Learning Regular Languages via Alternating Automata

CONFERENCE PAPER · JULY 2015

READS

7

3 AUTHORS, INCLUDING:

Dana Fisman

University of Pennsylvania

42 PUBLICATIONS **540** CITATIONS

SEE PROFILE

Learning Regular Languages via Alternating Automata*

Dana Angluin
Yale University

Sarah Eisenstat
Massachusetts Institute of Technology

Dana Fisman
University of Pennsylvania

Abstract

Nearly all algorithms for learning an unknown regular language, in particular the popular L^* algorithm, yield deterministic finite automata. It was recently shown that the ideas of L^* can be extended to yield non-deterministic automata, and that the respective learning algorithm, NL^*, outperforms L^* on randomly generated regular expressions.

We conjectured that this is due to the existential nature of regular expressions, and NL^* might not outperform L^* on languages with a universal nature. In this paper we introduce UL^* — a learning algorithm for *universal automata* (the dual of non-deterministic automata); and AL^* — a learning algorithm for *alternating automata* (which generalize both universal and non-deterministic automata). Our empirical results illustrate the advantages and trade-offs among L^*, NL^*, UL^* and AL^*.

1 Introduction

Regular languages are an important class of languages, in that they are simple enough to model many systems/phenomena in real life and enjoy a large variety of theoretical properties, enabling efficient algorithms for many questions involving regular languages. Regular languages can be recognized by many different formalisms, for instance, regular expressions, finite automata, and monadic second order logic of one successor. There are several classes of finite automata, differing in the "branching type": deterministic finite automata (DFA), non-deterministic finite automata (NFA), universal finite automata (UFA) and alternating finite automata (AFA); and all are equally expressive, and as expressive as the class of regular languages.

A deterministic automaton $\mathcal{D}$ processes a given word $w = \sigma_1\sigma_2\ldots\sigma_\ell$ by transitioning from state to state, according to the transition relation. The DFA accepts w from a given state q, if the state q' that the automaton arrives at after reading σ_1 accepts the suffix $v = \sigma_2\ldots\sigma_\ell$. A non-deterministic automaton $\mathcal{N}$, when in state q and reading letter σ may transition to one of several states (as determined by its transition relation). The NFA accepts w, from state q if one of the states $q_1, \ldots, q_k$ it may arrive at after reading σ_1 accepts the suffix v. A dual notion is that of *universal automata*. Like an NFA the UFA's trasition relation maps the current state q and the next letter to read σ to a set of states, but the interpretation now is that the UFA accepts w from state q if all of the states $q_1, \ldots, q_k$ it arrives at after reading σ_1 accept the suffix v. *Alternating automata* (AFA) generalize both NFAs and UFAs, and can delegate the role of checking the next suffix to several states, combining their result using conjunctions and disjunctions. For instance, a transition from q upon reading σ to $(q_1 \wedge q_4) \vee q_3$ stipulates that the word σv is accepted from q if either v is accepted from q_3 or v is accepted from both q_1 and q_4.

While all these formalisms are equivalent in terms of expressive power, they differ in other qualities. For instance, an NFA may be exponentially smaller than the minimal DFA, and an AFA may be doubly-exponentially smaller than the minimal DFA. Nearly all algorithms for learning regular languages use the class of DFAs as the concept class. This is because the class of DFAs has several properties making it favorable for learning algorithms. In particular, for every regular language L there exists a unique minimal DFA $\mathcal{M}$ recognizing it, and the states of this DFA have the following property, termed *residuality*: Every state q can be associated with a word w_q such that the language accepted from that state, $[\![\mathcal{M}_q]\!]$, is exactly the set of suffixes of w_q in L (i.e. the set of words v such that $w_q v$ is in L). The Myhill-Nerode theorem states that for every regular language there exists a number n such that, any DFA with fewer than n states cannot correctly recognize L, and any DFA $\mathcal{D}$ with more than n states that correctly recognizes L has two states q_1 and q_2 for which $[\![\mathcal{D}_{q_1}]\!] = [\![\mathcal{D}_{q_2}]\!]$ and thus one of the states is redundant.

The residuality property is essential for many learning algorithms. In particular, residuality is essential for the popular L^* algorithm [Angluin, 1987], that learns a regular language from equivalence and membership queries. L^* makes use of the notion of an observation table. An observation table is a tuple (S, E, M) where S is a prefix-closed set of strings, E is a set of experiments trying to differentiate the S strings, and $M : S \times E \to \{0, 1\}$ is a matrix with 1 in the entry for (s_i, e_j) iff $s_i e_j \in L$. If two rows s_1 and s_2 differ in the entry with respect to a certain experiment e, then s_1 and s_2 cannot be representatives for the same state, since there exists

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

a suffix e which will be accepted by a state corresponding to one but not the other. If two rows agree on all experiments observed so far, there is currently no reason to think they do not correspond to the same automaton state.

The L^* algorithm builds a DFA from a given observation table, by associating a state with every distinct row in the table, and determining the transition on letter σ from a state corresponding to row s to go to the state that corresponds to the row $s\sigma$. The language accepted from state corresponding to s, is therefore indeed the residual of s with respect to L — it consists of all the suffixes e of s for which se is in L.

Non-deterministic automata do not, in general, have the residuality property. Suppose an NFA $\mathcal{N}$ recognizing L reads a word w that is in L. There is at least one accepting run of $\mathcal{N}$ on w, but there may be several non-accepting runs of $\mathcal{N}$ on w. Suppose $w = u\sigma v$ (where u and v are words, and σ is a letter) and when reading u, $\mathcal{N}$ had no non-deterministic choices to make, and it got to state q. A choice from state q on σ to state q' is "bad" for w, if the suffix v is not accepted from q'. While this is perfectly fine, and $\mathcal{N}$ still accepts w and correctly recognizes L, the state q' does not correspond to a residual language.

[Denis *et al.*, 2001a] have introduced the brilliant notion of *non-deterministic residual finite state automata* (NRFA) — a special type of NFA, where each state does correspond to a residual language, and intuitively, in the above sense, every choice can be "salvaged".[1] As in the deterministic case, each state q of an NRFA $\mathcal{N}$ recognizing a language L can be associated with a characteristic word w_q, such that the words accepted from this state onward, are exactly all the allowed suffixes of w_q in L. However, if there are n residual languages for L by the Myhil-Nerode theorem, an NRFA for L may have fewer than n states. If a residual language L_i of L does not correspond to any state of the NRFA then it corresponds to a union of some states of the NRFA.

The *residuality* property has been shown to be useful for various learning algorithms [Denis *et al.*, 2001b; Esposito *et al.*, 2002; Bollig *et al.*, 2009; Kasprzik, 2010; 2011]. In particular, [Bollig *et al.*, 2009] showed that the ideas of L^* can be generalized to obtain a minimal NRFA rather than the minimal DFA. They name their algorithm NL^*. Since the minimal NRFA is at most as large as the minimal DFA and may be exponentially smaller, this approach is very appealing. They further show, that the worst case complexity of NL^* in terms of membership queries and equivalence queries, is slightly worse than that of L^*, which makes sense given the obtained succinctness. Nonetheless, they show that on randomly generated regular expressions, NL^* outperforms L^* in both these measures.

Fascinated by these results, in this work we extend the definition of residual automata to *universal automata* and *alternating automata*, and the ideas of NL^* to learning algorithms UL^* and AL^* yielding UFAs and AFAs, respectively. A residual NFA for L need not have a state for a residual language of L, say L_i, if L_i can be obtained by a union of other residual languages. Likewise a residual UFA (URFA) need not have a state for a residual language of L, L_i, if L_i can be obtained by an intersection of other residual languages. A residual AFA (ARFA) need not have a state for a residual language of L, L_i, if L_i can be obtained by a monotone combination of some residual languages $L_1, \ldots, L_k$ it does have a state for.

[1] Previous literature used the term RFSA or *residual finite state automata* for what we call here NRFA.

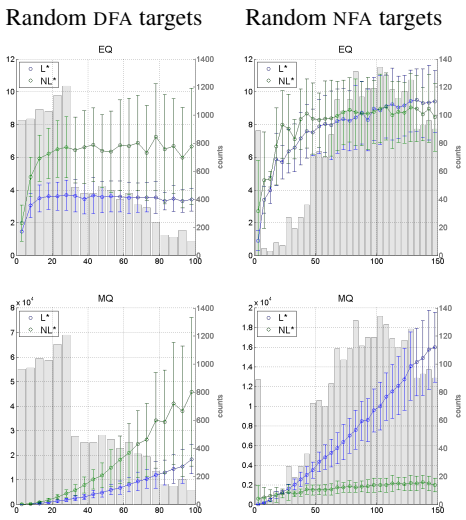

Figure 1: Results of L^* and NL^* on randomly generated DFAs (on the left) and NFAs (on the right). The upper row presents number of equivalence queries; the lower row, number of membership queries. The x-axis shows the number of states in the minimal DFA of the target language. The results are binned in groups of 5, showing the average value. The error bars correspond to the standard deviations. The blue line is for L^* and the green for NL^*. The light grey bars on the background represent the number of experiments in the respective bin, whose scale is on the right.

[Bollig *et al.*, 2009] have shown that NL^* outperforms L^* on randomly generated regular expressions. Random regular expressions, having an operator for union but not for intersection, are more reminiscent of NFAs, so it is likely that NL^* would outperform L^* on randomly generated NFAs. It is not clear how would L^* and NL^* compare on randomly generated DFAs, UFAs and AFAs. In a preliminary experiment we conducted we compared NL^* and L^* on randomly generated DFAs with up to 100 states, and randomly generated NFAs with 10 states (whose minimal equivalent DFAs yielded up to and more than 150 states). As can be seen in Fig. 1, NL^* indeed outperforms L^* on randomly generated NFAs, but on randomly generated DFAs, L^* outperforms NL^*.[2]

Since AFAs generalize both NFAs and UFAs, as well as DFAs, and the ideas of AL^* generalize those of NL^* and L^*, we provide our algorithm as a scheme $\mathrm{X}L^*$, for $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$. That is, the algorithms L^*, NL^*, UL^*, AL^* for generating DFAs, NFAs, UFAs, and AFAs, respectively, are obtained from $\mathrm{X}L^*$ by calling different sub-procedures, parameterized by the type $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$.

We tested all four algorithms L^*, NL^*, UL^* and AL^* on randomly generated DFAs, NFAs, UFAs and AFAs. The results show, that for $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$, roughly speaking, $\mathrm{X}L^*$ outperforms the others on randomly generated X-FAs. There is a clear advantage of AL^* over the others in term of the number

[2] For randomly generated DFAs, the number of states of the learned automata are essentially the same for L^* and NL^*. For randomly generated NFAs, NL^* produces much smaller automata.

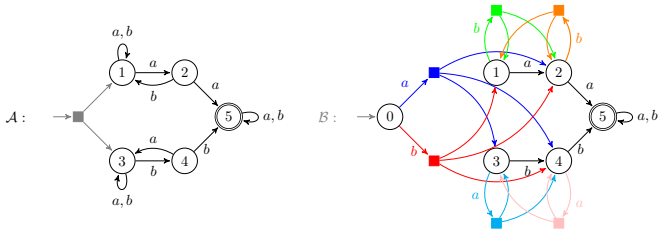

Figure 2: Two ARFAs for $L = \Sigma^* aa\Sigma^* \cap \Sigma^* bb\Sigma^*$ where $\Sigma = \{a, b\}$. Conjunction between edges is depicted by a filled rectangle from which the edges split (the edges and rectangle share the same color for easier parsing).

of states of the learned automaton, and an advantage of L^* in terms of the number of equivalence queries.

2 Residual Alternating Automata

Alternating Automata

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 its elements. An *alternating automaton* 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 $\mathbb{B}_\vee(S)$ and $\mathbb{B}_\wedge(S)$ be the restriction of $\mathbb{B}_+(S)$ to use only $\vee$ or only $\wedge$, respectively. 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*, *universal*, or *deterministic*, respectively.

The formal definition of a run tree of an alternating automaton may be found in [Kupferman and Vardi, 1998]. We denote by $[\![A]\!]$ the set of strings w accepted by $\mathcal{A}$. For an automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, and state $q \in Q$, we use $\mathcal{A}_q$ to denote the automaton obtained from $\mathcal{A}$ by making the initial condition q, i.e. $\mathcal{A}_q = (\Sigma, Q, q, \delta, F)$. Similarly, for a Boolean expression $b \in \mathbb{B}_+(Q)$ we use $\mathcal{A}_b = (\Sigma, Q, b, \delta, F)$ to denote the automaton obtained from $\mathcal{A}$ by making the initial condition b.

Residual Automata

If L is a language and u is a string, the *residual* of L with respect to L is $u^{-1}L = \{v \mid uv \in L\}$. A language R is a residual language of L if there exists a string u such that $R = u^{-1}L$.

Definition 1. *Let $\mathcal{A}$ 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 $[\![D]\!]$. Thus, the minimal DFA is a DRFA, and so is any trimmed DFA.

Example 1. *Let $\Sigma = \{a, b\}$ and $L = \Sigma^* aa\,\Sigma^* \cap \Sigma^* bb\,\Sigma^*$. Fig. 2 shows two* ARFA*s $\mathcal{A}$ and $\mathcal{B}$ recognizing L. 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 $[\![\mathcal{A}]\!] = L$. To see that this 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 $\mathcal{B}$ 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 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 $[\![\mathcal{A}_i]\!] = [\![\mathcal{B}_i]\!]$ for $i \in [1..5]$, and $[\![\mathcal{B}_0]\!] = \epsilon^{-1}L$. Thus $\mathcal{B}$ is also a residual automaton.

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 [Chandra and Stockmeyer, 1976].[3] We show that the same holds for their residual counterparts.

Theorem 1. NRFA *and* URFA *may be exponentially more succinct than* DRFA*s.* ARFA *may be exponentially more succinct than* NRFA*s and* URFA*s, and doubly-exponentially more succinct than* DRFA*s.*

The proof is omitted; the method is 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.

3 Learning Alternating Automata

The L^* algorithm makes use of the notion of an *observation table*. An observation table is a Boolean matrix whose rows and columns are associated with different strings. We define some general notions on Boolean vectors and matrices.

Boolean vectors and matrices Let n be a positive integer and $\{0, 1\}^n$ be all Boolean vectors of length n, also denoted $\mathbb{B}^n$. Extend $\vee$ and $\wedge$ to Boolean vectors componentwise. We also define a partial ordering on Boolean vectors: $u \leq v$ if $u[j] \leq v[j]$ for all indices $1 \leq j \leq n$. Let $\mathbf{0}$ denote the vector of all 0 entries, and $\mathbf{1}$ denote the vector of all 1 entries.

Let M be an $m \times n$ matrix. The i-th row of M is denoted M_i and the i-th column of M is denoted M^i. Note that the rows of M are vectors of size n whereas its columns are vectors of size m. For a subset of indices $I \subseteq [1..m]$ we use M_I for the $|I| \times n$ matrix obtained from M by deleting the rows $j \notin I$. We can analogously define M^I for $I \subseteq [1..n]$. The *row-index* of M is the number of distinct rows of M and its *column-index* is the number of distinct columns in M.

[3] The exact bounds depend on the exact definition of AFA. [Kozen, 1976] defined AFAs in which the transition relations 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, 1976], who showed that exists an AFA with n states for which the smallest DFA requires $2^{2^n/\sqrt{n}}$. [Meyer and Fischer, 1971] use $\exists$ and $\forall$ states, and show that a DFA for an AFA with $5n + 2$ states requires at least 2^{2^n}.

An observation table An observation table $\mathcal{T}$ is a tuple (S, E, M) where S is a list of strings, representing candidate *states*, E is a list of strings, 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 $s_i e_j \in L$. By abuse of notation we use $M_{s,e}$ for $M_{i,j}$ where s is the i-th string of S, and e the j-th string of E. Similarly, we use M_s instead of M_i and for a subset of strings $U \subseteq S$ we use M_U for M_{I_U} where I_U is the set of indices of U.

Let $\Sigma^*_<$ be the list of all strings ordered lexicographically. 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 the states of the minimal DFA for L correspond to distinct rows of $\mathcal{T}_L$.

Monotone, union and intersection bases The L^* algorithm starts with an empty observation table, which it gradually extends and fills using membership queries. In intermediate steps, when the observation tables is *closed* (defined shortly), it builds a corresponding automaton on which it asks an equivalence query. The definition of closed tables makes use of the following general definitions regarding Boolean vectors.

Let $V \subseteq \mathbb{B}^n$ be a set of vectors. For a Boolean expression $b \in \mathbb{B}_+(V)$ we use $[\![b]\!]$ for the element of $\mathbb{B}^n$ obtained by applying b to V. Note that for $v \in V$, $v \in \mathbb{B}_+(V)$ and $[\![v]\!] = v$. For a set of Boolean expressions B we use $[\![B]\!]$ for $\cup_{b \in B}[\![b]\!]$. 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 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.

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 $\text{X} \in \{\text{D}, \text{N}, \text{U}, \text{A}\}$. The definition of a table being closed is with respect to a set $P \subseteq S$. The first part is the same for all four: it requires that the empty string ϵ is in P and that for every $p \in P$, all its one-letter extensions $p\sigma$ for $\sigma \in \Sigma$ are in S.

As for the second part, intuitively, for L^*, the set P is the set of distinct rows of the table. For NL^*, the set P is the set of *prime* rows of the table (that is why we use P to denote it), i.e., the other rows of the table should be expressible in terms of disjunctions of the prime rows. For UL^*, the set P is the set of prime rows w.r.t *conjunction*, i.e. the other rows of the table should be expressible in terms of conjunctions of the prime rows. For AL^*, the set P corresponds to a *monotone basis* for the other distinct rows.

Formally, an observation table $\mathcal{T} = (S, E, M)$ is D-*closed with respect to a subset P of S*, if $\epsilon \in P$, $P\Sigma \subseteq S$ and for every row $s_i \in S \setminus P$, there exists $s_j \in P$ such that $M_i = M_j$. An observation table $\mathcal{T} = (S, E, M)$ is A-*closed* w.r.t. a subset P of S, if $\epsilon \in P$, $P\Sigma \subseteq S$ and M_P is a subset monotone basis for M_S. It is N-*closed* and U-*closed* w.r.t. a subset P of S, if $\epsilon \in P$, $P\Sigma \subseteq S$ and M_P is a subset union basis for M_S or a subset intersection basis for M_S, respectively.

Next we introduce the notion of X-*minimality*. Intuitively, being closed with respect to P guarantees us that all distinct rows are expressible by the allowed combinations of P rows. A trivial way to achieve this is to include all rows. Since we use P 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 P that can be expressed by means of the other rows, we can remove it from P. Formally, let $\mathcal{T}$ be A-*closed* w.r.t P. We say that $\mathcal{T}$ is A-*minimal* w.r.t P if for every $p \in P$ and $P' \subseteq P \setminus p$, $M_p \notin [\![\mathbb{B}_+(P')]\!]$. 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 P. For $s \in S$ let $b_s \in \mathbb{B}_+(P)$ be a Boolean expression over P satisfying $M_s = [\![b_s]\!]$. Then the tuple $(\Sigma, P, \lambda, \delta, F)$ where $F = \{p \in P \mid M_{p,\epsilon} = 1\}$, and $\delta(p, \sigma) = b_{p\sigma}$, is an AFA, which we denote $\mathcal{A}^P_\mathcal{T}$.

3.1 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 set U such that $\mathcal{T}$ is X-closed and minimal w.r.t to it, and (2) given U and a string s, how to find, if possible, an expression b_s, in $\mathbb{B}_\text{X}(U)$ such that $M_s = [\![b_s]\!]$, where $\mathbb{B}_\text{A}(Q)$, $\mathbb{B}_\text{N}(Q)$, $\mathbb{B}_\text{U}(Q)$ and $\mathbb{B}_\text{D}(Q)$ correspond to $\mathbb{B}_+(Q)$, $\mathbb{B}_\vee(Q)$, $\mathbb{B}_\wedge(Q)$ and Q, respectively.

Finding a Union/Intersection Basis
Because of the duality of union and intersection we can consider just one of these, and the results follow for the other.

Claim 1. *Let $v \in \mathbb{B}^n$ and $U \subseteq \mathbb{B}^n$. There is a polynomial time algorithm to determine whether $v \in \mathbb{B}_\vee(U)$.*

Proof. We compute the union of all vectors $u \in U$ s.t. $u \leq v$. Then v is equal to this union if and only if $v \in \mathbb{B}_\vee(U)$. □

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

Theorem 2. *[Bollig* et al.*, 2009] Given a set of vectors V, there is a polynomial time algorithm to find a minimum cardinality subset union basis for V.*[4]

[4]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, 1975] who termed it *the set basis problem.*

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.

Finding a Monotone Basis

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

Claim 2. *Let $v \in \mathbb{B}^n$ and $U \subseteq \mathbb{B}^n$. There there is a polynomial time algorithm to determine whether $v \in \mathbb{B}_+(U)$.*

Proof. If there exists a monotone formula over U representing v, then there exists one in DNF form, i.e., in the form $I_1 \cup I_2 \ldots \cup I_k$ where I_i are intersections of subsets of U. The following procedure checks if there is a DNF formula over U representing v.

Assume $U = \{u_1, \ldots, u_m\}$. A DNF formula over U can be represented by a set of vectors in $\mathbb{B}^m$ since a vector in $\mathbb{B}_m$ can represent a subset S of U, by having index i set to 1 iff the respective vector $v_i \in S$. The given set of vectors U can be represented by an $m \times n$ matrix, whose rows are U. In this matrix, a column M^i represents exactly the set of all vectors u in U, with $u[i] = 1$. Thus, if M^i is one of the disjuncts of D then $[\![D]\!][i] = 1$, unless $M^i = \mathbf{0}$.

Consider the given vector v. Let I_0^v be the set of indices i such that $v[i] = 0$, and likewise I_1^v be the set of indices i such that $v[i] = 1$. Assume D is a DNF formula for v over U. We note that first, for every $i \in I_0^v$, the column vector $M^i \notin D$, unless $M^i = \mathbf{0}$. As otherwise $[\![D]\!][i] = 1$ contradicting it represents v for which $v[i] = 0$. Second, for every $i \in I_1^v$, either the column vector $M^i \in D$ or some column vector M^j s.t. $\mathbf{0} \neq M^j \leq M^i$ is in D, as otherwise none of the disjuncts of D will have 1 in the i-th index, contradicting $v[i] = 1$. Thus, to check if such a DNF formula exists it suffices to check that for all pairs of columns $M^j \leq M^k$ we have $v[j] \leq v[k]$. □

Given v can be represented by a formula in $\mathbb{B}_+(U)$, there may in general be many different formulas achieving that. Considering both constraints in the proof of Claim 2, for $i \in I_1^v$, we must include in the DNF formula the smallest $M^j < M^i$. We return the union of the minimal $M^j < M^i$ for all $i \in I_1^v$. This gives a monotone DNF expression for v; a monotone CNF expression can be derived dually.

Unlike the case with union or intersection, finding a monotone subset of minimum cardinality is NP-hard.

Theorem 3. *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 is omitted; it is a polynomial time reduction from the problem of monotone not-all-equal 3-SAT.

3.2 The learning algorithm

From claims 1 and 2 we can extract a procedure *In*$\mathbb{B}_{\mathrm{X}}(P, s)$ for $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$ that determines whether M_s, the row corresponding to string s in the observation table, can be represented as an adequate combination of M_P, the rows corresponding to strings P, and returns one such formula if the answer is "yes".

Algorithm 1: XL^* for $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}, \mathrm{U}, \mathrm{A}\}$

```
oracles  : MQ, EQ
members: Observation table T = (S, E, M),
         Candidate states set P
methods : IsXClosed, IsXMinimal, XFind&AddCols,
         InB_X, XExtractAut
S = ⟨ϵ⟩, E = ⟨ϵ⟩, P = ⟨ϵ⟩ and M_{ϵ,ϵ} = MQ(ϵ).
repeat
    (a_1, s_1) = T.IsXClosed(P)
    if a_1 = "no" then
        P.AddString(s_1)
    else
        (a_2, s_2) = T.IsXMinimal(P)
        if a_2 = "no" then
            P.RemoveString(s_2)
        else
            A = T.XExtractAut(P)
            (a_3, s_3) = EQ(A)
            if a_3 = "no" then
                T.XFind&AddCols(s_3)
until a_3 = "yes"
return A
```

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 P 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 P, by calling procedure $\mathcal{T}$.*Is*X*Closed*(P). If the answer is "no", the procedure returns a string $s_1 \in S \cup P\Sigma$ that cannot be expressed by $\mathbb{B}_{\mathrm{X}}(P)$ and the algorithm adds s_1 to P and loops. Once $\mathcal{T}$ is closed with respect to P the algorithm checks whether $\mathcal{T}$ is minimal with respect to P, by calling procedure $\mathcal{T}$.*Is*X*Minimal*(P). If the answer is "no", the procedure returns a string $s_2 \in P$ that can be expressed by $\mathbb{B}_{\mathrm{X}}(P \setminus \{s_2\})$. The algorithm then removes s_2 from P, and loops. Once $\mathcal{T}$ has been found to be X-closed and minimal with respect to P, 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.

The procedure X*Find&AddCols* examines all suffixes of s_3. For each suffix it checks whether it induces a new column, when projected on the strings in P and their one-letter extensions. Formally, let $M_{P\Sigma}$ be the matrix obtained from M by keeping only rows corresponding to strings in $P \cup P\Sigma$. The procedure X*Find&AddCols*(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 string s corresponding to a row of $M_{P\Sigma}$. If this column is not a column of $M_{P\Sigma}$ it adds e to E and the obtained MQ results to M.[5]

Proving termination of L^* is quite straightforward, proving

[5]The original formulation of L^*, upon receiving a counterexample from the equivalence oracle, added it and all its prefixes to the

the termination of NL^* is much more intricate since unlike the case of L^*, processing of a counter example does not necessarily yield a new row in the observation table. This is true for AL^* as well. Our proof of termination is based on the following theorem that guarantees that processing of a counter example yields a new column in the observation table.

Theorem 4. *Let $\mathcal{T} = (S, E, M)$ be an observation table for L, which is closed and minimal w.r.t. P, and let $\mathcal{A} = \mathcal{A}^P_{\mathcal{T}}$. If $w \in [\![\mathcal{A}]\!] \iff w \notin L$ for some string w, then there is some suffix v of w such that in $\mathcal{T}' = (S, E \cup \{v\}, M)$ the column M^v is different from all other columns.*

The proof is omitted; the method builds on comparing for increasing suffixes v of w whether $v \in [\![\mathcal{A}_{p_i}]\!] \iff p_i v \notin L$ for every $p_i \in P$.

Clearly, if the algorithm terminates it returns an AFA recognizing the unknown language L. We express the complexity of the algorithm as a function of three parameters: the row-index of the complete observation table, which we denote n; the column-index of the complete observation table, which we denote m; and the maximum length of a counterexample returned by the equivalence oracle, which we denote ℓ. [6] Note that when a new row is added to the table, it does not imply that a new column is created, and vice versa.

Before we proceed we note that all the involved procedures *Is*X*Closed*(), *Is*X*Minimal*(), X*Find&AddCols*() and X*ExtractAut*() may invoke calls to MQ, and all but the latter use *In*$\mathbb{B}_{\mathrm{X}}$() as a sub-procedure. Since each equivalence query yields a new experiment, the number of equivalence queries and the number of iterations of the main loop is bounded by m. Each call to procedure *AddString* results in the addition of a string s to S introducing a new row in M, and thus can be called at most n times. The size of P is bounded by n, thus each iteration of the main loop involves at most n calls to *RemoveString*. The processing of the counterexample by *Find&AddCols* goes over all suffixes of the counter example, at most ℓ, and all rows of $P \cup P\Sigma$, at most $n + n|\Sigma|$. Thus overall running time of the algorithms is bounded by *poly*$(m, n, \ell, |\Sigma|)$.

Theorem 5. *The algorithm AL^* returns an* AFA *for an unknown regular language L, after at most m equivalence queries, and $O(mn\ell)$ membership queries, where n and m are the row-index and column-index of $\mathcal{T}_L$, respectively, and ℓ is the length of the longest counterexample.* [7]

rows of the observation table. [Maler and Pnueli, 1995] suggested instead to add the counterexample and all its suffixes to the columns. [Bollig *et al.*, 2009] pursue this direction. We modified this approach to include just those suffixes that will contribute a new distinct column to the matrix.

[6] The row-index of $\mathcal{T}_L$ equals the number of states of the minimal DFA. The column-index of $\mathcal{T}_L$ equals the number of states in the reverse language of L. However, one can argue that the column-index is a complexity measure of the given language L itself.

[7] We note that [Bollig *et al.*, 2009] as well provide the complexity measure of NL^* in terms of the minimal DFA (rather than the minimal NRFA). The number of equivalence and membership queries of NL^* is bounded by $O(n^2)$ and $O(\ell n^3)$, resp. It is hard to compare n^2 to m which may be exponentially bigger or smaller than n [Leiss, 1981]. We note that on our experiments (see Section 4) the max number of columns in the obtained observation tables for AL^* was much smaller than n^2: for random UFAs and AFAs with minimal DFA of at most 300 states, it was 45 and 150, respectively (and the max number of equivalence queries was 21 and 50, respectively).

4 Empirical Results

We implemented the algorithmic scheme XL^* and the sub-procedures *Is*X*Closed*, *Is*X*Minimal*, X*Find&AddCols*, *In*$\mathbb{B}_{\mathrm{X}}$, X*ExtractAut* for all X $\in$ D, N, U, A. We have tested the four resulting algorithms L^*, NL^*, UL^* and AL^* on four sets of randomly generated automata; in each case the alphabet size was 3 and the number of states was as indicated. The first set consists of randomly generated DFAs of size 1 to 100. The second set consists of randomly generated NFAs of size 10. The third set consists of randomly generated UFAs of size 10. The forth set consists of randomly generated AFAs of size 7. Equivalence queries were implemented by randomly querying 10,000 membership queries. Lengths were uniformly chosen from 0 to $t + 2$ where t is the number of states in the target X-FA. For AFA targets the initial condition and transition relation chose randomly a formula with one alternation of disjunctions and conjunctions.

Figure 3 shows the results of L^*, NL^*, UL^* and AL^* on randomly generated AFAs (on the left) and UFAs (on the right). The upper row presents number of states in the learned automaton, the middle row the number of equivalence queries, and the lower row, the number of membership queries. The x-axis shows the number of states in the minimal DFA of the target language. The results are binned in groups of 10 showing the average value. The error bars correspond to the standard deviations. The blue line is for L^*, the green for NL^*, the purple line for UL^* and the red line for AL^*. The light grey bars on the background represent the number of experiments in the respective bin, whose scale is on the right.

On AFA targets we can see that the number of states produced by AL^*, is significantly smaller than the others, and the number of states produced by NL^* and UL^* is roughly the same. AL^* also outperforms the others in the number of membership queries, followed by UL^*, NL^*, and L^*, in this order. On the measure of equivalence queries, L^* uses the fewest equivalence queries, followed by UL^*, NL^* and AL^*.

On UFA targets, UL^* comes first in all measures, but the difference in the obtained number of states between AL^* and UL^* is negligible. For lack of space we do not include the results on the sets of randomly generated DFAs and NFAs. The latter is, roughly speaking, symmetrical to that of randomly generated UFAs where the blue and purple colors switch. For DFAs, all algorithms produce essentially the same number of states, which is the number of states in the minimal DFA. In this set, L^*, outperforms the others in all parameters, NL^* and UL^* perform equally well in all measures, and AL^* asks more equivalence and membership queries than the other three.

5 Discussion

Since the empirical results show that XL^* performs better on randomly generated X-RFAs, it is clear that if one has some knowledge of the target language, classifying it as being of a deterministic, existential, universal or alternating nature,

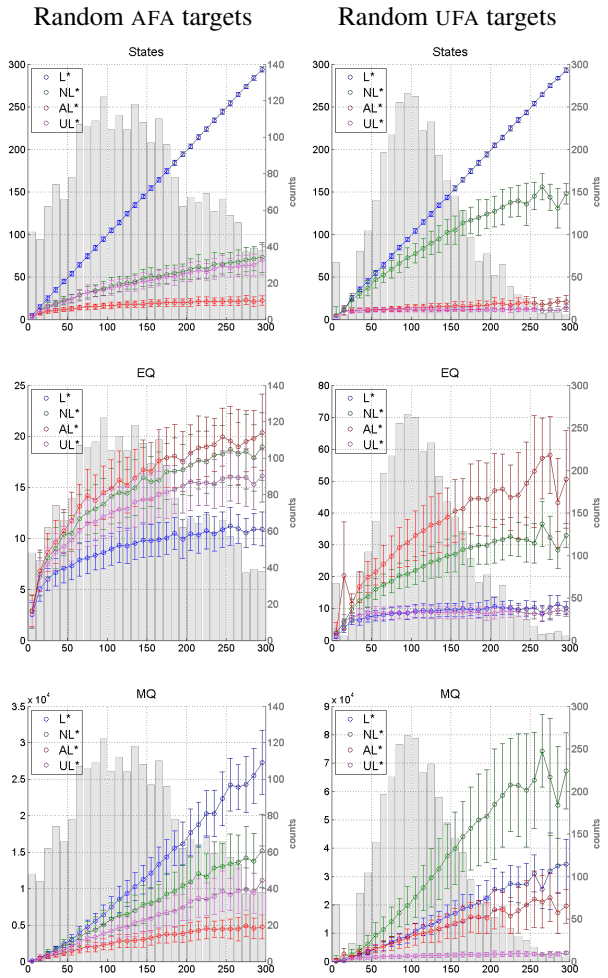

Figure 3: Results of L^*, NL^*, UL^* and AL^* on randomly generated AFAs (on the left) and UFAs (on the right).

then the corresponding XL^* algorithm is to be preferred. [8] In cases where no such apriori knowledge is available, one might choose the algorithm that has better overall performance on the parameter that is most significant/costly for the application at hand. For instance, if succinctness is crucial one might chose AL^*, but if equivalence queries are very expensive one might prefer L^*. Since all XL^*'s algorithm share the same observation table, one might also consider heuristics where, at different stages of the algorithm, it tries to learn different target types among DRFA, NRFA, URFA and ARFA.

A generalization of alternating automata as defined here (sometimes referred to as *Boolean automata* [Leiss, 1981]), allows any Boolean formulas, i.e., all formulas over $\vee$, $\wedge$ and $\neg$, in the initial condition and the transition relations. We have not studied a corresponding definition for the residual case or a learning algorithm for it. However, De Morgan's Laws show that any Boolean formula has an equivalent formula in which negation is only applied to variables, so it seems reasonable that the ideas of AL^* can be extended to this case by considering for the basis the given set of vectors and their negations.

Our focus in this work was to obtain an L^*-style algorithm for AFAs. To this aim, we introduced residual AFAs, a generalization of residual NFAs. However, while we conjecture that AL^* produces ARFAs, and our experiments did not refute this conjecture, we weren't yet able to prove it. Residual NFAs have been studied from automata theoretic preservative [Denis *et al.*, 2001a] and are now quite well understood. A better understanding of ARFAs, might help us fill in the missing part of the puzzle, and is worthwhile regardless.

Acknowledgments We would like to thank David Eisenstat for suggesting the approach used for Claim 2 and Nissim Ofek for his suggestions with regard to plotting the experiments.

[8] Phrasing properties in terms of universal and alternating automata may not be natural or intuitive. However, in many situations, e.g., model checking of hardware systems, one usually writes a lot of properties, and the verification tasks concerns the conjunction of all, which takes a universal nature if the single properties are deterministic in nature, and an alternating nature if the single properties are of an existential nature.

References

[Angluin, 1987] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[Bollig *et al.*, 2009] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In *21st IJCAI*, pp. 1004–1009, 2009.

[Chandra and Stockmeyer, 1976] AK. Chandra and LJ. Stockmeyer. Alternation. In *17th FOCS*, pp. 98–108, 1976.

[Denis *et al.*, 2001a] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. In *18th STACS*, pp. 144–157, 2001.

[Denis *et al.*, 2001b] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSA. In *12th ALT*, pp. 348–363, 2001.

[Esposito *et al.*, 2002] Y. Esposito, A. Lemay, F. Denis, and P. Dupont. Learning probabilistic residual finite state automata. In *6th ICGI*, pp. 77–91, 2002.

[Kasprzik, 2010] A. Kasprzik. Learning residual finite-state automata using observation tables. In *12th DCFS*, pp. 205–212, 2010.

[Kasprzik, 2011] A. Kasprzik. Inference of residual finite-state tree automata from membership queries and finite positive data. In *15th DLT*, pp. 476–477, 2011.

[Kozen, 1976] D. Kozen. On parallelism in Turing machines. In *17th FOCS*, pp. 89–97, 1976.

[Kupferman and Vardi, 1998] O. Kupferman and MY. Vardi. Weak alternating automata and tree automata emptiness. In *13th STOC*, pp. 224–233, 1998.

[Leiss, 1981] EL. Leiss. Succint representation of regular languages by boolean automata. *Theor. Comput. Sci.*, 13:323–330, 1981.

[Maler and Pnueli, 1995] O. Maler and A. Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995.

[Meyer and Fischer, 1971] AR. Meyer and MJ. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Symp. on Switching and Autom. Theory*, pp. 188–191, 1971.

[Stockmeyer, 1975] LJ. Stockmeyer. The set basis problem is NP-complete. Technical Report RC-5431, IBM, 1975.