Myhill–Nerode theorem asserts that the number of states in a minimal DFA for a language L matches exactly the number of different residual languages for L, and that each state corresponds to exactly one of the residual languages. Learning algorithms usually identify state candidates with prefixes u of words in the unknown language, and infer that two candidate prefixes u_1 and u_2 correspond to two different states if they found a word v such that $u_1 v \in L$ and $u_2 v \notin L$ or vice versa. In other words, if they established that the residual language of u_1 is different than that of u_2.

While DFAs enjoy the residuality property, NFAs (non-deterministic automata) in general do not. For this reason learning algorithms, including $\mathbf{L}^*$, output DFAs rather than NFAs although the latter is a more succinct representation. This problem motivated Dennis, Lemay and Terlutte [28] to define *non-deterministic residual finite automata* (NRFA), a certain type of NFAs that does enjoy the residuality property. This idea can also be applied to alternating automata, inducing ARFA, *alternating residual finite automata*. Thanks to the residuality property of NRFAs and ARFAs polynomial learning algorithms $\mathbf{NL}^*$ [18] and $\mathbf{AL}^*$ [9] outputting NFAs and AFAs, resp., can be devised.

Learning regular ω-languages. *Reactive systems* are systems (or programs) that interact with their environment via inputs and outputs in an ongoing manner. Computations of such systems are naturally viewed as an infinite sequence of states. Such systems are thus modeled by ω-automata, automata processing infinite words (ω-words). There are several types of ω-automata (Büchi, Muller, Parity, Rabin, and Streett), differing in the way acceptance is defined. None of the models enjoy a residuality property. For this reason the problem of learning regular ω-languages (the class of languages recognized by ω-automata) was considered open for many years [47].

The *weak* regular ω-languages, is the sub-class of the regular ω-languages that can be recognized by *weak* parity automata. Weak parity automata do possess a residuality property [68]. This non-trivial sub-class can be polynomially learned using an algorithm termed $\mathbf{L}^{\mathrm{W}\omega}$ [54] that builds on this property, and overcomes additional challenges encountered when trying to extend $\mathbf{L}^*$ to learn ω-languages.

It turns out that the full class of regular ω-languages can actually be learned [31], using a reduction to languages of finite words, termed $L_\$$ [21]. Albeit, the efficiency of the resulting algorithm is hindered by the non-polynomiality of this reduction.

A series of long studies pursuing a Myhill–Nerode type characterization for the full class of regular ω-languages culminated in a representation of ω-languages using *families of right congruences* [55]. Based on these ideas a learning algorithm, termed $\mathbf{L}^\omega$, that yields a close representation termed *families of DFAs* (FDFAs) was devised [10]. While the FDFA model was shown to be exponentially more succinct than the $L_\$$ model, the complexity of $\mathbf{L}^\omega$ is not polynomial in the size of the FDFA (and can be bounded by the size of the $L_\$$ representation) [10]. Empirically though, the learning algorithm $\mathbf{L}^\omega$ behaves well and significantly better than the $L_\$$-based algorithm, on randomly generated targets [11].

Overview. The paper is organized as follows. In Section 2 we provide preliminaries regarding regular languages. In Section 3 we briefly summarize early results on learning regular languages. Section 4 explains the $\mathbf{L}^*$ algorithm. Section 5 describes the $\mathbf{NL}^*$ and $\mathbf{AL}^*$ algorithms following the introduction of the notion of residual non-deterministic and alternating automata. Section 6 briefly surveys extensions of $\mathbf{L}^*$ to richer formalisms.

We then turn to discuss inference of regular ω-languages. In Section 7 we provide preliminaries regarding regular ω-languages. Section 8 briefly summarizes early results on learning regular ω-languages. In Section 9 we describe the learning algorithm $\mathbf{L}^{\mathrm{W}\omega}$ for learning weak regular ω-languages. Section 10 describes the learning algorithm $\mathbf{L}^\omega$ for learning the full-class of regular ω-languages following the introduction of the model of families of DFAs.

2. Regular languages

Words and languages. As usual, an *alphabet* is a finite non-empty set of symbols, referred to as *letters*. A *word* is a finite sequence of letters, and an *ω-word* is an infinite sequence of letters. A *language* is a set of words, and an *ω-language* is a set of ω-words. The empty word is denoted by λ. For a word $w = a_1 a_2 \dots a_n$ where $a_i \in \Sigma$ for $i \in [1..n]$ we use $w[i..j]$ for the infix $a_i a_{i+1} \dots a_j$ for $1 \leq i \leq j \leq n$.

Automata and DFAs. A **deterministic automaton** is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ consisting of an alphabet Σ, a finite set Q of states, an initial state $q_0 \in Q$, and a transition function $\delta : Q \times \Sigma \to Q$. A run of an automaton on a finite word $v = a_1 a_2 \dots a_n$ is a sequence of states $q_0, q_1, \dots, q_n$ starting with the initial state, such that $q_{i+1} = \delta(q_i, a_i)$ for each $0 < i \leq n$. We use $\mathcal{A}(v)$ to denote the state q_n the automaton reaches after processing v from its initial state q_0. By augmenting an automaton $\mathcal{A}$ with an acceptance condition we get an **acceptor**, a machine that accepts some words and rejects others. For the case of finite words, by augmenting $\mathcal{A}$ with a set $F \subseteq Q$ of accepting states, we obtain a **DFA** $\mathcal{D} = (\Sigma, Q, q_0, \delta, F)$. The DFA accepts the word v iff $\mathcal{D}(v) \in F$. We use $[\![\mathcal{D}]\!]$ for the set of words accepted by $\mathcal{D}$. Given a state $q \in Q$ we use $\mathcal{D}_{[q}$ for the DFA $(\Sigma, Q, q, \delta, F)$, i.e., for the DFA obtained from $\mathcal{D}$ by making q the initial state. We use $\mathcal{D}_{q]}$ for the DFA $(\Sigma, Q, q_0, \delta, \{q\})$, i.e., for the DFA obtained from $\mathcal{D}$ by making the set of final states the singleton $\{q\}$.

The class of **regular languages** consists of all languages that can be recognized by a DFA.

Residuality and the right congruence. For a language L over Σ and words $x, y \in \Sigma^*$ we use $x \sim_L y$ iff $\forall z \in \Sigma^*.\ xz \in L \iff yz \in L$. The relation $\sim_L$ is referred to as the **right congruence** of L. It is an equivalence relation. For a word $u \in \Sigma^*$ we use $[u]$ to denote the set of words forming the equivalence class in which u resides. The number of equivalence classes in this relations is referred to as the **index** of L. The Myhill–Nerode Theorem [58,60] states that for every regular language L, the index of $\sim_L$ is finite, and it equals the number of states in a minimal DFA. Moreover the states $\{q_1, q_2, \dots, q_n\}$ of the minimal DFA $\mathcal{D}$ for L can be mapped to the equivalence classes $E_1, E_2, \dots, E_n$ of $\sim_L$ such that for a state q_i, the language E_i is exactly $\mathcal{D}_{q_i]}$, i.e. the set of all words that take $\mathcal{D}$ from its initial state to state q_i.

Every state q_i of the minimal DFA $\mathcal{D}$ can also be mapped to the language R_i corresponding to $\mathcal{D}_{[q_i}$, i.e. to all words that take $\mathcal{D}$ from state q_i to an accepting state. The languages $R_1, R_2, \dots, R_n$ are called the *residual languages* of L. Formally, for a language L and a word u, the **residual of u with respect to** L, denoted $u^{-1}L$, is the set $\{v \mid uv \in L\}$. A language R is a residual language of L if there exists a word u such that $R = u^{-1}L$. Thus, by the Myhill–Nerode Theorem, the number of residual languages is finite and equals the number of states in the minimal DFA.

Note that while the languages $E_1, E_2, \dots, E_n$ partition Σ^* this is not true for the residual languages $R_1, R_2, \dots, R_n$. That is, a word w can belong to only one of the languages E_i, but it may belong to multiple languages R_i.

Right congruence and automata. A right congruence $\sim$ can be naturally associated with an automaton $\mathcal{A}_\sim = \langle \Sigma, Q, q_0, \delta \rangle$ as follows: the set of states Q are the equivalence classes of $\sim$. The initial state q_0 is the equivalence class $[\lambda]$. (Recall that we use λ for the empty word.) The transition function δ is defined by $\delta([u], \sigma) = [u\sigma]$. Similarly, given a deterministic automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$ we can naturally associate with it a right congruence as follows: $x \sim_{\mathcal{A}} y$ iff $\mathcal{A}(x) = \mathcal{A}(y)$.

3. Early results on learning regular languages

Identification in the limit. The first results on learning regular language were motivated by the desire to simulate the process of a child learning a language. For this reason, the model of *learning from text* assumed that at each time the learner is exhibited a word in the language, and answers with a concise description of what he thinks the language is. A learner is said to *identify a language L in the limit* if there exists a time point, after which all its answers are the same and constitute a concise description of L. Likewise, a learner is said to identify a class of languages $\mathbb{C}$ in the limit if it can identify in the limit any language $L \in \mathbb{C}$. In [34] Gold has shown that any class of languages that consists of finite languages and at least one infinite language, cannot be correctly identified in the limit from text. This in particular holds for regular languages.

Another model was then suggested, that of *learning from an informant*. In this model at every time step the informant provides the learner with a word in the alphabet, and the information whether it is in the language or not. As before the learner answers with a concise description of a language. The definition of identification in the limit remains the same. Under this model the regular languages (and the entire hierarchy of context-free languages, context-sensitive languages and primitive-recursive languages) are identifiable in the limit [34].

Learning from finite data. A sample S for a language L over Σ is a pair of sets of words $S^+ \subseteq \Sigma^*$ and $S^- \subseteq \Sigma^*$ such that $S^+ \subseteq L$ and $S^- \cap L = \emptyset$. That is, S^+ consists of *positive examples*, whereas S^- consists of *negative examples*. The model of *learning from finite data* assumes the learning algorithm **A** receives the entire sample $S = (S^+, S^-)$ at once and expects **A** to output a DFA $\mathbf{A}(S)$ that agrees with the given sample. For efficiency reasons, and in light of Occam's razor's principle stating that among all possible explanations of a phenomenon, the simplest is to be preferred, we would like the outputted DFA to be minimal.

A minimal DFA can be outputted using the method of *identification by enumeration*, which says: given an enumeration of all DFAs starting with the ones of smallest size and gradually growing, return the first DFA that agrees with S. Since

the number of DFAs of size n over a given alphabet of size k is quite large ($n2^{n-1}n^{(k-1)n}$ by [29]) this method is not very efficient. Ideally, we would like to find a polynomial method for returning the smallest DFA, or a small enough DFA.

Gold [35] has shown that the task of computing a minimal DFA that agrees with a given sample is hard. More precisely, he showed that the decision problem "given a sample S and an integer n, does a DFA with n states exist?" is NP-complete.

However, there exist various implementations for obtaining a minimal DFA that agrees with a given sample. Most build on the approach of Biermann and Feldman [16]. The idea of Biermann and Feldman is to associate with every word u in the sample (and every prefix thereof) a variable $\mathbf{u}$ representing the state that the minimal DFA reaches after reading u. Then, articulate the set of constraints these variables should adhere to. These are the following constraints[1]:

$$\{\mathbf{u} = \mathbf{v} \rightarrow \mathbf{ua} = \mathbf{va} \mid ua, va \in \mathit{Pref}(S), a \in \Sigma\} \tag{1}$$

$$\{\mathbf{u} \neq \mathbf{v} \mid u \in S^{+} \text{ and } v \in S^{-}\} \tag{2}$$

The first constraint states that if two words u and v in the sample (or prefixes thereof) reach the same state $\mathbf{u}$, then their one letter extensions ua and va should reach the same state $\mathbf{ua}$ for every $a \in \Sigma$. The second constraint states that two words u and v such that one is a positive example and the other is a negative example cannot reach the same state of the DFA.

We can now regard the problem as a *constraint-solving problem* and try to find assignments to all variables in a range of the form $\{1, 2, \ldots, n\}$. The assignment maps each variable $\mathbf{u}$ to one of the DFA states $\{1, 2, \ldots, n\}$. Thus, finding an assignment for the smallest possible such range that solves this set of equations guarantees finding a minimal DFA that agrees with the sample. See [62,36,37,48] for refinements and implementations of this approach.

Another fundamental approach, due to Oncina and Garcia [63], termed RPNI, guarantees identification of a (not necessarily minimal) DFA agreeing with the sample in polynomial time. In this approach one first constructs a *prefix tree automaton* (PTA) for the positive examples S^{+}. The PTA is a DFA with separate paths (modulo common prefixes) from the initial state (the root of the tree) to accepting states. In the second stage the algorithm tries to successively merge states, and a merge succeeds if the resulting DFA does not accept any of the negative examples S^{-}.

4. Query learning of regular languages via DFAs

In [3] Angluin suggested to examine the problem of learning with queries. Under this paradigm the learner communicates with a teacher that can answer certain types of queries about the language. Several types of queries can be considered. For instance, a membership query asks whether a word w is a member of L, and is answered either "yes" or "no". An equivalence query asks whether a concise representation of an hypothesis language L' is equal to L. In the case of an equivalence query, the answer may be "yes", in which case the learning algorithm has succeeded in exact identification of the target concept, or it may be "no", accompanied by a counterexample, that is, a word w in L but not in L' or vice versa. She has shown that there is neither a polynomial-time learning algorithm to learn a regular language from only membership queries [3] nor a polynomial-time learning algorithm to learn a regular language from only equivalence queries [5].[2] On the positive side, she proposed an algorithm termed $\mathbf{L}^*$ that learns regular languages using membership and equivalence queries in polynomial time.[3] The following section describes the $\mathbf{L}^*$ algorithm.

The idea behind $\mathbf{L}^*$. Let L be a language over a given alphabet Σ. As a thought experiment consider an infinite matrix M, whose rows and columns are titled with all words in Σ^*. If u_i is the title of the i-th row, and v_j is the title of the j-th column then the (i, j) element of the matrix, $M_{i,j}$, has 1 if $u_i v_j \in L$, and 0 otherwise. That is, the matrix records membership results. Consider a row title u_i. The row itself, an infinite sequence of 1's and 0's, corresponds exactly to the residual language of u_i w.r.t. to L. Indeed, $M_{i,j} = 1$ iff $v_j \in {u_i}^{-1}L$. Thus, by the Myhill–Nerode theorem the number of distinct rows in the matrix is finite.

While the matrix is infinite, two rows u_i and u_j can be distinguished by just one word v_k (for which $M_{i,k} \neq M_{j,k}$). Therefore only finitely many title columns are required to distinguish all the different rows. The $\mathbf{L}^*$ algorithm tries to discover the states of the minimal DFA. Each state candidate is associated with a row-title word u_i. Two state candidates u_i and u_j are considered different if there exists a column-title word v_k distinguishing them. The $\mathbf{L}^*$ algorithm uses a data structure called *an observation table* with which it tries to find the required distinctions.

Definition 4.1. An **observation table** $\mathcal{T}$ is a tuple (S, E, M) where S is a list of words, representing candidate *states*, E is a list of words containing λ, representing *experiments* trying to differentiate the elements of S, and M is a binary matrix of $|S|$ rows and $|E|$ columns. We say that $\mathcal{T}$ is an observation table for L if $M_{i,j} = 1$ iff $s_i e_j \in L$.

[1] For a set of words W we use $\mathit{Pref}(W)$ to denote the set $\{u \mid \exists w \in W, u \text{ is a prefix of } w\}$.

[2] The latter result assumes the counterexamples given by the oracle are arbitrary. In the case the oracle is guaranteed to return the lexicographically least counterexamples, there is a polynomial-time learning algorithm [39].

[3] For learning with other types of queries (subset, superset, disjointness and exhaustiveness) see [4].

By abuse of notation we use $M_{s,e}$ for $M_{i,j}$ where s is the i-th word of S, and e the j-th word of E. Similarly, we use M_s instead of M_i for the i-th row of the matrix, and for a subset $U \subseteq S$ where I_U is the set of indices of U, we use M_U for M_{I_U}, the matrix obtained by taking all the rows $i \in I_U$. The **row-index** of $\mathcal{T}$ is the number of distinct rows in M. The **column-index** of $\mathcal{T}$ is the number of distinct columns in M.

Definition 4.2. Let $\Sigma^*_<$ be a fixed enumeration of all words over Σ. If $\mathcal{T} = (\Sigma^*_<, \Sigma^*_<, M_L)$ is an observation table for L then $\mathcal{T}$ is said to be the **complete observation table** for L. For a language L we use $\mathcal{T}_L$ to denote the complete observation table for L. By the Myhill–Nerode theorem, if L is regular then the row-index of $\mathcal{T}_L$ is finite and equals the index of L. The column-index of $\mathcal{T}_L$ is also finite, and equals the index of L^{rev}, the reverse language of L. This is since the transpose of the matrix corresponds to the matrix of L^{rev} (which is also regular since regular languages are closed under reverse).

4.1. The L* algorithm

The **L*** algorithm starts with an empty observation table and gradually fills it using membership queries. When the table satisfies certain criteria it is possible to extract from it a DFA and ask an equivalence query. The criteria the table should satisfy is termed *closedness*.

Definition 4.3. An observation table $\mathcal{T} = (S, E, M)$ is said to be **closed** with respect to a subset B of S if it satisfies the following criteria

1. Initialization: $\lambda \in B$
2. Conseqution: $\forall s \in B.\ \forall \sigma \in \Sigma.\ s\sigma \in S$
3. Coverage: $\forall s \in S \setminus B.\ \exists s' \in B.\ M_s = M_{s'}$.

We say that two row titles $s, s' \in S$ are equivalent, denoted $s \equiv s'$, iff $M_s = M_{s'}$. A table is said to be **minimal w.r.t.** B if for every two row titles $s_1, s_2 \in B$ we have that $s_1 \not\equiv s_2$. Therefore, an observation table which is closed and minimal with respect to B induces an equivalence relations on the row titles, where B consists of one representative per each equivalence class. For a row title $s \in S$ we use $[s]$ for the row title $s' \in B$ such that $s \equiv s'$. If $\mathcal{T} = (S, E, M)$ is closed and minimal with respect to B then we can extract from it the DFA $\mathcal{D}^B_\mathcal{T} = (\Sigma, B, \lambda, \delta, F)$ where $\delta(s, \sigma) = [s\sigma]$ and $F = \{s \in B \mid M_{s\lambda} = 1\}$.

Claim 4.4. *Let $\mathcal{T} = (S, E, M)$ be the complete observation table for L. Let B be a subset of S such that $\mathcal{T}$ is closed and minimal w.r.t. B. Let $\mathcal{D}$ be the DFA $\mathcal{D}^B_\mathcal{T}$ constructed as above. Then for every $u \in B$ we have that $[\![\mathcal{D}_{[u}]\!] = u^{-1}L$.*

Proof. Assume towards contradiction this is not the case, and let v be such that $v \in [\![\mathcal{D}_{[u}]\!] \iff v \notin u^{-1}L$. Assume w.l.o.g. that $v \in [\![\mathcal{D}_{[u}]\!]$ though $uv \notin L$. Let q be the state that $\mathcal{D}_{[u}$ reaches after reading v. Then q is the state that $\mathcal{D}$ reaches after reading uv. By definition of $\mathcal{D}$ we have that q is the element of B such that the row of q equals the row of uv (i.e. $M_q = M_{uv}$). By definition of the accepting states of $\mathcal{D}$ we know that $M_{q,\lambda} = 1$. Thus $M_{uv,\lambda} = 1$ contradicting that $uv \notin L$. □

It follows that $\mathcal{D}^B_\mathcal{T}$ recognizes L, since its initial state recognizes $\lambda^{-1}L = L$. Since B is minimal, it follows that $\mathcal{D}^B_\mathcal{T}$ is the minimal DFA for L.

Corollary 4.5. *Let $\mathcal{T}$ be the complete observation table for a language L. If $\mathcal{T}$ is closed and minimal with respect to B, then $\mathcal{D}^B_\mathcal{T}$ is the minimal DFA for L.*

Since the construction of $\mathcal{D}^B_\mathcal{T}$ is insensitive to the set of columns E, it follows that for any observation table $\mathcal{T}'$ for L such that $\mathcal{T}'$ is closed and the row-index of $\mathcal{T}'$ is the same as the index of L, the DFA $\mathcal{D}^B_{\mathcal{T}'}$ is the minimal DFA for L.

Corollary 4.6. *Let $\mathcal{T}$ be an observation table for a language L with index n. If $\mathcal{T}$ is closed and minimal with respect to B and $|B| = n$, then $\mathcal{D}^B_\mathcal{T}$ is the minimal DFA for L.*

The **L*** algorithm, summarized in Algorithm 1 maintains an observation table $\mathcal{T} = (S, E, M)$ and a base set B. It initializes the lists S, E and B with $\langle\lambda\rangle$ and the entry $M_{\lambda,\lambda}$ with the result of the membership query on λ. It then gradually enhances its observation table and the set B until it obtains a closed table. The procedure *AddRow*(s) adds s to S and ask membership queries $\text{MQ}(s, e)$ for every $e \in E$. If $M_s \neq M_b$ for every $b \in B$ it adds s to B. Note that this keeps B minimal.

When the observation table is closed it extracts from it the DFA $\mathcal{A} = \mathcal{D}^B_\mathcal{T}$ and asks an equivalence query about $\mathcal{D}^B_\mathcal{T}$. If the answer is "yes" it returns $\mathcal{D}^B_\mathcal{T}$. Otherwise it gets a counterexample s'. In this case, it invokes *Find&AddCol*(s'). The procedure *Find&AddCol*(s') takes the counterexample s' and finds a word s to add to E that will distinguish a new state as follows. Let $v = s'$. Assume $|v| = k$. Let $s_0 = \lambda$, and for $1 \leq i \leq k$, let $s_i = \delta(\lambda, v[1..i])$ where δ is the transition relation of $\mathcal{D}^B_\mathcal{T}$. Then, s_i is the state that $\mathcal{D}^B_\mathcal{T}$ reaches after reading the prefix of length i of v. Consider the sequence of membership

Algorithm 1: $\mathbf{L}^*$.

oracles : MQ, EQ
members: Observation table $\mathcal{T} = (S, E, M)$, Base set B
methods : *IsClosed*, *IsMinimal*, *Find&AddRows*, *ExtractAut*
$S = \langle\lambda\rangle$, $E = \langle\lambda\rangle$, $B = \langle\lambda\rangle$ and $M_{\lambda,\lambda} = \text{MQ}(\lambda)$.
repeat
  $(a, s) = IsClosed()$
  if $a =$ "no" **then**
    $AddRow(s)$
  else
    $\mathcal{A} = ExtractAut()$
    $(a', s') = \text{EQ}(\mathcal{A})$
    if $a' =$ "no" **then**
      $Find\&AddCol(s')$
until $a' =$ "yes"
return $\mathcal{A}$

queries $\text{MQ}(s_0, v[1..k])$, $\text{MQ}(s_1, v[2..k])$, $\text{MQ}(s_2, v[3..k])$, …, $\text{MQ}(s_k, \lambda)$. Then the first query is the membership query on v and the last query is the result of $\mathcal{D}^B_{\mathcal{T}}$ on v. Since v is a counterexample, the first and last queries differ. Let i be the first index for which the result of the $(i+1)$-th query is different than the result of the i-th query. Then adding the column title $v[i+1..]$ will distinguish rows $s_{i-1}v[i]$ from s_i, though $\delta(s_{i-1}, v[i]) = s_i$. The procedure *Find&AddCol*(s') adds $v[i+1..]$ to E and asks the required membership queries to fill in the matrix. Since s_{i-1} is in B, by conseqution, the row $s_{i-1}v[i]$ must be in S. Let $u = s_{i-1}v[i]$ and $u' = s_i$ and $v' = v[i+1..]$. We now have that $M_{u,v'} \neq M_{u',v'}$, thus u' is different than u. In addition, for every other row $u'' \in B$, u' differs from u'' in the same column in $E \setminus \{v'\}$ that u differs from u''. Thus, u' is different from all rows in B and is added to B.

This shows that $\mathbf{L}^*$ converges. We claim that it converges in time polynomial in n, the index of the unknown language, ℓ, the maximal length of a counterexample, and k, the size of the alphabet. To see that, note that since each equivalence query yields a new row, the number of equivalence queries is bounded by the row-index, which is equivalent to n. Thus, the outer loop is repeated at most n times. Since a new column-title is added only upon processing a counterexample, the size of E is also bounded by n. The size of S is bounded by $n + kn$. Processing of a counterexample invokes at most ℓ membership queries, and filling the matrix invokes at most $|S| \times |E| = O(kn^2)$ membership queries.

Theorem 4.7 *([3]). Given an unknown regular language L with index n, the algorithm $\mathbf{L}^*$ returns a minimal DFA for L in time polynomial in n, ℓ and k, where ℓ is the size of the longest counterexample given, and k is the size of the alphabet.*

Example 4.8. We provide an example of a run of the $\mathbf{L}^*$ algorithm on the language $L = aba^*$, see Fig. 1. The algorithm starts with an observation table for $S = E = \{\lambda\}$, and fills the (λ, λ) entry using membership queries. To close the table it adds $\lambda \cdot a$ and $\lambda \cdot b$ to S and asks additional membership queries, obtaining the closed observation table encapsulated in blue. The rows of a and b are equivalent to the row of λ in this table, and thus a single state, λ, is identified. The algorithm invokes

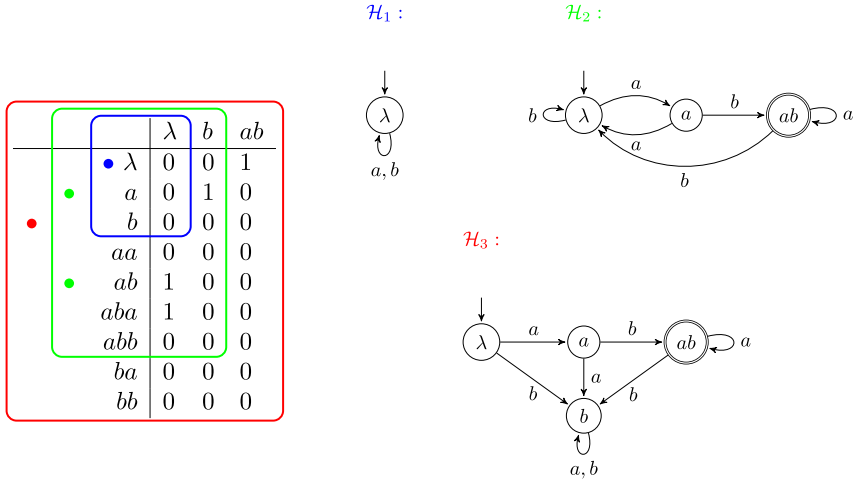


Fig. 1. An example of a run of $\mathbf{L}^*$ on target $L = aba^*$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

ExtractAut to extract the DFA $\mathcal{H}_1$ and asks an equivalence query. Since $\mathcal{H}_1$ does not recognize L the query is responded negatively, and a counterexample is returned.

Suppose the returned counterexample is $v = ab$. The algorithm then invokes *Find&AddCol*(ab). This process sets $s_0 = s_1 = s_2 = \lambda$ and asks the following series of membership queries: MQ$(\lambda \cdot ab)$, MQ$(\lambda \cdot b)$, and MQ$(\lambda \cdot \lambda)$, getting the answers 1, 0 and 0, respectively. The first index i for which the $i+1$'th answer is different from the i-th answer is 1. Thus, the suffix $v[2..] = ab[2..] = b$ is added to E, and the algorithm asks MQs to fill in the table.

In the extended table the row of a is different from that of λ so a is identified as a new state, and $a \cdot a$ and $a \cdot b$ are added to S. By asking additional MQs to fill in the table, ab is discovered to be a new state, and thus in order to close the table $ab \cdot a$ and $ab \cdot b$ are added to S. Filling in the table with the necessary MQs no additional distinct rows are observed, and thus the table encapsulated in green is the second closed table the algorithm obtains. Invocation of *ExtractAut* yields the DFA $\mathcal{H}_2$. The algorithm asks an equivalence query on $\mathcal{H}_2$ and is answered negatively.

Suppose the counterexample returned is $v = aaab$. The algorithm then invokes *Find&AddCol*$(aaab)$. This process sets $s_0 = \lambda$, $s_1 = a$, $s_2 = \lambda$, $s_3 = a$, $s_4 = ab$ and asks the following series of membership queries: MQ$(\lambda \cdot aaab)$, MQ$(a \cdot aab)$, MQ$(\lambda \cdot ab)$, MQ$(a \cdot b)$, and MQ$(ab \cdot \lambda)$, getting the answers 0, 0, 1, 1 and 0, respectively. The first index i for which the $i+1$th answer is different from the i-th answer is 2. Thus the suffix $v[3..] = aaab[3..] = ab$ is added to E, and the algorithm asks MQs to fill in the table. In the extended table the row of b becomes different than that of λ so b is identified as a new state, and $b \cdot a$ and $b \cdot b$ are added to S.

Filling the table with the missing MQs reveals no new states, and so the third obtained closed table is the one encapsulated in red. The algorithm invokes *ExtractAut* which yields the DFA $\mathcal{H}_3$ and asks an equivalence query on $\mathcal{H}_3$. This query is answered positively since $\mathcal{H}_3$ recognizes L and the algorithm terminates.

4.2. Implementations

The original formulation of $\mathbf{L}^*$, upon receiving a counterexample on an equivalence query, added it and all its prefixes to the rows of the observation table. Maler and Pnueli [53] suggested instead to add the counterexample and all its suffixes to the columns. Rivest and Schapire [66] observed that one can extract a single suffix to add to the columns. Kearns and Vazirani [44] suggested that since the purpose of the table is to distinguish states, a more compact representation achieving this is a decision tree, referred to as a *discrimination tree*. This suggestion reduces the MQ complexity to $O(kn^2 + n \log \ell)$. These ideas were combined and refined further in an algorithm called TTT [40] which outperforms the other variants of $\mathbf{L}^*$ in practice (though its upper bound complexity is the same as that of Kearns and Vazirani's). Some of these algorithms as well as other active automata learning algorithms are implemented in the open source libraries LibAlf [19] and LearnLib [41].

5. Query learning of regular languages via NFAs and AFAs

5.1. Extending the residuality notion to NFAs and AFAs

Regular languages can also be recognized by non-deterministic automata (NFAs) and alternating automata (AFAs), which are more succinct than DFAs. Specifically, NFAs can be exponentially more succinct than DFAs and AFAs may be exponentially more succinct than NFAs and doubly exponentially more succinct than DFAs [23]. Thus, a learning algorithm that yields NFAs and AFAs has a better potential to scale than an algorithm yielding DFAs.

The challenge in using NFAs and AFAs in a learning algorithm, is that most learning algorithm build on the residuality property (i.e. the property that each state of the automaton correspond to a residual language of the language to be learned) and NFAs and AFAs do not enjoy the residuality property. This problem motivated Denis, Lemay and Terlutte [28] to define *non-deterministic residual finite automata* (NRFA).[4] An NRFA is an NFA in which every state does correspond to a residual language. While most of the NFAs are not residual, some NFAs are. Dennis et al. showed that there exists a canonical NRFA for every regular language, the canonical NRFA has a minimal number of states (among all NRFA recognizing the same language), and a maximal number of transitions (among all NRFAs with a minimal number of states recognizing the language). Moreover, they showed that the canonical NRFA for a language L may be exponentially smaller than the minimal DFA for L. On the other hand, in some cases, the canonical NRFA may be exponentially bigger than the smallest NFA for L.[5]

Bollig et al. [18] have shown that the ideas of $\mathbf{L}^*$ can be used to learn NRFAs, and suggested a polynomial time algorithm termed $\mathbf{NL}^*$ for that. Angluin, Eisenstat and Fisman [9] defined ARFAs, alternating automata in which each state corresponds to a residual language, and investigated the problem of extending these ideas further to obtain a learning algorithm that yields AFAs. They established that ARFAs may be doubly-exponentially more succinct than DFA, and that there doesn't seem to be a natural candidate for a canonical ARFA. Roughly speaking, this has to do with the fact that a set of vectors doesn't in general have a unique minimal monotone basis, which also seem to be the root of the fact that finding a monotone basis is NP-complete [9]. In spite of this downside, they provided a polymomial time algorithm termed $\mathbf{AL}^*$ for learning regular languages via AFAs.

[4] They termed them RFSA for *residual finite state automata*.

[5] By duality this holds for canonical universal residual finite automata (URFA) as well [9] where, informally, an automaton is called universal if its branching is interpreted as conjunction rather than disjunction. See formal definition shortly.

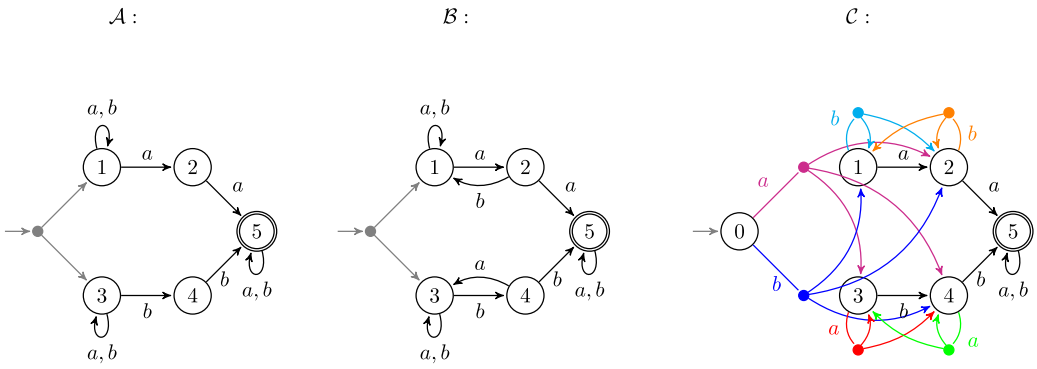


Fig. 2. Three AFAs for $L = \Sigma^* aa\Sigma^* \cap \Sigma^* bb\Sigma^*$ where $\Sigma = \{a, b\}$. Conjunction between edges is depicted by a bullet from which the edges split (the edges and the bullet share the same color for easier parsing). AFAs $\mathcal{B}$ and $\mathcal{C}$ are residual whereas AFA $\mathcal{A}$ is not.

In what follows we define alternating automata and residual automata, prove some properties on residual automata, and provide the learning algorithm **AL*** that generalizes **NL*** and **L***.

Alternating finite automata (AFAs). Let S be a finite set. We use $\mathbb{B}_+(S)$ to denote the set of Boolean expressions obtained from S by applying $\wedge$ (Boolean AND) and $\vee$ (Boolean OR) to $S \cup \{\text{true}, \text{false}\}$. An **alternating finite automaton (AFA)** is a tuple $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ where Σ is the alphabet, Q is a finite set of states, $I \in \mathbb{B}_+(Q)$ is the initial condition, $\delta : Q \times \Sigma \to \mathbb{B}_+(Q)$ is the transition relation and $F \subseteq Q$ is the set of accepting states.

Let $\mathbb{B}_\vee(S)$ (resp. $\mathbb{B}_\wedge(S)$) be the restriction of $\mathbb{B}_+(S)$ not to use $\wedge$ (resp. $\vee$). When the Boolean expressions in I and δ are restricted to $\mathbb{B}_\vee(Q)$, $\mathbb{B}_\wedge(Q)$, or Q we say that $\mathcal{A}$ is **non-deterministic** (NFA), **universal** (UFA), or **deterministic** (DFA), respectively.[6]

A DFA $\mathcal{D}$ has a single run on a given word. If the word is of length n the run consist of a sequence of $n+1$ states, capturing its transitions from the initial state, on reading the given word letter by letter. If the last state is accepting, the run is accepting, and we say that $\mathcal{D}$ accepts the given word. An NFA $\mathcal{N}$ may have many runs on a given word, and we say that it accepts a given word if one of the runs on that word accepts it. Intuitively, a UFA $\mathcal{U}$ has a single run, but the run is a tree rather than a sequence. For the run-tree to be accepting all leaves of the tree need to be accepting states. An AFA $\mathcal{A}$ may have several runs on a given word, each is a run-tree (which may in some cases be a single path). The automaton accepts a word if in one of these trees all the leaves are accepting.

Formally, we define acceptance of AFAs as follows. We extend δ to a function δ^* from $\mathbb{B}_+(Q) \times \Sigma^*$ to $\mathbb{B}_+(Q)$ as follows. For $q \in Q$, $\delta^*(q, \lambda) = q$. For a Boolean expression $\theta \in \mathbb{B}_+(Q)$ and a letter $\sigma \in \Sigma$, $\delta^*(\theta, \sigma)$ is obtained by replacing every $q \in \theta$ by $\delta(q, \sigma)$. Finally, for $w \in \Sigma^*$, $\delta^*(\theta, w\sigma) = \delta^*(\delta^*(\theta, w), \sigma)$. Next, given the accepting set F we denote $[\![\theta]\!]_F$ the result of evaluating a Boolean expression $\theta \in \mathbb{B}_+(Q)$ by assigning 1 to $q \in F$ and 0 to $q \notin F$. Then $w \in [\![\mathcal{A}]\!]$ iff $[\![\delta^*(\lambda, w)]\!]_F = 1$.

Residual automata.

Definition 5.1. Let $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ be an alternating automaton. We say that $\mathcal{A}$ is an **alternating residual automaton (ARFA)** iff $[\![\mathcal{A}_{[q]}]\!]$ is a residual language of $[\![\mathcal{A}]\!]$, for every state q of $\mathcal{A}$.

If $\mathcal{A}$ is non-deterministic or universal we say that it is a *non-deterministic residual automaton* **(NRFA)** or *universal residual automaton* **(URFA)**, respectively. We can also use **DRFA** for deterministic residual automata. By the Myhill–Nerode theorem, for every state q of the minimal DFA $\mathcal{D}$ of a language L we have that $[\![\mathcal{D}_{[q]}]\!]$ is a residual language of $[\![\mathcal{D}]\!]$. Thus the minimal DFA is a DRFA, and so is any trimmed DFA.[7]

Example 5.2. Let $\Sigma = \{a, b\}$ and $L = \Sigma^* aa\,\Sigma^* \cap \Sigma^* bb\,\Sigma^*$. Fig. 2 shows three AFAs $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ recognizing L. The AFA $\mathcal{A}$ is not an ARFA, while $\mathcal{B}$ and $\mathcal{C}$ are. The initial condition of $\mathcal{A}$ is $1 \wedge 3$. Since $[\![\mathcal{A}_{[1]}]\!] = \Sigma^* aa\Sigma^*$ and $[\![\mathcal{A}_{[3]}]\!] = \Sigma^* bb\Sigma^*$, we get $[\![\mathcal{A}]\!] = L$. The same is true for $\mathcal{B}$ thus $[\![\mathcal{B}]\!] = L$ as well.

To see that $\mathcal{B}$ is a residual automaton we have to show for each state i a representative word w_i, such that $w_i^{-1} L = [\![\mathcal{A}_{[i]}]\!]$. We give the following representatives: $w_1 = bb$, $w_2 = bba$, $w_3 = aa$, $w_4 = aab$, $w_5 = aabb$.

It is more challenging to see that $\mathcal{C}$ correctly recognizes L. To get intuition, one can check what states are "active" after reading a certain word, these correspond to splits of a conjunctive part of a transition. All active states need to reach an

[6] The name *universal* refers to the fact that the transitions correspond to $\forall$ quantifier rather than to $\exists$ quantifier as in the case of *non-deterministic* automata. This should not be confused with other usages of universal e.g. for *universal Turing machines* or *Conway's universal automata*.

[7] A DFA is said to be *trimmed* if for every state q there is a word w_1 taking the initial state to q, and a word w_2 taking q to an accepting state.

accepting state for the automaton to accept. After reading a, the active set is $\{2, 3, 4\}$, after reading aa, it is $\{5, 3, 4\}$ after reading aab it is $\{5, 4\}$ and after reading $aabb$ it is $\{5\}$. We note that $[\![\mathcal{B}_{[i}]\!] = [\![\mathcal{C}_{[i}]\!]$ for $i \in [1..5]$, and $[\![\mathcal{C}_{[0}]\!] = \lambda^{-1}L$. Thus $\mathcal{C}$ is also a residual automaton.

To see that $\mathcal{A}$ is not residual we note that $[\![\mathcal{A}_{[2}]\!]$ accepts only words beginning with a. But every word u in Σ^* can be extended by, say, the word $v = baaaabbbb$ beginning with b such that $uv \in L$. Thus, every residual language of L accepts some words beginning with b and hence $[\![\mathcal{A}_{[2}]\!]$ does not correspond to any of the residual languages of L. (That is, for every $u \in \Sigma^*$, $[\![\mathcal{A}_{[2}]\!] \neq u^{-1}L$.)

Succinctness. It is well known that NFAs and UFAs may be exponentially more succinct than DFAs, and AFAs may be exponentially more succinct than NFAs and UFAs, and doubly-exponentially more succinct than DFAs [23].[8] The same holds for their residual counterparts.

Theorem 5.3 *([9]). NRFA and URFA may be exponentially more succinct than DRFAs. ARFAs may be exponentially more succinct than NRFAs and URFAs, and doubly-exponentially more succinct than DRFAs.*

The proof of this theorem uses a general transformation of an automaton for L to a residual automaton for a language L' that is essentially as hard to recognize as L.

The idea behind **NL*** *and* **AL***. Suppose a given regular language L has index n, and its residual languages are $R_1, R_2, \ldots, R_n$. Then the minimal DFA $\mathcal{D}$ for L has n states $Q = \{q_1, q_2, \ldots, q_n\}$ such that $[\![\mathcal{D}_{[q_i}]\!] = R_i$. Suppose one of the residual languages is a union of two others, say $R_i = R_j \cup R_k$. Then we can build an NFA for L with states $Q \setminus \{q_i\}$ and replace all the incoming edges to q_i with edges to both q_j and q_k. The resulting NFA is residual since each state corresponds to a residual language. Thinking of this from the observation table point of view, we note that a row that can be represented as a bitwise Boolean-OR of two other rows corresponds to a residual language which is a union of two other residual languages. Therefore, if we want to extract from the table the required set of states, it is enough to find a subset B of rows S such that every row in $S \setminus B$ is a bitwise Boolean-OR of some rows in B. Likewise, if we would like to obtain a residual UFA, we should find a subset B of rows S such that every row in $S \setminus B$ is a bitwise Boolean-AND of some rows in B. Finally, if we would like to obtain a residual AFA, we should find a subset B of rows from S such that every row in $S \setminus B$ is a monotone Boolean combination of some rows in B.

Monotone, union and intersection bases. Let $V \subseteq \mathbb{B}^n$ be a set of vectors. For a Boolean expression $\theta \in \mathbb{B}_+(V)$ we use $[\![\theta]\!]$ for the element of $\mathbb{B}^n$ obtained by applying θ to V component-wise. E.g. if $n = 3$, $V = \{v_1, v_2, v_3, v_4\}$ for $v_1 = [000]$, $v_2 = [011]$, $v_3 = [101]$ and $v_4 = [110]$, and $\theta = v_2 \wedge v_3$ and $\theta' = v_1$ then $[\![\theta]\!] = [001]$ and $[\![\theta']\!] = [000]$. For a set of Boolean expressions Θ we use $[\![\Theta]\!]$ for $\cup_{\theta\in\Theta}[\![\theta]\!]$. A set of vectors $U \in \mathbb{B}^n$ is said to be

a **monotone basis** of V, if $V \subseteq [\![\mathbb{B}_+(U)]\!]$,
a **union basis** of V, if $V \subseteq [\![\mathbb{B}_\vee(U)]\!]$, and
an **intersection basis** of V, if $V \subseteq [\![\mathbb{B}_\wedge(U)]\!]$

If U is a monotone basis for V and a subset of V we say it is a **subset monotone basis** for V, otherwise we may use **general monotone basis** to emphasize that U need not be a subset of V. The definitions of **subset union basis** and **subset intersection basis** are analogous.

5.2. Finding an adequate basis

As mentioned previously, the learning algorithm gradually builds an observation table using membership queries. Two missing ingredients for the learning algorithm are (1) how to find a subset $B \subseteq V$ such that B is union (resp. intersection or monotone) basis for V, and (2) given $B \subseteq V$ and a vector $v \in V$, how to find, if possible, an expression b_v in $\mathbb{B}_\vee(Q)$ (resp., $\mathbb{B}_\wedge(Q)$ or $\mathbb{B}_+(Q)$) such that $[\![b_v]\!] = v$.

5.2.1. Finding a union or intersection basis

Because of the duality of union and intersection we can consider just one of these, and the results follow for the other.

Note that a vector u can be part of a union of vectors which is equal to v iff $u \leq v$ (where for $u, v \in \mathbb{B}^n$, $u \leq v$ if for no $i < n$ we have $u[i] = 1$ and $v[i] = 0$). Thus we can compute the union of all vectors $u \in B$ s.t. $u \leq v$, and v is equal to this union if and only if $v \in \mathbb{B}_\vee(B)$.

[8] The exact bounds depend on the exact definition of AFA. Kozen [45] defined AFAs in which the transition relation may also use negation, and showed that there exists an AFA with n states for which the smallest DFA requires 2^{2^n} states. We follow the definition of Chandra and Stockmeyer [23], who showed that there exists an AFA with n states for which the smallest DFA requires $2^{2^n/\sqrt{n}}$. Meyer and Fischer [57] used $\exists$ and $\forall$ states, and showed that a DFA for an AFA with $5n + 2$ states requires at least 2^{2^n}.

Claim 5.4. *Let $v \in \mathbb{B}^n$ and $B \subseteq \mathbb{B}^n$. There is a polynomial time algorithm to determine whether $v \in \mathbb{B}_\vee(B)$.*

Given a set of vectors B and a vector $u \in B$, we say that u is **union-redundant** for B if $u \in \mathbb{B}_\vee(B \setminus \{u\})$. We observe that if $u \in B$ is not union-redundant for B then it must appear in any subset union basis for B because it cannot be formed by union using the other elements of B.

Theorem 5.5 *([18]). Given a set of vectors V, there is a polynomial time algorithm to find a minimum cardinality subset union basis for V.*[9]

Note that this shows that there is a unique subset union basis of V of minimum cardinality, and by duality this is true also for intersection.[10]

5.2.2. Finding a monotone basis

Answering whether a vector v can be expressed as an adequate formula over B is more complicated for the monotone case, but can still be done in polynomial time.

The idea is as follows. Without loss of generality we can consider only formulas in DNF. Assume $v \in \mathbb{B}^n$ and $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{B}^n$. We can represent a DNF formula over B as a set of vectors in $\mathbb{B}^m$ (since a subset S of B can be represented as a vector u_S in $\mathbb{B}^m$ with $u_S[i] = 1$ iff $v_i \in S$). We can represent B as an $m \times n$ binary matrix, call it M. Suppose for some index $k \in [1..n]$, the vector v we are trying to find a DNF formula θ_v for, has $v[k] = 1$. Consider M^k, the k-th row of M. We have that $M^k[i] = 1$ in exactly those $i \in [1..m]$ such that $v_i[k] = 1$. Thus, if $M^k \neq \mathbf{0}$ and M^k is one of the vectors of θ_v, we will have $[\![\theta_v]\!][k] = 1$ as desired. If θ_v contains instead another vector $u \in \mathbb{B}^m$ such that $\mathbf{0} < u \leq M^k$, this is fine too (we still get $[\![\theta_v]\!][k] = 1$ as desired). However, if include such u in θ_v but there is a $k' \in [1..n]$ such that $v[k'] = 0$ and $M^{k'} \geq u$, then it won't hold that $[\![\theta_v]\!] = v$ since we will get $[\![\theta_v]\!][k'] = 1$ while $v[k'] = 0$. Thus v can be represented as a formula over B iff there are no $k, k' \in [1..n]$ such that $v[k] = 1$ and $v[k'] = 0$ yet $M^k \leq M^{k'}$. If there are no such violation we can construct θ_v as follows. Let I_1^v be the set of indices $k \in [1..n]$ such that $v[k] = 1$. For every $j \in I_1^v$ we can search for an index $j' \in I_1^v$ such that $M^{j'} \leq M^j$ and set θ_v to be the set of these $M^{j'}$s.

Claim 5.6 *([9]). Let $v \in \mathbb{B}^n$ and $B \subseteq \mathbb{B}^n$. There is a polynomial time algorithm to determine whether $v \in \mathbb{B}_+(B)$.*

Unlike the case with union or intersection, finding a monotone subset of minimum cardinality is NP-hard. This may not be surprising since unlike the case of union or intersection, there is no unique monotone basis.

Theorem 5.7 *([9]). Given a set of vectors V and a nonnegative integer k, it is NP-complete to determine whether there is a subset monotone basis U for V s.t. $|U| \leq k$.*

The proof of this theorem uses a polynomial time reduction from the problem of monotone not-all-equal 3-SAT.

Closed and minimal observation tables. Let $\mathcal{T} = (S, E, M)$ be an observation table. The definition of $\mathcal{T}$ being closed is different for **L***, **NL***, **UL***, and **AL***. We thus use X-**closed** for $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$. The definition of a table being closed is with respect to a set $B \subseteq S$. The first and second part is the same for all four: it requires that the empty word λ is in B and that for every $b \in B$, all its one-letter extensions $b\sigma$ for $\sigma \in \Sigma$ are in S. The third part requires the rows that are not in the base set B to be expressible by rows in the base set B, using the corresponding allowed set of Boolean expressions:

Definition 5.8 *(X-closedness).* An observation table $\mathcal{T} = (S, E, M)$ is said to be X-**closed** with respect to a subset B of S for $\mathrm{X} \in \{\mathrm{L}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$ if it satisfies the following criteria

1. Initialization: $\lambda \in B$
2. Conseqution: $\forall s \in B.\ \forall \sigma \in \Sigma.\ s\sigma \in S$
3. Coverage: $\forall s \in S.\ \exists \theta \in \mathbb{B}_\mathrm{X}(B).\ M_s = [\![\theta]\!]$ where $\mathbb{B}_\mathrm{X}(B)$ is $\mathbb{B}_+(B)$, $\mathbb{B}_\wedge(B)$, $\mathbb{B}_\vee(B)$, or B when X is A, U, N or L, respectively.

Next we introduce the notion of X-*minimality*. Intuitively, being closed with respect to B guarantees that all distinct rows are expressible by the allowed combinations of B rows. A trivial way to achieve this is to include all rows. Since we use B to derive the set of states of the learned automaton, we want it to be as small as possible. If there is a row in B that can be expressed by means of the other rows, we can remove it from B. Formally, let $\mathcal{T}$ be A-**closed** w.r.t. B. We say that $\mathcal{T}$ is

[9] Note that this question, of finding a **subset** union basis, is different from the question of finding a **general** union basis. The latter was proved NP-complete by Stockmeyer [69] who termed it **the set basis problem**.

[10] For those familiar with lattices, note that the minimum union (resp. intersection) basis of $\mathbb{B}^n$ is the set of join (resp. meet) irreducible elements in the lattice $(\mathbb{B}^n, \leq)$.

A-**minimal** w.r.t. B if for every $b \in B$ and $B' \subseteq B \setminus \{b\}$, $M_b \notin [\![\mathbb{B}_+(B')]\!]$. The notions of N-**minimal** and U-**minimal** observation tables are defined analogously.

From tables to automata. Let $\mathcal{T} = (S, E, M)$ be an observation table which is A-closed and minimal with respect to B. For $s \in S$ let $\theta_s \in \mathbb{B}_+(B)$ be a Boolean expression over B satisfying $\theta_s = s$ if $s \in B$, and otherwise $M_s = [\![\theta_s]\!]$. Then the tuple $(\Sigma, B, \lambda, \delta, F)$ where $F = \{b \in B \mid M_{b,\lambda} = 1\}$, and $\delta(b, \sigma) = \theta_{b\sigma}$, is an AFA, which we denote $\mathcal{A}^B_{\mathcal{T}}$. Applying these to N- and U-closed tables, we obtain an NFA and a UFA, respectively. Note that in the case of A-closed table, the choice of θ_s is not necessarily unique, while it is unique for N- and U-closed tables.

5.3. The **AL*** algorithm

Algorithm 2: xL* for X $\in$ {D, N, U, A}.

oracles : MQ, EQ
members: Observation table $\mathcal{T} = (S, E, M)$, Candidate states set B
methods : *Isx Closed*, *Isx Minimal*, x*Find&AddCol*, *In*$\mathbb{B}_X$, x*ExtractAut*
$S = \langle\lambda\rangle$, $E = \langle\lambda\rangle$, $B = \langle\lambda\rangle$ and $M_{\lambda,\lambda} = \text{MQ}(\lambda)$.

```
repeat
    (a1, s1) = IsxClosed()
    if a1 = "no" then
        AddRow(s1)
    else
        (a2, s2) = IsxMinimal()
        if a2 = "no" then
            RemoveFromBase(s2)
        else
            A = xExtractAut()
            (a3, s3) = EQ(A)
            if a3 = "no" then
                xFind&AddCol(s3)
until a3 = "yes"
return A
```

Algorithm 2 presents a scheme for the algorithms **AL***, **NL***, **UL***, and **L*** for learning a regular language using membership and equivalence queries yielding AFAs, NFAs, UFAs, and DFAs, respectively.

As in **L*** the algorithm gradually builds the observation table asking membership queries and trying to close it and keep it minimal. Since the question of finding a minimum subset monotone basis is NP-hard, the algorithm proceeds greedily, maintaining, in addition to its observation table $\mathcal{T} = (S, E, M)$, a subset B of S that is the current candidate for extracting an adequate basis. In each iteration of its main loop, it checks whether $\mathcal{T}$ is closed with respect to B, by calling procedure *Isx Closed*(). If the answer is "no", the procedure returns a word s_1 that cannot be expressed by $\mathbb{B}_X(B)$ and the algorithm adds s_1 to B and loops. Once $\mathcal{T}$ is closed with respect to B the algorithm checks whether $\mathcal{T}$ is minimal with respect to B, by calling procedure *Isx Minimal*(). If the answer is "no", the procedure returns a word $s_2 \in B$ that can be expressed by $\mathbb{B}_X(B \setminus \{s_2\})$. The algorithm then removes s_2 from B, and loops. Once $\mathcal{T}$ has been found to be x-closed and minimal with respect to B, the algorithm calls x*ExtractAut*() to obtain an x-automaton $\mathcal{A}$, on which it asks an equivalence query. If the answer is "yes", the algorithm returns $\mathcal{A}$. Otherwise the algorithm uses the counterexample s_3 provided by the equivalence oracle to extract experiments to add to E.

For the algorithm to converge the processing of a counterexample should increase our knowledge on the unknown language. In the **L*** algorithm we have seen (in Section 4) that processing of a counterexample yields at least one new row in the table (i.e. a row distinct from all rows in B). For **AL*** processing of a counterexample is guaranteed to yield one new column (i.e. a column distinct from all columns in the table). Intuitively, as is the case in **L*** the current observation table does not have sufficient information to differentiate all states of the minimal DFA, and one of the counterexample's suffixes reveals such new information. For **AL*** this means that a monotone formula θ_s that covers a row s in the current observation table (i.e. $[\![\theta_s]\!] = [\![s]\!]$), does not cover it when the row is extended with the new information (extracted from the respective suffix of the counterexample). If the projection of the new column to the rows of s and the rows used in θ_s was the same as all the other columns (projected on these rows) then θ_s would still cover s, contradicting that the suffix revealed new information on s. The same argument can be applied to **NL***, **UL*** and **L***. A convergence argument for **NL*** that does not rely on the number of columns can be found in [18].[11]

We explain first the workings of A*Find&AddCol*. The procedure A*Find&AddCol* examines all suffixes of s_3. For each suffix it checks whether it induces a new column, when projected on the words in B and their one-letter extensions. Formally, let $M_{B\Sigma}$ be the matrix obtained from M by keeping only rows corresponding to words in $B \cup B\Sigma$. The procedure

[11] The argument works by tracking the increase or decrease of four parameters: the number of distinct rows in the table, the overall number of rows in the table, the number of rows in the base, and the number of pairs of different rows in the table [18].

Table 1
Query complexity of **L***, **NL*** and **AL***.

	L*	**NL***	**AL***
EQ	n	$O(n^2)$	m
MQ	$O(k\ell n^2)$	$O(k\ell n^3)$	$O(k\ell nm)$

A*Find&AddCol*(s_3) computes for each suffix e of s_3 that is not already in E, its corresponding column, by calling MQ(se) for every word corresponding to a row of $M_{B\Sigma}$. If this column is not a column of $M_{B\Sigma}$ it adds e to E and the obtained MQ results to M.

Note that all the procedures *Is*x*Closed*(), *Is*x*Minimal*(), x*Find&AddCol*() and x*ExtractAut*() may invoke calls to MQ, and all but the latter use *In*$\mathbb{B}_{\mathrm{x}}$() as a sub-procedure.

Clearly, if the algorithm terminates it returns an automaton recognizing the unknown language L. The complexity of the algorithm is expressed as a function of four parameters: the row-index of the complete observation table, denoted n; the column-index of the complete observation table, denoted m; the maximum length of a counterexample returned by the equivalence oracle, denoted ℓ, and the size of the alphabet, denoted k. (Recall that the row-index and column-index of the complete observation table for L, as defined in Definition 4.2, are equivalent to the number of states in the minimal DFA for L and L^{rev}, respectively.)

We note that unlike in the case of **L***, for the algorithms **NL***, **UL*** and **AL*** it may be that an intermediate automaton has more state than the final automaton. This is since it may be that a minimum union/intersection/monotone basis of a subset U' of the final sets of distinct rows U has cardinality greater than a minimum basis of U. To see that **AL*** indeed terminates we note that each call to the equivalence oracle, results in the addition of word e to E that induces at least one more new column in M. Thus, the main loop is executed at most m times. Each call to procedure *AddRow* results in the addition of a word s to S introducing a new row in M, and thus can be called at most n times. The size of B is bounded by n, thus each iteration of the main loop involves at most n calls to *RemoveFromBase*. The processing of the counterexample by *Find&AddCol* goes over all suffixes of the counter example, at most ℓ, and all rows of $B \cup B\Sigma$, at most $n+nk$. Thus overall running time of the algorithm is bounded by $poly(m, n, \ell, k)$.

Theorem 5.9. *The algorithm* **AL*** *returns an AFA for an unknown language L, after at most m equivalence queries, and* $O(mn\ell k)$ *membership queries, where n and m are the row-index and column-index of* $\mathcal{T}_L$*, respectively,* ℓ *is the length of the longest counterexample, and* $k = |\Sigma|$.

Note that for **NL*** (resp. **UL***) the returned NFA (resp. UFA) is the canonical NRFA (resp. URFA) whereas for **AL*** since a monotone basis is not unique, we have no definition of a canonical ARFA. Moreover, the AFAs returned by **AL*** are not guaranteed to be residual. This is fine, since the motivation for using **AL*** is obtaining a smaller automaton rather than a residual automaton.

If for some reason one does insist on obtaining an ARFA, it is possible to add an extra step to **AL*** that guarantees the returned AFA is residual [15]. The idea is as follows. If the base set is prefix closed, the emitted AFA is guaranteed to be residual. This is since, when the base set is prefix closed, if u is a string in the base, then the state u is the only state that is reachable by reading u from the initial state, and thus $[\![\mathcal{A}_{[u]}]\!] = u^{-1}[\![\mathcal{A}]\!]$. If the set of rows S is prefix closed, and the base set B covers S, then the obtained AFA is still residual, since it can be obtained by a process removing states in $S \setminus B$ that does not change the accepted language or harm residuality. Thus, if we change **AL*** so that when an equivalence query returns "yes" we first close S for prefixes, then close the table, and B' is the revised base, then the AFA extracted taking B' as the base, is residual [15].

Comparing **L***, **NL*** *and* **AL***. The number of EQ and MQ asked by the **L***, **NL***, and **AL*** algorithms w.r.t. the row index n, the column index m, the size of the alphabet k and the length of the maximal counterexample ℓ is summarized in Table 1.

One can see that from **L*** to **NL*** there is an increase in a factor of $O(n)$, whereas from **L*** to **AL*** a factor of n is replaced by a factor of m. It is hard to compare the row index n and the column index m. Since the transposed matrix of an observation table for L is an observation table for L^{rev}, the reversed language of L, it follows that m is the row index of the reversed language. Leiss [46] showed that in the worst case the minimal DFA for L^{rev} may be exponentially bigger than that of L. However, for symmetry reasons, the minimal DFA for L^{rev} may also be exponentially smaller than that of L. One can argue that the column-index is a complexity measure of the given language L itself.

An empirical study comparing the four algorithms **L***, **NL***, **UL***, and **AL*** was reported in [9]. There, they randomly generated DFAs, NFAs, UFAs, and AFAs and have compared the four different algorithms in terms of the size of the resulting automaton, and the number of EQ and MQ queries issued. They report that in terms of the number of states in the resulting automaton, the **AL*** algorithm always produced smaller automata (as expected); that in terms of number of MQ queries the **xL*** algorithm outperforms the others when the targets are xFAs (for x$\in$ {D,N,U,A}); and that in terms of number of EQ queries the **L*** algorithm is always preferable.

6. Query learning of richer formalisms

The $\mathbf{L}^*$ algorithm can be easily adapted to learn input-output machines such as Moore machines. In a Moore machine every state is associated with an output symbol that is emitted when the automaton reaches that state. Thus, the values in an observation matrix can range over the set of outputs rather than the set of Boolean values. A *membership query* is replaced with an *output query*, a query that receives a word w as input and returns the output emitted by the machine after reading w. Adaption to learning Mealy machines, where outputs are emitted on the edges, can be found in [61]. A more general model, an I/O-Automata, allows several inputs to be read at each step (or none at all) and several outputs to be emitted in each state (or none at all). An extension of $\mathbf{L}^*$ to I/O-Automata was provided in [1]. The extension in [20] considers transducers with lookback, where the input and output labels of a transition can refer to any of the last k read letters (for some fixed k). In [17] Bojánczyk proposes a semantics of transducers that records the origin information. He provides notions of left and right derivatives for a transducer with origin information, and shows that a transducer is regular if and only if it has both finitely many left derivatives and finitely many right derivatives. From this property he derives an extension of $\mathbf{L}^*$ to learn transducers with origin information.

Maler and Mens [52] study an extension of $\mathbf{L}^*$ to learn *symbolic automata ranging over ordered alphabets* such as the integers or reals. The edge labels are then ranges such as $[2, 7)$. These automata are deterministic in the sense that for any given state, the ranges on the outgoing edges have an empty intersection. The algorithm is then an elegant generalization of $\mathbf{L}^*$. Each row label of the observation table is a symbolic word, but it is also associated with a concrete representative word. Each new counterexample either (1) creates a new state as in the $\mathbf{L}^*$ algorithm, or (2) refines the current partition to intervals (i.e. introduces a new symbolic symbol for an intermediate interval) or (3) modifies the current partition to intervals; e.g. a partition to [7, 13) and [13, 19) can be changed to [7, 11) [11, 19). This idea was generalized by Drews and D'Antoni [30] to learn symbolic automata over general effective Boolean Algebras. The learning algorithms and their evaluation rely on the existence of adequate partitioning functions for the respective Boolean algebra.

There has been work on extending $\mathbf{L}^*$ to learn *deterministic weighted automata* [14,13]. Weighted automata are automata where transitions are associated with weight. The value of a run is the sum of weights on the traversed transitions. The value of a word is the minimal value of a run processing that word. The extensions of $\mathbf{L}^*$ to learn deterministic weighted automata are based on the notion of *Hankel Matrix* and its minimum rank which is the generalization of the observation table of Angluin and the row index, and Fliess's theorem [33] which is a generalization of the Myhill–Nerode theorem to the weighted automata setting. The topic of extending $\mathbf{L}^*$ to learn *register automata*, automata that are extended with a set of register that can store data has also been studied. The main challenge is finding a Myhill–Nerode characterization to this model, or restrictions thereof, so that the residuality property can be used [38,22].

An approach trying to unify learning algorithms via category theory is studied in [72].

7. Regular ω-languages

ω-Automata. Reactive systems are systems that maintain an ongoing interaction with their environment. As such their computations are naturally viewed as infinite words. The set of computations of a system can be recognized by an ω-acceptor, an acceptor processing infinite words. A **deterministic ω-automaton** is a tuple (Σ, Q, q_0, δ) defined in the same way as a deterministic automaton. A run of a deterministic ω-automaton on a ω-word $w = a_1a_2a_3\ldots$ is an infinite sequence of states $q_0, q_1, q_2, \ldots$ starting with the initial state q_0 such that $q_i = \delta(q_{i-1}, a_i)$ for all $i > 0$. By augmenting an ω-automaton with an acceptance condition α we obtain an **ω-acceptor**. Various acceptance criteria for ω-acceptors have been suggested in the literature, Büchi, Rabin, Streett, Muller, and parity. All make use of the notion $inf(r)$, the set of states visited infinitely often during the run r. Not all ω-acceptors have the same expressive power. For this exposition it suffices to define only parity acceptors. A **parity** acceptance condition is a mapping $\kappa : Q \to \{1, \ldots, k\}$ of the states to a number in a given bounded set $\{1, \ldots, k\}$, which we refer to as **color**. For a subset $Q' \subseteq Q$, we use $\kappa(Q')$ for the set $\{\kappa(q) \mid q \in Q'\}$.

- A run r is **accepting** according to **parity condition** κ if the maximal color visited infinitely often is even. That is, if $\max(\kappa(inf(r)))$ is even.

We use $[\![\mathcal{P}]\!]$ for the set of words accepted by $\mathcal{P}$. A parity automaton is said to be **weak** if no two strongly connected states have distinct colors, i.e., if looking at the partition of its states to maximal strongly connected components (MSCCs) all states of an MSCC have the same color. Clearly every weak parity automaton can be colored with only two colors, one even and one odd, in which case the colors are often referred to as *accepting* or *rejecting*.

Example 7.1. Consider the four deterministic parity automata (DPAs) $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$ and $\mathcal{S}$, depicted in Fig. 3. The colors of the states are given by the number of circles around the state. E.g. for $\mathcal{S}$ we have $\kappa(1) = \kappa(4) = 1$, $\kappa(2) = 2$ and $\kappa(3) = 3$. We will use this convention henceforth. We have that $[\![\mathcal{P}]\!] = \Sigma^\omega \setminus (\Sigma^* bb \Sigma^\omega)$, $[\![\mathcal{Q}]\!] = a^*ba^\omega + a^*ba^*ba^*b\Sigma^\omega$, $[\![\mathcal{R}]\!] = (a^*b)^\omega$, and $[\![\mathcal{S}]\!] = (a + ba + bba)^*(a^*ba)^\omega$. The DPAs $\mathcal{P}$ and $\mathcal{Q}$ are weak, while the DPAs $\mathcal{R}$ and $\mathcal{S}$ are not weak. Clearly we can alter $\mathcal{Q}$ to use just two colors without changing the language it accepts. This does not hold for $\mathcal{S}$ which cannot be recognized by a DPA using less than three colors.

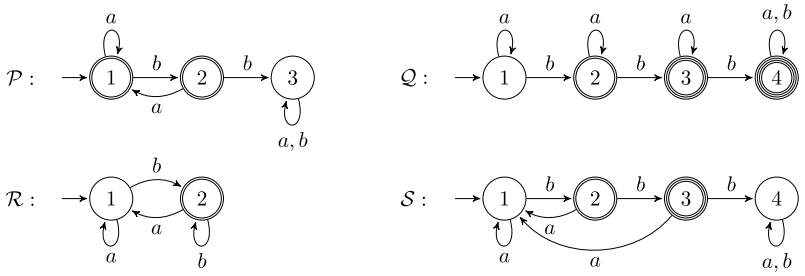


Fig. 3. Four DPAs.

We use DPAs for deterministic parity automata, and $\mathbb{DP}$ to denote the class of languages recognized by DPAs. We use $\mathbb{DW}$ to denote the class of languages recognized by weak DPAs. For $\mathbb{k} \in \mathbb{N}$ we use $\mathbb{DP}_{\mathbb{k}}$ to denote the class of languages recognized by DPAs using at most $\mathbb{k}$ colors. Thus $\mathbb{DP} = \cup_{\mathbb{k}\in\mathbb{N}} \mathbb{DP}_{\mathbb{k}}$. The following relations about the expressive power of the classes have been established through a series of works.

Claim 7.2 *([75,68,51,26]).*

- $\mathbb{DP}_{\mathbb{k}+1} \supsetneq \mathbb{DP}_{\mathbb{k}}$ *for every* $\mathbb{k} \in \mathbb{N}$,
- $\mathbb{DP}_{\not\in} \supsetneq \mathbb{DW}$.

Note also that $\mathbb{DP}_{\mathbb{1}\mathbb{1}}$ contains only the trivial automata. The class of **regular ω-languages** is the class of languages accepted by DPAs, i.e. $\mathbb{DP}$. We refer to the languages accepted by weak DPAs, namely $\mathbb{DW}$, as the class of **weak regular ω-languages**. A well known subset of the weak ω-regular languages is the class of **safety languages**. Formally, these are the languages that can be expressed by weak DPAs where no accepting MSCC is reachable from a rejecting MSCC. Intuitively, these are the languages that can express properties saying that something bad never happens. For example, out of the four DPAs in Fig. 3 only $\mathcal{P}$ recognizes a safety language, the language in which there are no two consecutive occurrences of b.

Finite representations of ω-words. In the context of learning ω-automata using membership queries, the question how can an ω-word be finitely represented arises. Two approaches have been used in the literature of learning ω-languages: one uses prefixes of ω-words, and the latter uses ultimately periodic words. An **ultimately periodic word** is a word of the form uv^ω for finite words $u \in \Sigma^*$ and $v \in \Sigma^+$. That is, it is an ω-word with prefix u, followed by an infinite repetition of v. The ultimately periodic word uv^ω can be finitely represented via the pair (u, v). It follows from McNaughton's theorem [56] that two regular ω-languages are equivalent iff they agree on the set of ultimately-periodic words.

Residuality and right congruence of ω-automata. The **right congruence of an ω-language** L is defined similarly to the finite words case, only that the suffixes are assumed to be ω-words. Formally, let $x, y \in \Sigma^*$, then $x \sim_L y$ iff $\forall w \in \Sigma^\omega$ we have $xw \in L \iff yw \in L$. By the above, when L is a regular ω-language we can equivalently consider only ultimately periodic words as suffixes and define $x \sim_L y$ iff $\forall u \in \Sigma^*, v \in \Sigma^+$ we have $xuv^\omega \in L \iff yuv^\omega \in L$.

Recall from Section 2 that a right congruence $\sim$ can be naturally associated with an automaton $\mathcal{A}_\sim$ and an automaton $\mathcal{A}$ can be naturally associated with a right congruence $\sim_{\mathcal{A}}$. As is the case for regular languages over finite words, the right congruence $\sim_{\mathcal{D}}$ associated with an ω-acceptor $\mathcal{D}$ recognizing a regular ω-language L, refines $\sim_L$. However, none of the traditional ω-automata have a Myhill–Nerode characterization, in the sense that the states of the automaton can be mapped to the equivalence classes of $\sim_L$. In fact, the right congruence relation is not informative enough when it comes to ω-languages. Consider for instance the language $L = (a + b)^* b^\omega$. For every $x, y \in \Sigma^*$ and every $w \in \Sigma^\omega$ it holds that $xw \in L \iff yw \in L$. Indeed, membership in L is determined solely by examining the suffix. Thus the equivalence relation $\sim_L$ has only one equivalence class, but clearly any ω-automaton recognizing L requires at least two states.

8. Identification in the limit of regular ω-languages

For this reason the question of learning ω-regular languages was open for many years [47]. Early attempts have managed to learn the safety languages. Particularly, de la Higuera and Janodet [27] give positive results for polynomially learning in the limit *safe* regular ω-languages from *prefixes*, and negative results for learning any strictly subsuming class of regular ω-languages from prefixes. This work was extended in [42] to learning bi-ω languages from subwords, where a bi-ω word is a two-sided infinite word. Saoudi and Yokomori [67] considered ultimately periodic words and provided an algorithm claimed to efficiently learn in the limit any unknown safety language from positive data and restricted subset queries. The efficiency claim about the provided algorithm was found to be incorrect [6], but Angluin showed that the class $\mathbb{DW}$ (which subsumes the safety languages) is identifiable in the limit from positive data and membership queries with polynomial time and data [6].

9. Query learning of weak regular ω-languages

While the right congruence relation is not informative enough for regular ω-languages, it is informative enough for *weak* regular ω-languages. Indeed, Staiger has shown that every weak ω-regular language L is accepted by an automaton isomorphic to $\sim_L$ [68].[12] Maler and Pnueli [53] devised a learning algorithm, which we term $\mathbf{L}^{\mathrm{W}\omega}$, to learn the weak regular ω-languages using membership and equivalence queries. This section describes $\mathbf{L}^{\mathrm{W}\omega}$.

We adopt the notation $\mathcal{P}_{q]}$ from Section 2 to the ω-word case. That is, given a parity automaton $\mathcal{P} = (\Sigma, P, p_0, \delta, \kappa)$ we use $\mathcal{P}_{q]}$ for the DFA $(\Sigma, P, p_0, \delta, \{q\})$.

Claim 9.1 *([68]). Let L be a weak ω-regular language. There exists a weak DPA $\mathcal{P}$ such that for each state q of $\mathcal{P}$ and each word w_q in $\mathcal{P}_{q]}$ we have $[w_q] = \mathcal{P}_{q]}$.*

Example 9.2. Consider the weak DPA $\mathcal{Q}$ depicted in Fig. 3. The relation $\sim_{[\![\mathcal{Q}]\!]}$ has 4 equivalence classes: $[\lambda]$, $[b]$, $[bb]$ and $[bbb]$. We have that $[\![\mathcal{Q}_{1]}]\!]=a^*=[\lambda]$, $[\![\mathcal{Q}_{2]}]\!]=a^*ba^*=[b]$, $[\![\mathcal{Q}_{3]}]\!]=a^*ba^*ba^*=[bb]$, and $[\![\mathcal{Q}_{4]}]\!]=a^*ba^*ba^*(a+b)^*=[bbb]$.

Consider the weak DPA $\mathcal{P}$ depicted in Fig. 3. The relation $\sim_{[\![\mathcal{P}]\!]}$ has 3 equivalence classes: $[\lambda]$, $[b]$ and $[bb]$. We have that $[\![\mathcal{P}_{1]}]\!]=(a^*ba)^*=[\lambda]$, $[\![\mathcal{P}_{2]}]\!]=(a^*ba)^*b=[b]$, and $[\![\mathcal{P}_{3]}]\!]=(a^*ba)^*bb(a+b)^*=[bb]$.

ω-Observation table. Building on this property of weak regular ω-languages, one can imagine an extension of $\mathbf{L}^*$ to learn weak ω-languages. The algorithm will store information on membership in the language using an observation table similar to the finite word case. The row titles remain finite words, but for column titles we take ultimately periodic words. This is since concatenation in the ω-word case is well defined only if its left argument is a finite word, and its right argument is an ω-word, and by McNaughton's theorem it suffices to consider ultimately periodic words for the right argument of concatenation. To the definition of a **closed ω-observation table** an additional requirement is added: that the set E of column titles is suffix closed. Note that in the case of ω-words the suffix property is not an order relation over Σ^ω: for instance, $(ab)^\omega$ and $(ba)^\omega$ are suffixes of each other. Given an ultimately periodic word uv^ω its set of suffixes consists of all words of the form $u'v^\omega$ where u' is a suffix of u, as well as all words of the form $(v')^\omega$ where v' is a rotation of v.

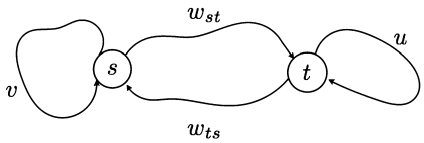
From tables to automata. From an ω-observation table $\mathcal{T}$ closed with respect to B we can extract an automaton in the same way as in the finite word case. That is, $\mathcal{A}_\mathcal{T} = (\Sigma, B, \lambda, \delta)$ where $\delta(s, \sigma) = [s\sigma]$. To make $\mathcal{A}_\mathcal{T}$ an acceptor we need to define the acceptance condition. Since we work with weak regular ω-languages, we need to assign the states an accepting/rejecting colors, so that no MSCC has two states with contradicting colors.

In the case of finite words, a state s is determined to be accepting if $M_{s\lambda} = 1$, i.e. if the word s is in the language. Here, a state s should be accepting if for all words v that loop back to s, we have that sv^ω is in the language. So the algorithm can check if this holds for all words v^ω in the column titles. For weak automata, we need to maintain the property that each pair of strongly connected states (i.e. each pair of states s, s' residing in the same MSCC) agree on this marking. (For this reason, we insist that the set of title columns is suffix closed.) If this does not hold a *conflict* is detected. A **conflict** is a triple (s, u, v) such that $\mathcal{A}_{[s}$ loops on both u and v yet either $su^\omega \in L$ and $sv^\omega \notin L$ or vice versa. If this is the case whatever marking we give s (accepting/rejecting) this will contradict the result of membership for one of the words su^ω or sv^ω. The algorithm should be able to extract information from the conflict that will lead to the discovery of more states.

Conflict detection. Procedure *MarkOrDetect*() takes as input an observation table $\mathcal{T} = (S, E, M)$ closed with respect to a base set B, and the extracted automaton $\mathcal{A}$, and returns either a consistent marking of $\mathcal{A}$'s MSCCs to accepting/rejecting or a detected conflict. It first computes all the MSCCs of $\mathcal{A}$. Then for each MSCC Q of $\mathcal{A}$ it proceeds as follows. It marks the MSCC as "unmarked", and marks each state as "un-associated". For each state $s \in Q$ and each $e \in E$, if $e = v^\omega$ and $\mathcal{A}_{[s}$ loops on v it operates as follows:

- If Q is unmarked, it marks Q with M_{se} (i.e. Q gets a mark in $\{1, 0\}$ and it is marked 1 iff $se \in L$) and associates s with witness v.
- If Q is marked a and $a = M_{se}$ (i.e. there is no contradiction to the previous marking) and s is "un-associated" it associates s with witness v.
- Otherwise, Q is marked with a and $a \neq M_{se}$. It then finds a state t in Q that is associated with some witness u. It follows that $s(v)^\omega \in L \iff t(u)^\omega \notin L$. If $s = t$ it returns *Conflict*(s, v, u). Otherwise, it finds a word w_{st} taking s to t and a word w_{ts} taking t to s. Two such words must exists since Q is strongly-connected. It then checks whether $s(w_{st}w_{ts})^\omega$ is in L. If the answer agrees with a then it disagrees with $s(v)^\omega$ and the procedure returns *Conflict*$(s, v, w_{st}w_{ts})$, otherwise the answer disagrees with $t(u)^\omega$ and the procedure returns *Conflict*$(t, u, w_{ts}w_{st})$.

[12] Note that the language $L = (a + b)^*b^\omega$ given to show that $\sim_L$ is not informative enough is not weak.



If no conflict is detected during the above process, it returns the marking of the MSCCs.

Conflict resolution. Given a conflict (s, u, v) the constructed automaton $\mathcal{A}$ loops from s on both words u and v, though one of the words su^ω and sv^ω is accepted by the unknown language L while the other is rejected. In a DPA $\mathcal{P}$ that correctly recognizes L either su leads to a state different than s or sv does. Let's assume w.l.o.g. that sv leads to a different state. The reason that in $\mathcal{A}$ both su and sv lead to s is that in our observation table there is no experiment (column title) distinguishing s and sv. To resolve the conflict we need to find a column title to add to the table that will distinguish s and sv.

The conflict resolution builds on the special structure of weak DPAs. Suppose the DPA for the target language has less than n states. Then reading sv^n we are bound to repeat a state q that we already visited, so reading sv^ω will loop on q as well. Assume w.l.o.g. that $su^\omega \in L$ and $sv^\omega \notin L$. Consider the word $sv^n u^\omega$.

If $sv^n u^\omega \notin L$ then there exists some $k \in [0..n]$ such that $sv^k u^\omega \in L$ and $sv^{k+1} u^\omega \notin L$. Thus, the word $v^k u^\omega$ distinguishes between s and sv and we can add it as a column title.

Otherwise, $sv^n u^\omega \in L$. Therefore the word $sv^n u^n$ must exit the MSCC in which q resides, which is a rejecting MSCC, and lead to an accepting MSCC. Since the number of MSCCs is bounded by n the number of alternations between accepting and rejecting MSCCs is bounded as well. Thus there exists some word $w \in \Delta^n_{u,v}$ distinguishing s and sv where $\Delta^n_{u,v}$ is the set of words of the form xu^ω or xv^ω where x is a prefix of $(u^n v^n)^n$ or a prefix of $(v^n u^n)^n$.

Claim 9.3 *([53]). Let L be a weak ω-regular language. Let $s \in \Sigma^*$ and $u, v \in \Sigma^+$ be such that $su^\omega \in L$ and $sv^\omega \notin L$. Let $t \in \{su, sv\}$ be a finite word such that $t \sim_L s$. Then there exists an ω-word $w \in \Delta^n_{u,v}$ such that $tw \in L \iff sw \notin L$.*

Therefore by asking membership queries for every $w \in \Delta^n_{u,v}$ we can find a distinguishing column title to add to the table.

Algorithm 3: $\mathbf{L}^{W\omega}$.

oracles : MQ, EQ
members: Observation table $\mathcal{T} = (S, E, M)$, Base set B
methods : *IsClosed*, *ExtractAut*, *AddRow*, *AddCol*, *MarkOrDetect*, *ResolveConflict*
$S = \langle\lambda\rangle$, $E = \langle\rangle$, $B = \langle\lambda\rangle$ and $M = \langle\rangle$.
repeat
  $(a_1, s_1) = IsClosed()$
  if $a_1 =$ *"no"* **then**
    $AddRow(s_1)$
  else
    $\mathcal{A}_1 = ExtractAut()$
    $(a_2, s_2, u_2, v_2, \mathcal{P}_2) = MarkOrDetect(\mathcal{A}_1)$
    if $a_2 =$ *"Conflict"* **then**
      $w_2 = ResolveConflict(s_2, u_2, v_2)$
      $AddCol(w_2)$
    else
      $(a_3, s_3) = \text{EQ}(\mathcal{P}_2)$
      if $a_3 =$ *"no"* **then**
        forall $v \in Suffixes(s_3)$ **do**
          $AddCol(v)$
until $a_3 =$ *"yes"*
return $\mathcal{P}_2$

The workings of $\mathbf{L}^{W\omega}$. With this we established the workings of the procedures required for the $\mathbf{L}^{W\omega}$ algorithm for learning a weak regular ω-language. The $\mathbf{L}^{W\omega}$ learning algorithm is summarized in Algorithm 3.

Theorem 9.4 *([53]). Given an unknown weak regular ω-language L with index n, the algorithm $\mathbf{L}^{W\omega}$ returns a minimal weak DPA for L in time polynomial in n, ℓ and k, where ℓ is the size of the longest counterexample given, and k is the size of the alphabet.*

An example of a run of $\mathbf{L}^{W\omega}$ is available in [53].

10. Query learning of the full class of regular ω-languages

When considering all regular ω-languages we have to confront the fact that the right congruence relation $\sim_L$ for an ω-language L is not informative enough. For example, consider the languages R and S recognized by the non-weak DPAs $\mathcal{R}$ and $\mathcal{S}$ of Fig. 3, respectively. We have that $|\Sigma^*/\sim_R| = 1$ though a minimal DPA for R requires at least 2 states and $|\Sigma^*/\sim_S| = 2$ though a minimal DPA for S requires 4 states. For this reason, the problem of learning the class of all ω-regular languages was considered open for many years [47].

Reduction to finite words. It turns out, that an unknown ω-regular language can be learned using MQ and EQ. This follows from a result of Calbrix, Nivat and Podelski [21] showing a reduction from an ω-regular language L over Σ^ω to a regular language $L_\$$ of words in $\Sigma^*\$\Sigma^+$ such that $u\$v \in L_\$$ iff $uv^\omega \in L$. Thus, one can use $\mathbf{L}^*$ and learn $L_\$$ rather than L. Building on this idea Farzan et al. [31] devised a learning algorithm in the context of assume-guarantee reasoning. The problem is that the reduction of L to $L_\$$ is not polynomial. Indeed, for a language that can be recognized by a Büchi automaton with n states, the size of a DFA for $L_\$$ is bounded by $2^n + 2^{2n^2+n}$ [21].

Other congruences. Since the right congruence is not enough to characterize ω-languages, perhaps we need another type of relation. In particular, we would like the relation to relate two words if they comprise the same part of loops. We can define for $x, y \in \Sigma^*$

$$x \overset{\circ}{\sim}_L y \quad \text{iff} \quad \forall u, v \in \Sigma^*.(u(xv)^\omega \in L \iff u(yv)^\omega \in L)$$

But this is not enough. Consider the language $L_1 = ab^\omega + cd^\omega$. The relation $\overset{\circ}{\sim}_{L_1}$ has 4 equivalence classes, one consisting of the empty word, one consisting of all words b^+, another one of all words d^+ and the last for the rest of the words. While it nicely captures that all sequences of b's are equivalent in the sense that a suffix extending one to a legitimate loop will also extend the other, it does not differentiate words a and c. Clearly any automaton for L_1 would need to distinguish prefixes beginning with a from prefixes beginning with c.

So one can consider a relation demanding both as follows:

$$x \ddot{\sim}_L y \quad \text{iff} \quad (x \sim_L y \quad \wedge \quad x \overset{\circ}{\sim}_L y)$$

The problem with this definition is that we don't always want to consider the two relations $\sim_L$ and $\overset{\circ}{\sim}_L$ simultaneously. For instance, consider the language $L_2 = b^*a(bbba)^\omega$. While we would like to distinguish b from bb when the loop is processed, we do not need to distinguish b from bb while the prefix is processed, and the relation $\ddot{\sim}_L$ will unnecessarily do so. Such relations and observations were studied in the literature [70,43,50,68,12] in a quest to come up with a Myhill–Nerode characterization for all regular ω-languages.

This line of works culminated in the work of Maler and Staiger [55] who defined *families of right congruence* (FORC) and a particular FORC, *the syntactic FORC*, for which they showed to have a Myhill–Nerode characterization in the sense that a language is ω-regular iff it is recognized by a finite FORC, and the syntactic FORC is the minimal FORC recognizing a language.

Based on these ideas Angluin and Fisman [10] defined a *family of DFAs (FDFA)* that relaxes some of the conditions in the definition of FORC, and a particular FDFA, *the recurrent FDFA*. They showed that the syntactic FORC may be exponentially more succinct than $L_\$$, and the recurrent FDFA may be quadratically smaller than the syntactic FORC.

Family of DFAs. In a family of DFAs (FDFA) there is a leading automaton $\mathcal{A}$ and for each state q of the leading automaton, there is a DFA $\mathcal{D}_q$. Since automata naturally map to right congruence, one can define an FDFA by defining a leading right congruence $\sim$, and for each equivalence class u of $\sim$, a right congruence $\approx_u$. Roughly speaking, the prefix part of a periodic word is processed by the leading automaton $\mathcal{A}$, and the periodic part is processed by one of the DFAs $\mathcal{D}_q$. The question which DFA should be used to process the period is determined by $\mathcal{A}$ on its processing of the prefix. When using an FDFA as an acceptor for ω-languages, we would like the FDFA to return the same result when processing distinct pairs (u, v) and (x, y) satisfying that $uv^\omega = xy^\omega$. We say that an FDFA $\mathcal{F}$ is **saturated** iff for all pairs (u, v), (x, y) such that $uv^\omega = xy^\omega$ either $\mathcal{F}$ accepts both (u, v) and (x, y) or it rejects both.

The saturation requirement can also be applied to $L_\$$, there we require the DFA to return the same result for $u\$v$ and $x\$y$ if $uv^\omega = xy^\omega$. It is this requirement that makes $L_\$$ so big with respect to an ω-automaton for the language. Consider for instance the DPA $\mathcal{U}$ in Fig. 4, recognizing the language $U = (aba + bab)^\omega$. For the DPA to accept $(ab)^\omega$ (as it should since $(ab)^\omega \in U$) it traverses the period $ababab$ of length 6. The DFA for $L_\$$ must accept $\lambda\$ab$. As shown in [10] insisting on finding every period can cause the DFA for $L_\$$ to be exponentially bigger than a respective parity automaton.

As another example consider the DPA $\mathcal{V}$ in Fig. 4 over alphabet $\Sigma = \{0, 1, 2\}$. Some of the (ultimately) periodic words that are accepted by $\mathcal{V}$ are easy to spot, for instance, $(11)^\omega$, $(22)^\omega$, $(100201)^\omega$, $(10020122)^\omega$. But some periodic words that are accepted by $\mathcal{V}$ are hard to spot, e.g. $(1012)^\omega$. The periods that are easy to spot are those that take state 0 back to itself.

The acceptance criteria for an FDFA was designed to avoid this problem. When an FDFA $\mathcal{F}$ receives a pair (u, v) if first normalizes it to another pair (x, y) so that $uv^\omega = xy^\omega$, but presumably, establishing that the automaton may loop on y after reading x is easier than establishing that the automaton may loop on v after reading u (or uv^i for some $i \geq 0$). Formally,

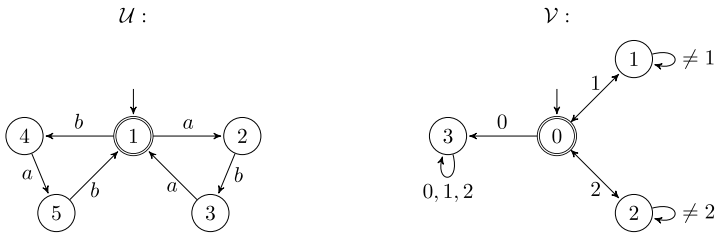


Fig. 4. Two DPAs.

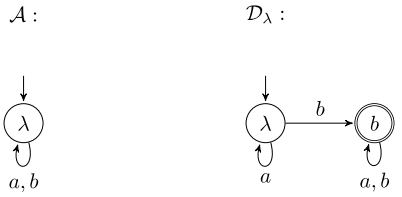


Fig. 5. The recurrent FDFA for $L_{\infty b}$.

the **normalization of** (u, v) **w.r.t. an automaton** $\mathcal{A}$ is the pair (x, y) such that $x = uv^i$, $y = v^j$ and $i \geq 0$, $j \geq 1$ are the smallest numbers for which $\mathcal{A}(uv^i) = \mathcal{A}(uv^{i+j})$. Since we consider complete deterministic automata, such a unique pair is guaranteed. We note that normalization is idempotent, that is, if the normalization of (u, v) with respect to $\mathcal{A}$ is (x, y) then (x, y) is the normalization of itself w.r.t. $\mathcal{A}$.

Definition 10.1 *([10]).* A **family of DFAs (FDFA)** $\mathcal{F} = (\mathcal{A}, \{\mathcal{D}_q\})$ over an alphabet Σ consists of a leading automaton $\mathcal{A} = (\Sigma, Q, q^0, \delta)$ and progress DFAs $\mathcal{D}_q = (\Sigma, S_q, s_q^0, \delta_q, F_q)$ for each $q \in Q$. The FDFA $\mathcal{F}$ **accepts** (u, v) if $\mathcal{D}_q$ accept y where (x, y) is the normalization of (u, v) w.r.t. $\mathcal{A}$ and $\mathcal{A}(x) = q$. We use $(\!|F|\!)$ for the set of pairs (u, v) accepted by $\mathcal{F}$. If $\mathcal{F}$ is saturated we use $[\![F]\!]$ for the ultimately periodic words uv^ω such that $(u, v) \in (\!|F|\!)$.

Definition 10.2 *([10]).* Let $x, y, u \in \Sigma^*$ and L be an ω-language. We use $x \approx_L^u y$ iff $\forall v \in \Sigma^*$ it holds that $(uxv \sim_L u$ and $u(xv)^\omega \in L)$ iff $(uyv \sim_L u$ and $u(yv)^\omega \in L)$. We refer to $\approx_L^u$ as the **recurrent congruence relation**. An FDFA $\mathcal{F} = (\mathcal{A}, \{\mathcal{D}_q\})$ is the **Recurrent FDFA** for L if $\mathcal{A}$ is the automaton for $\sim_L$ and for every state q of $\mathcal{A}$ we have that $\mathcal{D}_q$ is the DFA for $\approx_L^u$ for some $u \in \mathcal{D}_{q]}$.

Example 10.3. Consider the DPA $\mathcal{R}$ of Fig. 3 recognizing the language $L_{\infty b}$ of all words with infinitely many $b's$. The recurrent FDFA for $L_{\infty b}$ is given in Fig. 5. The leading automaton $\mathcal{A}$ has one state since $|\Sigma^*/\sim_{L_{\infty b}}| = 1$, the DFA $\mathcal{D}_\lambda$ for this state has two states since $|\Sigma^*/\approx_{L_{\infty b}}^{\lambda}| = 2$. Clearly it accepts all ultimately periodic words with at least one b in their periodic part.

Example 10.4. Consider the DPA $\mathcal{V}$ of Fig. 4 recognizing the language V. The recurrent FDFA for V is given in Fig. 6. The leading automaton is $\mathcal{A}$, and the progress DFAs corresponding to its 4 states are $\mathcal{D}_\lambda$, $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$. We can see, for instance, that $\mathcal{D}_1$ accepts periods starting with 1, ending with 1 where the number of 1's is even and where 0 never occurs immediately after an occurrence of an odd 1, which is exactly the requirement that $\mathcal{V}$ imposes on periods that loop on state 1 of $\mathcal{V}$.

Claim 10.5 *([10]). Let L be a regular ω-language and $\mathcal{F}_L = (\mathcal{A}, \{\mathcal{D}_q\})$ be its recurrent FDFA. Then $(\!|\mathcal{F}_L|\!)$ is saturated and $[\![\mathcal{F}_L]\!]$ is the set of all ultimately periodic words of L.*

Finiteness and succinctness of FDFAs. Finiteness of the number of equivalences of $\sim_L$ follows from the fact that $\sim_{\mathcal{D}}$ refines $\sim_L$ for any ω-automaton $\mathcal{D}$ recognizing L. We claim that for any $u \in \Sigma^*$ and any ω-regular language L, the number of equivalence classes of $\approx_L^u$ is finite.

Claim 10.6. *For any regular ω-language L over Σ, and any word $u \in \Sigma^*$, the relations $\sim_L$ and $\approx_L^u$ have finitely many equivalence classes.*

In particular, if $\mathcal{P}$ is a DPA with n states accepting L, then the index of $\approx_L^u$ is bounded by $n2^n$. Indeed, let $m = n2^n$ and assume that $\sim_L$ has m' equivalence classes for some $m' > m$. Let $x_1, x_2, \ldots, x_{m'}$ be their representative words. Consider the word x_{m+1}. For every $i \in [1..m]$ there exists a word v_i distinguishing x_i from x_{m+1}. Thus either $\mathcal{P}$ reaches different states

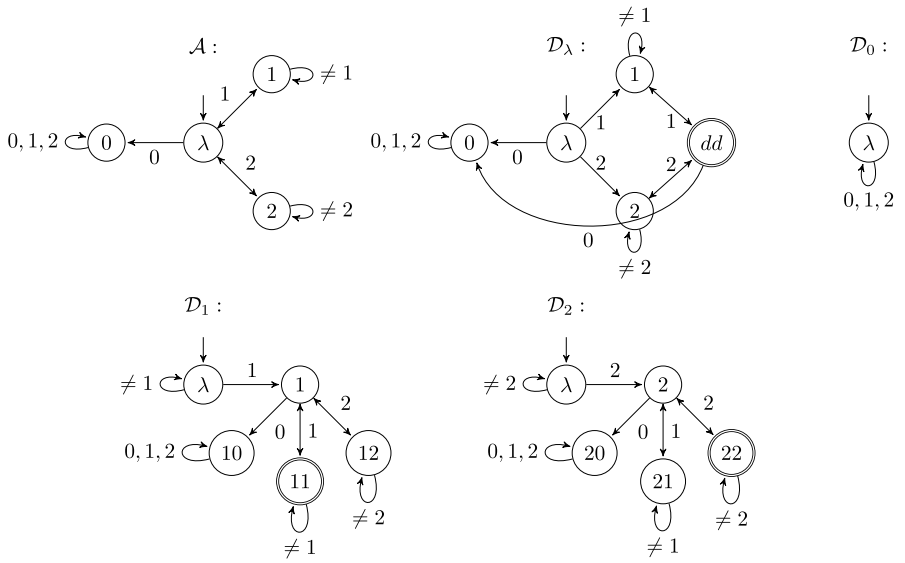


Fig. 6. The recurrent FDFA for $[\![\mathcal{V}]\!]$.

when reading ux_iv_i and $ux_{m+1}v_i$ or, it visits different states when looping on x_iv_i and on $x_{m+1}v_i$ after reading u. Let q be the states that $\mathcal{P}$ reaches after reading ux_{m+1} and let S be the set of states that it visits when reading x_{m+1} from q. Since there are n options for the state q and 2^n for the set of states S there must be $j \in [1..n2^n]$ such that $\mathcal{P}$ reaches q after reading ux_j and S is the set of states that it visits when reading x_j from q. Thus the set of states S' that $\mathcal{P}$ visits when reading x_jv_j from q is the same as the set of states that it visits when reading $x_{m+1}v_j$ from q, contradicting that v_j distinguishes x_j from x_{m+1}.

The motivation for using FDFAs rather than $L_\$$ for learning, was their potential succinctness. Indeed they can be exponentially more succinct than $L_\$$.

Claim 10.7 *([10]). There exists a family of languages $\{L_n\}$ such that the recurrent FDFA for L_n is of size $O(n^2)$ whereas the DFA for $L_\$$ requires at least $n!$ states.*

The family of languages $\{L_n\}$ used in the proof of this claim is a generalization of the DPA $\mathcal{V}$ of Fig. 4 which recognizes L_2.

Angluin, Boker and Fisman [8] have compared FDFAs in relation to traditional ω-automata, in various aspects including succinctness. They show that FDFAs are more succinct than DPAs in the sense that translating a DPA to an FDFA can always be done with only a polynomial increase, yet the other direction involves an inevitable exponential blowup in the worst case.

Claim 10.8 *([8]). Let $\mathcal{P}$ be a DPA with n states and k colors. There exists a saturated FDFA $\mathcal{F}$ with at most $O(kn)$ states recognizing the same language.*

Claim 10.9 *([8]). There exists a family of languages $\{F_n\}_{n\geq 1}$ over the alphabet $\{1, 2, \ldots, n\}$ such that for every $n \geq 1$, there exists a saturated FDFA of size at most $O(n^2)$ that characterizes F_n, while a DPA for F_n must be of size at least 2^{n-1}.*

10.1. The $\mathbf{L}^\omega$ algorithm

The FDFA model was introduced as an acceptor for regular ω-languages, since it directly corresponds to a set of right congruences, and most learning algorithms build on the existence of such relations. This representation alone does not reduce the problem to using $\mathbf{L}^*$ to learn the different relations. This is since we are not given MQ and EQ oracles for the languages accepted by the progress DFAs (but rather we are given MQ and EQ oracles for the unknown language).

A recurrent observation table. The leading congruence $\sim_L$ can be learned in the same manner as in $\mathbf{L}^{\mathrm{W}\omega}$. We need to establish how to learn the recurrent congruences $\approx_L^u$. Recall the definition of the recurrent congruences:

$$x \approx_L^u y \quad \text{iff} \quad \forall v \in \Sigma^* \\ (uxv \sim_L u \text{ and } u(xv)^\omega \in L) \quad \text{iff} \quad (uyv \sim_L u \text{ and } u(yv)^\omega \in L).$$

To establish this relation we will use a *recurrent observation table*. The **recurrent observation table for word u w.r.t. regular ω-language L and automaton $\mathcal{A}$** is a tuple (S, E, M) where S is a set of words in Σ^* corresponding to candidate states, E is a set of words in Σ^* corresponding to period-experiments, and for $x \in S$ and $v \in E$ we have that $M_{x,v}$ is a pair (m, ℓ) such that m is 1 if $u(xv)^\omega \in L$ and 0 otherwise, and ℓ is 1 if $\mathcal{A}(uxv) = \mathcal{A}(u)$ and 0 otherwise. Two rows x and y of a recurrent observation table are considered **distinct** if there exists $e \in E$ such that $M_{x,e} = (1,1)$ and $M_{y,e} \neq (1,1)$ or vice versa. Thus, rows x and y in a recurrent observation table for u are considered distinct if there is an evidence word e showing that $x \not\approx_L^u y$. From a recurrent observation table $\mathcal{T} = (S, E, M)$ that is closed with respect to $B \subseteq S$ we can extract an automaton in the usual manner. To make it an acceptor we need to decide which states (rows) should be declared accepting. We declare a state x accepting iff $M_{x\lambda} = (1,1)$, that is, if $\mathcal{A}(ux) = \mathcal{A}(u)$ and $ux^\omega \in L$.

The workings of $\mathbf{L}^\omega$. The $\mathbf{L}^\omega$ algorithm maintains an ω-observation table $\mathcal{T} = (S, E, M, B)$ to discover the leading automaton.[13] It tries to close the leading table (using *CloseLeadingTable*$(\mathcal{T})$), and when the table is closed it extract the current hypothesis for the leading automaton $\mathcal{A}$. For each state u of $\mathcal{A}$ it maintains a recurrent observation table $\mathcal{T}^u = (S^u, E^u, M^u, B^u)$ to discover the DFA for the recurrent relation $\approx_L^u$. The first component of a recurrent table entry is computed using membership queries, and the second using the current leading automaton $\mathcal{A}$. It tries to close the recurrent tables for all states u in the base B of the leading table $\mathcal{T}$ (using *CloseRecurrentTable*$(u, \mathcal{T}_u)$). When all tables are complete, it extracts the DFAs $\mathcal{D}_u$ for the recurrent table of u (using *ExtractRecurrentDFA*$(u, \mathcal{T}_u)$) for all states $u \in B$, and issue an equivalence query for $\mathcal{H} = (\mathcal{A}, \{\mathcal{D}_u\})$. If the result is true, it returns $\mathcal{H}$. Otherwise, it needs to find a new state either in the leading automaton or in one of the progress automata.

The algorithm is summarized in Algorithm 4:

Algorithm 4: The Learner $\mathbf{L}^\omega$.

```
Initialize leading table T = (S, E, T, B) with S = B = {λ}, and E = {(λ, σ) | σ ∈ Σ}.
CloseLeadingTable(T) and let A = ExtractLeadingAut(T).
forall u ∈ B do
    Initialize the table for u, T_u = (S_u, E_u, T_u, B_u), with S_u = B_u = E_u = {λ}.
    CloseRecurrentTable(u, A, T_u) and let D_u = ExtractRecurrentDFA(T_u).
repeat
    Let (a, u, v) be the teacher's response on equivalence query H = (A, {D_u}).
    Let (x, y) be the normalized factorization of (u, v) with respect to M.
    Let x̃ be A(x).
    if MQ(x, y) ≠ MQ(x̃, y) then
        E = E ∪ FindDistinguishingExperiment(x, y).
        CloseLeadingTable(T) and let A = ExtractLeadingAut(T).
        forall u ∈ B do
            CloseRecurrentTable(u, T_u) and let D_u = ExtractRecurrentDFA(T_u).
    else
        E_x̃ = E_x̃ ∪ FindDistinguishingExperiment(x̃, y).
        CloseRecurrentTable(x̃, A, T_x̃) and let D_x̃ = ExtractRecurrentDFA(T_x̃).
until a = "yes"
return H
```

The algorithm starts by initializing and closing the leading table, and the respective recurrent tables and asking an equivalence query about the resulting hypothesis. The algorithm then repeats the following loop until the equivalence query returns "yes".

If the equivalence query returns a counterexample (u, v) the learner first obtains the normalized factorization (x, y) of (u, v) with respect to its current leading automaton $\mathcal{A}$. It then checks whether membership queries for (x, y) and $(\tilde{x}, y)$, where $\tilde{x}$ is the state $\mathcal{A}$ arrives at after reading x, return different results. If so, it calls the procedure *FindDistinguishingExperiment* to find a distinguishing experiment to add to the leading table. It then closes the leading table and all the recurrent tables and obtains a new hypothesis $\mathcal{H}$.

If membership queries for (x, y) and $(\tilde{x}, y)$ return the same results, it calls the procedure *FindDistinguishingExperiment* to find a distinguishing experiment in the recurrent DFA for $\tilde{x}$. It then closes this table and obtains a new hypothesis.

Convergence. To see that the algorithm converges we will show that processing a counterexample always reveals a new state, either in the leading automaton or in one of the recurrent DFAs. This argument, together with finiteness of $\sim_L$ and finiteness of $\approx_L^u$ for any $u \in \Sigma^*$ (as per Claim 10.6) guarantees that the algorithm converges (even when exploring recurrent tables before the leading automaton is completely discovered).

Suppose the returned counterexample is (u, v), and its normalized factorization with respect to the current leading automaton $\mathcal{A}$ is (x, y). The learner then checks whether membership queries for (x, y) and $(\tilde{x}, y)$ return different results

[13] We now include the candidate base set B in the tuple of the observation table $\mathcal{T}$.

where $\tilde{x} = \mathcal{A}(x)$. Let $|x| = n$ and for $i \in [1..n]$ let $s_i = \mathcal{A}(x[1..i])$ be the state that the leading automaton reaches after reading the first i symbols of x. Then $\tilde{x} = s_n$, and we know that a sequence of membership queries with (x, y), $(s_1 x[2..n], y)$, $(s_2 x[3..n], y)$, and so on, up to $(s_n, y) = (\tilde{x}, y)$ has different answers for the first and last queries. Thus, a sequential search of this sequence suffices to find a consecutive pair, say $(s_{i-1}x[i..n], y)$ and $(s_i x[i+1..n], y)$, with different answers to membership queries. This shows that the experiment $(x[i+1..n], y)$ distinguishes $s_{i-1}x[i]$ from s_i in the leading table, though $\delta(s_{i-1}, x[i]) = s_i$, so that adding it, there will be at least one more state in the leading automaton.

If membership queries for (x, y) and $(\tilde{x}, y)$ return the same answers, we look for an experiment that will distinguish a new state in the recurrent table of $\tilde{x}$. Let $\tilde{y} = \mathcal{A}_{\tilde{x}}(y)$. Consider the sequence of experiments (λ, y), $(s_1, y[2..n])$, $(s_2, y[3..n])$ up to (s_n, λ) and the respective entries (m_0, ℓ_0), $(m_1, \ell_1), \ldots, (m_n, \ell_n)$ in the table $T_{\tilde{x}}$. We know that out of (m_0, ℓ_0) and (m_n, ℓ_n) one is $(1, 1)$ and the other one is not. Therefore for some i we should have that (m_i, ℓ_i) is $(1, 1)$ and (m_{i-1}, ℓ_{i-1}) is not, or vice versa. Thus, the experiment $y[i + 1..n]$ distinguishes $s_{i-1}y[i]$ from s_i.

Theorem 10.10 *([10]). Given an unknown regular ω-language L, the algorithm $\mathbf{L}^\omega$ always halts and returns the recurrent FDFA for L.*

An example of a run of $\mathbf{L}^\omega$ is available in [11].

Unfortunately, the worst case time and query complexity of $\mathbf{L}^\omega$ is not polynomial in the outputed FDFA. Intuitively, the reason is that both the leading table and the recurrent tables are worked out simultaneously. It can be that before the algorithm has established the correct leading automaton, trying to establish one of the recurrent DFAs, the algorithm has to find "hard periods", because on the current leading automaton, the presumed period v loops, but v won't loop (at least not for the respective prefix u) on the correct leading automaton.

10.2. Implementations and experimental results

Experiments in learning randomly generated Muller automata were conducted in [11]. Out of 600 examples, in over 99% of the cases the resulting FDFA was smaller than the DFA for $L_\$$, in over 85% of the cases the resulting FDFA was more than twice smaller than the DFA for the corresponding $L_\$$, and in some cases it was more than 64 times smaller.

An algorithm based on $\mathbf{L}^\omega$ outputting Büchi automata, and using discrimination trees instead of observation tables was proposed in [49]. Implementation of this algorithm as well as table and discrimination-tree based implementation of the algorithm in [31] and the other variants of $\mathbf{L}^\omega$ proposed in [10] are publicly available in the ROLL library [49]. This work also reports on an empirical study comparing all these algorithms.

11. Summary

We have seen that the residuality property, the relation between the states of the automaton to the right-congruence relation of the language (or to a set of right congruence relations) is the foundation for many learning algorithms. For regular languages over finite words we have seen the algorithms $\mathbf{L}^*$, $\mathbf{NL}^*$, and $\mathbf{AL}^*$, that output a DFA, NFA, and AFA, resp. using membership and equivalence queries. These algorithms are polynomial w.r.t. the row index (the size of the right congruence of the language), the column index (the size of the right congruence of the reversed language), the size of the alphabet, and the length of the maximal counterexample.

For weak regular ω-languages we have seen the learning algorithm $\mathbf{L}^{\mathrm{W}\omega}$ that outputs weak parity automata. This algorithm is polynomial w.r.t. size of the right congruence, the size of the alphabet, and the length of the maximal counterexample. For regular ω-languages we have seen the learning algorithm $\mathbf{L}^\omega$ that outputs families of DFAs. The algorithm clearly cannot be polynomial w.r.t. the size of the right congruence of a regular ω-language since the right congruence is not informative enough. We expected that it would be polynomial w.r.t. the size of the recurrent FDFA, because it is informative enough. However, it is not, since the leading automaton and progress DFAs are learned simultaneously, and before the correct leading automaton is revealed, an unnecessarily big progress DFA may be constructed. Any result finding a sub-class of the regular ω-languages which strictly subsumes the weak languages, and can be polynomially learned, is very much of interest.

Recent work [7] looked at the problem of learning regular ω-tree languages. Given an ω-word language L, the notation $\mathit{Trees}_d(L)$ is used for the set of all d-ary ω-trees t all of whose paths are in L. In some literature $\mathit{Trees}_d(L)$ is referred to as the *derived* language of L. For a class $\mathbb{C}$ of ω-word, the notation $\mathit{Trees}_d(\mathbb{C})$, is used for the class of all d-ary ω-trees languages $\mathit{Trees}_d(L)$ for $L \in \mathbb{C}$. A natural question is whether there exists a polynomial time reduction of learning derived ω-tree languages $\mathit{Trees}_d(\mathbb{C})$ to learning the underlying class of ω-word languages $\mathbb{C}$. A positive answer is provided in [7] for any class of ω-word languages in $\mathbb{DP}^+_2$, the class of languages accepted by a DPA using the colors $\{1, 2\}$ (which is equivalent to the class of languages accepted by a deterministic Büchi automaton). Since $\mathbb{DP}^+_2 \supsetneq \mathbb{DW}$ it follows that the class $\mathit{Trees}_d(\mathbb{DW})$ can be polynomially learned using membership and equivalence queries. Furthermore, if a polynomial time algorithm for $\mathbb{DP}^+_2$ will be found, it will automatically lift to $\mathit{Trees}_d(\mathbb{DP}^+_2)$.

To conclude, while for the regular language case the problem of polynomial learnability using membership and equivalence queries has been satisfactorily resolved, for the class of ω-regular languages (and tree languages) more research is required.

Acknowledgements

I would like to thank Dana Angluin for fascinating discussions on the underlying topics, and the reviewers for their thorough review and insightful questions, that greatly increased the quality of this paper. This research was supported by the United States–Israel Binational Science Foundation (BSF) grant 2016239.

References

[1] F. Aarts, F. Vaandrager, Learning I/O automata, in: CONCUR 2010 – Concurrency Theory, 21th International Conference CONCUR 2010, Paris, France, August 31–September 3, 2010, Proceedings, 2010, pp. 71–85.
[2] R. Alur, P. Černý, P. Madhusudan, W. Nam, Synthesis of interface specifications for Java classes, in: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '05, ACM, New York, NY, USA, ISBN 1-58113-830-X, 2005, pp. 98–109.
[3] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (2) (1987) 87–106.
[4] D. Angluin, Queries and concept learning, Mach. Learn. 2 (4) (1987) 319–342.
[5] D. Angluin, Negative results for equivalence queries, Mach. Learn. 5 (1990) 121–150.
[6] D. Angluin, The Class DBW ∩ DCW of ω-Languages Is Identifiable in the Limit from Positive Data and Membership Queries with Polynomial Time and Data, Tech. Rep. YALEU/DCS/TR-1528, Department of Computer Science, Yale University, 2016.
[7] D. Angluin, T. Antonopoulos, D. Fisman, Query learning of derived omega-tree languages in polynomial time, in: 26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20–24 2017, Stockholm, Sweden, 2017, 10, 21 pp.
[8] D. Angluin, U. Boker, 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, 2016, 11, 17 pp.
[9] D. Angluin, S. Eisenstat, D. Fisman, Learning regular languages via alternating automata, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, 2015, pp. 3308–3314.
[10] D. Angluin, D. Fisman, Learning regular omega languages, in: Algorithmic Learning Theory – 25th International Conference, ALT 2014, Bled, Slovenia, October 8–10, 2014, Proceedings, 2014, pp. 125–139.
[11] D. Angluin, D. Fisman, Learning regular omega languages, Theor. Comput. Sci. 650 (2016) 57–72.
[12] A. Arnold, A syntactic congruence for rational omega-languages, Theor. Comput. Sci. 39 (1985) 333–335.
[13] B. Balle, M. Mohri, Learning weighted automata, in: Proc. 6th Int. Conf. on Algebraic Informatics, 2015, pp. 1–21.
[14] F. Bergadano, S. Varricchio, Learning behaviors of automata from multiplicity and equivalence queries, SIAM J. Comput. 25 (6) (1996) 1268–1280.
[15] S. Berndt, M. Liskiewicz, M. Lutter, R. Reischuk, Learning residual alternating automata, in: The Thirty-First AAAI Conference on Artificial Intelligence, AAAI-17, 2017.
[16] A.W. Biermann, J.A. Feldman, On the synthesis of finite-state machines from samples of their behavior, IEEE Trans. Comput. (ISSN 0018-9340) 21 (6) (1972) 592–597.
[17] M. Bojańczyk, Transducers with origin information, in: Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8–11, 2014, Proceedings, 2014, pp. 26–37.
[18] B. Bollig, P. Habermehl, C. Kern, M. Leucker, Angluin-style learning of NFA, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11–17, 2009, 2009, pp. 1004–1009.
[19] B. Bollig, J. Katoen, C. Kern, M. Leucker, D. Neider, D.R. Piegdon, libalf: the automata learning framework, in: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15–19, 2010, Proceedings, 2010, pp. 360–364.
[20] M. Botincan, D. Babic, Sigma*: symbolic learning of input–output specifications, in: Proc. 40th ACM Syp. on Principles of Programming Language, 2013, pp. 443–456.
[21] H. Calbrix, M. Nivat, A. Podelski, Ultimately periodic words of rational ω-languages, in: Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7–10, 1993, Proceedings, 1993, pp. 554–566.
[22] S. Cassel, F. Howar, B. Jonsson, B. Steffen, Active learning for extended finite state machines, Form. Asp. Comput. 28 (2) (2016) 233–263.
[23] A.K. Chandra, L.J. Stockmeyer, Alternation, in: 17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 1976, pp. 98–108.
[24] M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, 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, 2015, pp. 114–130.
[25] Y. Chen, E.M. Clarke, A. Farzan, M. Tsai, Y. Tsay, B. Wang, Automated assume-guarantee reasoning through implicit learning, in: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15–19, 2010, Proceedings, 2010, pp. 511–526.
[26] J.-E.P.D. Perrin, Infinite Words: Automata, Semigroups, Logic and Games, Springer, 2004.
[27] C. de la Higuera, J.-C. Janodet, Inference of [omega]-languages from prefixes, Theor. Comput. Sci. 313 (2) (2004) 295–312.
[28] F. Denis, A. Lemay, A. Terlutte, Residual finite state automata, in: 18th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2001, Dresden, Germany, February 15–17, 2001, Proceedings, 2001, pp. 144–157.
[29] M. Domaratzki, D. Kisman, J. Shallit, On the number of distinct languages accepted by finite automata with n states, in: Third International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, DCAGRS 2001, Vienna, Austria, July 20–22, 2001, Preproceedings, 2001, pp. 67–78.
[30] D. Drews, L. D'Antoni, Learning symbolic automata, in: Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings, Part I, 2017, pp. 173–189.
[31] A. Farzan, Y.-F. Chen, E.M. Clarke, Y.-K. Tsay, B.-Y. Wang, Extending automated compositional verification to the full class of omega-regular languages, in: C. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 4963, Springer, Berlin, Heidelberg, 2008, pp. 2–17.
[32] L. Feng, M.Z. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: Fundamental Approaches to Software Engineering – 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26–April 3, 2011, Proceedings, 2011, pp. 2–17.
[33] M. Fliess, Matrices de Hankel, J. Math. Pures Appl. (1974) 197–224.
[34] E.M. Gold, Language identification in the limit, Inf. Control 10 (5) (1967) 447–474.
[35] E.M. Gold, Complexity of automaton identification from given data, Inf. Control 37 (3) (1978) 302–320.
[36] O. Grinchtein, M. Leucker, N. Piterman, Inferring network invariants automatically, in: Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings, 2006, pp. 483–497.
[37] M. Heule, S. Verwer, Exact DFA identification using SAT solvers, in: Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13–16, 2010, Proceedings, 2010, pp. 66–79.

[38] F. Howar, B. Steffen, B. Jonsson, S. Cassel, Inferring canonical register automata, in: Verification, Model Checking, and Abstract Interpretation – 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22–24, 2012, Proceedings, 2012, pp. 251–266.
[39] O.H. Ibarra, T. Jiang, Learning regular languages from counterexamples, in: Proceedings of the First Annual Workshop on Computational Learning Theory, COLT '88, Cambridge, MA, USA, August 3–5, 1988, 1988, pp. 371–385.
[40] M. Isberner, F. Howar, B. Steffen, The TTT algorithm: a redundancy-free approach to active automata learning, in: Runtime Verification – 5th International Conference, RV 2014, Toronto, ON, Canada, September 22–25, 2014, Proceedings, 2014, pp. 307–322.
[41] M. Isberner, F. Howar, B. Steffen, The open-source LearnLib – a framework for active automata learning, in: Computer Aided Verification – 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18–24, 2015, Proceedings, Part I, 2015, pp. 487–495.
[42] M. Jayasrirani, M.H. Begam, D.G. Thomas, J.D. Emerald, Learning of Bi-ω languages from factors, in: Proceedings of ICGI 2012, in: JMLR Workshop and Conference Proceedings, vol. 21, 2012, pp. 139–144.
[43] H. Jürgensen, G. Thierrin, On ω-languages whose syntactic monoid is trivial, Int. J. Parallel Program. 12 (5) (1983) 359–365.
[44] M.J. Kearns, U.V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, USA, ISBN 0-262-11193-4, 1994.
[45] D. Kozen, On parallelism in Turing machines, in: 17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25–27 October 1976, 1976, pp. 89–97.
[46] E.L. Leiss, Succint representation of regular languages by Boolean automata, Theor. Comput. Sci. 13 (1981) 323–330.
[47] M. Leucker, Learning meets verification, in: Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7–10, 2006, Revised Lectures, 2006, pp. 127–151.
[48] M. Leucker, D. Neider, Learning minimal deterministic automata from inexperienced teachers, in: Leveraging Applications of Formal Methods, Verification and Validation, Technologies for Mastering Change – 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15–18, 2012, Proceedings, Part I, 2012, pp. 524–538.
[49] Y. Li, Y.-F. Chen, L. Zhang, D. Liu, A Novel Learning Algorithm for Büchi Automata based on Family of DFAs and Classification Trees, 2017.
[50] R. Lindner, L. Staiger, Eine Bemerkung über nichtkonstantenfreie sequentielle Operatoren, Elektron. Inf.verarb. Kybern. 10 (4) (1974) 195–202.
[51] C. Löding, W. Thomas, Alternating automata and logics over infinite words, in: Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference, IFIP TCS 2000, Sendai, Japan, August 17–19, 2000, Proceedings, 2000, pp. 521–535.
[52] O. Maler, I. Mens, Learning regular languages over large alphabets without a helpful teacher, in: Proc. 22nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, 2016.
[53] O. Maler, A. Pnueli, On the learnability of infinitary regular sets, in: Proceedings of the Fourth Annual Workshop on Computational Learning Theory, COLT 1991, Santa Cruz, California, USA, August 5–7, 1991, 1991, pp. 128–136.
[54] O. Maler, A. Pnueli, On the learnability of infinitary regular sets, Inf. Comput. 118 (2) (1995) 316–326.
[55] O. Maler, L. Staiger, On syntactic congruences for omega-languages, Theor. Comput. Sci. 183 (1) (1997) 93–112.
[56] R. McNaughton, Testing and generating infinite sequences by a finite automaton, Inf. Control 9 (5) (1966) 521–530.
[57] A.R. Meyer, M.J. Fischer, Economy of description by automata, grammars, and formal systems, in: 12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13–15, 1971, 1971, pp. 188–191.
[58] J. Myhill, Finite Automata and the Representation of Events, 1957.
[59] D. Neider, N. Jansen, Regular model checking using solver technologies and automata learning, in: NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14–16, 2013, Proceedings, 2013, pp. 16–31.
[60] A. Nerode, Linear automaton transformations, Proc. Am. Math. Soc. 9 (4) (1958) 541–544.
[61] O. Niese, An Integrated Approach to Testing Complex Systems, PhD Thesis, University of Dortmund, 2003.
[62] A.L. Oliveira, J.P.M. Silva, Efficient algorithms for the inference of minimum size DFAs, Mach. Learn. 44 (1/2) (2001) 93–119.
[63] J. Oncina, P. Garcia, Identifying regular languages in polynomial time, in: Advances in Structural and Syntactic Pattern Recognition, in: Series in Machine Perception and Artificial Intelligence, vol. 5, World Scientific, 1992, pp. 99–108.
[64] C.S. Pasareanu, D. Giannakopoulou, M.G. Bobaru, J.M. Cobleigh, H. Barringer, Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning, Form. Methods Syst. Des. 32 (3) (2008) 175–205.
[65] D.A. Peled, M.Y. Vardi, M. Yannakakis, Black box checking, J. Autom. Lang. Comb. 7 (2) (2002) 225–246.
[66] R.L. Rivest, R.E. Schapire, Inference of finite automata using homing sequences, Inf. Comput. (ISSN 0890-5401) 103 (2) (1993) 299–347.
[67] A. Saoudi, T. Yokomori, Learning local and recognizable omega-languages and monadic logic programs, in: EUROCOLT, in: LNCS, vol. 1121, Springer-Verlag, 1993, pp. 50–59.
[68] L. Staiger, Finite-state omega-languages, J. Comput. Syst. Sci. 27 (3) (1983) 434–448.
[69] L.J. Stockmeyer, The Set Basis Problem Is NP-Complete, Technical Report RC-5431, IBM, 1975.
[70] B. Trakhtenbrot, Finite automata and monadic second order logic, Sib. Math. J. (1962) 103–131.
[71] F. Vaandrager, Model learning, Commun. ACM 60 (2) (2017) 86–95.
[72] G. van Heerdt, M. Sammartino, A. Silva, CALF: categorical automata learning framework, in: 26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20–24, 2017, Stockholm, Sweden, 2017, 29, 24 pp.
[73] A. Vardhan, K. Sen, M. Viswanathan, 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, 2005, pp. 45–60.
[74] A. Vardhan, M. Viswanathan, LEVER: a tool for learning based verification, in: Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings, 2006, pp. 471–474.
[75] K.W. Wagner, A hierarchy of regular sequence sets, in: 4th Symposium on Mathematical Foundations of Computer, MFCS, 1975, pp. 445–449.