Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Learning Residual Alternating Automata

Sebastian Berndt, Maciej Liśkiewicz,* Matthias Lutter, Rüdiger Reischuk
Institute for Theoretical Computer Science, University of Lübeck
Ratzeburger Allee 160, 23562 Lübeck, Germany
{berndt,liskiewi,lutter,reischuk}@tcs.uni-luebeck.de

Abstract

Residuality plays an essential role for learning finite automata. While residual deterministic and non-deterministic automata have been understood quite well, fundamental questions concerning alternating automata (AFA) remain open. Recently, Angluin, Eisenstat, and Fisman (2015) have initiated a systematic study of residual AFAs and proposed an algorithm called AL* – an extension of the popular L* algorithm – to learn AFAs. Based on computer experiments they have conjectured that AL* produces residual AFAs, but have not been able to give a proof. In this paper we disprove this conjecture by constructing a counterexample. As our main positive result we design an efficient learning algorithm, named AL**, and give a proof that it outputs residual AFAs only. In addition, we investigate the succinctness of these different FA types in more detail.

1 Introduction

Learning finite automata is an important issue in machine learning and of great practical significance to solve substantial learning problems like pattern recognition, robot's navigation, automated verification, and many others (see e. g. the textbook (De la Higuera 2010)). Depending on applications, different types of automata might be required as desirable targets of learning. The ones of particular concern are deterministic (DFA), non-deterministic (NFA), the dual of NFAs – the universal finite automata (UFA), and their generalization – the alternating finite automata (AFA). Though they are of the same expressive power, the automata have different modeling capabilities and succinctness properties. A *minimal* (w. r. t. the number of states) DFA might be exponentially larger than an NFA and double-exponentially larger than an AFA. Thus, for many applications, e. g. in automated verification, it is desirable to work directly with AFAs rather than with the other types as the membership-problem for AFAs is still efficiently solvable.

In the common exact learning framework for FA the learner can ask *membership* queries to test if a word is accepted by the unknown target automaton and *equivalence* queries to compare his current hypothesis and, if there is a mismatch to receive a counterexample. This model has been introduced in (Angluin 1987) that launched a tremendous amount of subsequent research yielding many effective algorithms of relevance in machine learning and other areas.

Angluin (1987) has provided an algorithm, named L* that learns a *minimal* DFA in polynomial time. The minimality of the resulting DFA plays an important role here since this condition makes it unique (up to naming of states). Thus, L* learns precisely the target automaton if this is minimal.

Beside uniqueness, minimal DFAs have also another nice property termed *residuality*. An automaton $\mathfrak{A}$ accepting a language L is residual if every state q of $\mathfrak{A}$ can be associated with a word w_q such that the language accepted by $\mathfrak{A}_q$ – the automaton $\mathfrak{A}$ that starts in q, is exactly the set of words v for which $w_q v$ is in L. Thus, every state q of $\mathfrak{A}$ corresponds to the residual language of L determined by w_q.

For many learning algorithms the residuality property plays an essential role in inferring the target automaton. Angluin's L* algorithm makes heavy use of this concept: The states of a hypothesized automaton are represented by a prefix-closed set of strings such that for every state q_s corresponding to a string s, the language accepted from q_s is residual with respect to s and the target language. Unfortunately, non-deterministic automata, in general do not satisfy the residuality property. Even worse, for an NFA $\mathfrak{A}$ languages accepted by $\mathfrak{A}_q$, for states q of $\mathfrak{A}$, have no natural interpretation and two minimal NFAs can be non-isomorphic. The disadvantageous properties may lead to ambiguity problems and difficulties in learning automata. Moreover the goal is to learn automata containing a certain structure, that may be helpful for later use in specific applications, like e. g. in model checking. Residuality is one such structural property that allows to assign a natural semantic to the states of a complex automaton. This allows a simpler analysis of the (possibly) involved behaviour of the automaton.

Denis, Lemay, and Terlutte (2001) introduced the class of residual NFA (RNFA). For every regular language L there is a unique RNFA $\mathfrak{A}^L$ called *canonical* such that the number of states is minimal, the number of transitions between states is maximal, and for every state q of $\mathfrak{A}^L$ the language accepted by $\mathfrak{A}^L_q$ is residual. In addition, $\mathfrak{A}^L$ can be exponentially more succinct than the equivalent minimal DFA. Using the residuality property, Bollig et al. (2009) proposed a sophisticated extension of Angluin's algorithm named NL* that learns a

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG) grant LI 634/4-1.

canonical RNFA with a polynomial number of membership and equivalence queries.

Recently, Angluin, Eisenstat, and Fisman (2015) extended the definition of residual automata to universal and alternating automata, and provided UL*, a learning algorithm for UFAs, and AL*, a learning algorithm for AFAs. To analyze the advantages and trade-offs among these algorithms, the authors performed experiments and showed that for randomly generated automata, AL* outperforms the other algorithms w. r. t. the number of membership queries, but w. r. t. the number of equivalence queries L* is the best, followed by UL*, NL*, and AL* (which is justified due to the succinctness obtained). However, as the authors write, they have not been able to prove that AL* always outputs residual AFAs. Based on the experiments they have conjectured that this property indeed holds, but left its proof as future work.

In this paper we disprove their conjecture by providing a counterexample that has been constructed with the help of specially designed software tools for learning residual automata. Next, we continue the systematic study of residual AFAs and discuss several properties to get a better understanding of these machines. As our main positive result we design an efficient learning algorithm, named AL**, and give a proof that it outputs residual AFAs only. In addition, we investigate the succinctness of these different FA types in more detail.

2 Preliminaries

We denote the symmetric difference of sets by Δ, the Boolean value "true" as $\top$ and the value "false" as $\bot$. For a set S we denote by $\mathcal{F}(S)$ all formulas over S that can be generated with the binary operators $\wedge$ and $\vee$. For $R \in \{\wedge, \vee\}$, the set $\mathcal{F}_R(S)$ is the subset of $\mathcal{F}(S)$ of formulas that are build form S and operator R. In order to help the reader in understanding the complex behaviour of alternating finite automata, we compare them to the well-known deterministic and non-deterministic models.

2.1 Automata

An *alternating finite automaton (AFA)* – first introduced by Chandra, Kozen, and Stockmeyer (1981) – with alphabet Σ is a four-tuple (Q, Q_0, F, δ), where Q is the *set of states*, $Q_0 \in \mathcal{F}(Q)$ is the *initial configuration*, $F \subseteq Q$ are the *accepting states*, and $\delta\colon Q \times \Sigma \to \mathcal{F}(Q)$ is the *transition function*. If Q_0 and, for all $q \in Q$ and all $a \in \Sigma$, the transition $\delta(q, a)$ consist of a single state, such an automaton is called a *deterministic automaton (DFA)*. If $Q_0 \in \mathcal{F}_\vee(Q)$ and $\delta(q, a) \in \mathcal{F}_\vee(Q)$ for all $q \in Q$ and all $a \in \Sigma$, it corresponds to *non-deterministic finite automata (NFA)*. E. g. if $\delta(q, a) = p_1 \vee p_2$, this describes a non-deterministic choice between p_1 or p_2. If $Q_0 \in \mathcal{F}_\wedge(Q)$ and $\delta(q, a) \in \mathcal{F}_\wedge(Q)$ for all $q \in Q$ and all $a \in \Sigma$, the definition corresponds to *universal finite automata (UFA)*, where, e. g. a transition $\delta(q, a) = p_1 \wedge p_2$ leads to state p_1 and state p_2.

As usual, the function δ is extended to arbitrary formulas and strings: If $\varphi \in \mathcal{F}(Q)$, assume w. l. o. g. φ is in disjunctive normal form (DNF) with $\varphi = \bigvee_i M_i$ and $M_i = \bigwedge_j q_{i,j}$, then we define $\delta(\varphi, a) = \bigvee_i \bigwedge_j \delta(q_{i,j}, a)$ for a single symbol $a \in \Sigma$ and $\delta(\varphi, \epsilon) = \varphi$ for the empty string ϵ. For a non-empty string $wa \in \Sigma^*$, we define $\delta(\varphi, wa) = \delta(\delta(\varphi, w), a)$. For an NFA, this definition simply reduces to $\delta(q \vee p, a) = \delta(q, a) \vee \delta(q, b)$ as usual, if one interprets the formula $q \vee p$ as set $\{q, p\}$.

For an AFA $\mathfrak{A} = (Q, Q_0, F, \delta)$ and a formula $\varphi \in \mathcal{F}(Q)$, we define the *evaluation* of φ, denoted as $(\!|\varphi|\!)$, recursively as follows: for the empty set $\emptyset$ of states we let $(\!|\emptyset|\!) = \bot$, for singletons we define $(\!|q|\!) = \top$ if $q \in F$, $(\!|q|\!) = \bot$ if $q \notin F$, and finally $(\!|\varphi R \psi|\!) = (\!|\varphi|\!) R (\!|\psi|\!)$ for $R \in \{\wedge, \vee\}$. The AFA $\mathfrak{A}$ *accepts* a word w, if $(\!|\delta(Q_0, w)|\!) = \top$. For an NFA, $(\!|\delta(Q_0, w)|\!) = \top$ expresses the same as $\{q_1, \ldots, q_k\} \cap F \neq \emptyset$ if $\delta(Q_0, w) = q_1 \vee \ldots \vee q_k$ (i. e. when starting with initial configuration and reading the word w some accepting state is reached) and for a UFA it is the same as $\{q_1, \ldots, q_k\} \subseteq F$ if $\delta(Q_0, w) = q_1 \wedge \ldots \wedge q_k$ (i. e. all states reached are accepting). The *language* $L(\mathfrak{A})$ of the automaton $\mathfrak{A}$ is the set of all accepted strings.

For an AFA $\mathfrak{A} = (Q, Q_0, F, \delta)$ and a state $q \in Q$, we write $\mathfrak{A}_q$ to indicate the automaton $\mathfrak{A}_q = (Q, q, F, \delta)$ that starts with configuration q instead of Q_0.

2.2 Residuality

Let $L \subseteq \Sigma^*$ be a regular language. For a word $u \in \Sigma^*$, we define the *residual language* $u^{-1}L$ as $\{v \in \Sigma^* \mid uv \in L\}$. The set of all residual languages of L is denoted by $\mathrm{RES}(L)$. A residual language $u^{-1}L$ is called $\cup$-*prime*, resp. $\cap$-*prime* if $u^{-1}L$ cannot be defined as the union, resp. intersection of other residual languages. We denote the subsets of $\mathrm{RES}(L)$ by $\cup$-Primes(L), resp. $\cap$-Primes(L).

An automaton $\mathfrak{A}$ with states Q is *residual*, if $L(\mathfrak{A}_q) \in \mathrm{RES}(L)$ for all $q \in Q$, i. e. if every state corresponds to a prefix u and its residual language $u^{-1}L$. Let RNFA, RUFA and RAFA denote the appropriate residual restrictions.

2.3 Learning Algorithms

All of the learning algorithms xL* for automata (i. e. L*, NL*, UL*, and AL*) and our AL** follow a very similar pattern. Two sets $U, V \subseteq \Sigma^*$ are constructed, where U is prefix-closed and V is suffix-closed. For all strings $uv \in UV$ or $uav \in U\Sigma V$ a membership query is performed. The resulting matrix, indexed by $U \cup U\Sigma$ and V is called a *table*. The rows indexed by U correspond to possible states. To minimize the number of states, a subset B of rows (a *base*) is constructed such that all rows can be built from the elements of B. The specific way to "build" a row depends on the type of automaton. A hypothesized automaton is constructed from this subset B. For a row r_u indexed by $u \in U$ and a symbol $a \in \Sigma$, the transition $\delta(r_u, a)$ equals the formula that "builds" the row indexed by ua.

Formally – similar to Bollig et al. (2009) – for the prefix-closed set U and the suffix-closed set V, we define a $|U \cup U\Sigma| \times |V|$ *table* $\mathcal{T} = (T, U, V)$ with entries in $\{+, -\}$ determined by function $T\colon \Sigma^* \to \{+, -, \bot\}$ specified below. Let $W(\mathcal{T})$ denote the set $(U \cup U\Sigma)V$, for short. We call $W(\mathcal{T})$ the set of words described by $\mathcal{T}$.

Define T for $w \in \Sigma^*$ as

$$T(w) = \begin{cases} \bot & \text{if } w \notin W(\mathcal{T}), \\ + & \text{if } w \in W(\mathcal{T}) \cap L, \\ - & \text{if } w \in W(\mathcal{T}) \setminus L. \end{cases}$$

The entry of $\mathcal{T}$ in row x and column y is equal to $T(xy)$. Note, that to define $\mathcal{T}$ we need only values T on $W(\mathcal{T})$. We extend the domain of T to all words over Σ for the sake of completeness. For an example for $\mathcal{T}$ and for the forthcoming definitions concerning tables, see Fig. 1.

		V: ϵ	ab	b
U	ϵ	-	+	-
	a	-	-	+
R	b	-	-	-
	aa	-	-	-
	ab	+	-	+

Figure 1: Table $\mathcal{T} = (T, U, V)$ for the language $L = ab^+$, with $U = \{\epsilon, a\}$, $V = \{\epsilon, ab, b\}$, and $R = U\Sigma \setminus U = \{b, aa, ab\}$. The entries of the table are determined by T: the value in row x and column y is equal to $T(xy)$. For example, the value in row ab and column b is $+$ since $T(abb) = +$, as $abb \in L$ and $abb \in W(\mathcal{T})$. An example row is e. g. $r_\epsilon = (- + -)$. Furthermore, $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T}) = \{r_\epsilon, r_a\}$.

An automaton $\mathfrak{A}$ and a table $\mathcal{T}$ are *compatible*, if for all $w \in W(\mathcal{T})$, $\mathfrak{A}$ accepts w iff $T(w) = +$.

For every $u \in U \cup U\Sigma$, we associate a vector r_u of length $|V|$ over $\{+, -\}$ with $r_u[v] = T(uv)$ for $v \in V$. The vector r_u is called the *row* of u. The set of all rows is denoted by $\mathrm{Rows}(\mathcal{T})$. An important subset, denoted by $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, are those r_u with $u \in U$.

Finally, we say that $\mathcal{T} = (T, U, V)$ is *consistent* if for every $u, u' \in U$ and $a \in \Sigma$, we have that $r_u = r_{u'}$ implies $r_{ua} = r_{u'a}$. If $\mathcal{T}$ is consistent then, to simplify the notation, for any $r \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ and $a \in \Sigma$ we write ra for the vector r_{ua} s.t. $u \in U$ is any string with $r_u = r$.

3 Learning Residual Universal Automata

The classical result of Angluin (1987) that one can learn the unique minimal DFA for a regular language L from membership and equivalence queries – the algorithm L* – has been extended by Bollig et al. (2009) to NFAs. They have designed an algorithm NL* that learns the unique residual NFA with a minimal number of states and a maximal number of transitions between these states (the *canonical RNFA*) accepting L. Angluin, Eisenstat, and Fisman (2015) presented a modification of NL* named UL* algorithm to learn a residual UFA, but without a detailed analysis.

To better understand residual UFAs we introduce the following definition.

Definition 1 (Canonical RUFA)**.** *The canonical RUFA for a regular language L is the tuple (Q, Q_0, F, δ) where $Q = \cap$-Primes(L), $Q_0 = \{L' \in Q \mid L' \subseteq L\}$, $F = \{L' \in Q \mid \epsilon \in L'\}$, and $\delta(L_1, a) = \{L_2 \in Q \mid a^{-1}L_1 \subseteq L_2\}$.*

The canonical RUFA has the minimal number of states and the maximal number of transitions between these states, which makes it unique. In the following we prove that UL* always outputs such automata.

The order $- \leq +$ on the set $\{+, -\}$ is extended to a partial order on vectors by requiring $\leq$ to hold for each component. The binary operators $\sqcap, \sqcup$ on the set $\{+, -\}$ are defined by $a \sqcap b = \min\{a, b\}$ and $a \sqcup b = \max\{a, b\}$. For vectors, these operators are extended by performing the operation componentwise.

We say that a row r_u is $\sqcap$-*composite* if there are rows $r_{u_1}, r_{u_2}, \ldots, r_{u_k} \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, with $r_{u_i} \neq r_u$, such that $r_u = \bigsqcap_{i=1}^{k} r_{u_i}$. For example, in Fig. 1, the row r_b is composite, as $r_b = r_\epsilon \sqcap r_a$. Otherwise, r_u is $\sqcap$-*prime*. Let $\mathrm{Primes}_\sqcap(\mathcal{T})$ be the set of $\sqcap$-prime rows in $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$. For the table $\mathcal{T}$ in Fig. 1, r_ϵ, r_a, r_{ab} are $\sqcap$-prime, and $\mathrm{Primes}_\sqcap(\mathcal{T}) = \{r_\epsilon, r_a\}$. To simplify the notation, for every $r_u \in \mathrm{Rows}(\mathcal{T})$, let $\mathbb{B}_\sqcap(r_u) = \{r_{u'} \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T}) \mid r_u \leq r_{u'}\}$.

A table $\mathcal{T}$ is $\sqcap$-*closed* if every row $r_u \in \mathrm{Rows}(\mathcal{T})$ can be generated from a subset of rows in $\mathrm{Primes}_\sqcap(\mathcal{T})$ that are combined with the $\sqcap$ operator. A subset of rows that can generate all rows of a table $\mathcal{T}$ using $\sqcap$ is called a $\sqcap$-*basis*. Thus, $\mathcal{T}$ is $\sqcap$-closed if $\mathrm{Primes}_\sqcap(\mathcal{T})$ is a $\sqcap$-basis for $\mathcal{T}$. Note that the table $\mathcal{T}$ in Fig. 1 is *not* $\sqcap$-closed, as the row $r_{ab} \in \mathrm{Rows}(\mathcal{T})$ is not composable by rows of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.

Algorithm UL* constructs the RUFA in a way dual to the nondeterministic case. For a consistent and $\sqcap$-closed table $\mathcal{T}$, let the UFA $\mathfrak{A}(\mathcal{T}) = (Q, Q_0, F, \delta)$, with $Q = \mathrm{Primes}_\sqcap(\mathcal{T})$, $Q_0 = \mathbb{B}_\sqcap(r_\epsilon) \cap Q$, $F = \{r \in Q \mid r[\epsilon] = +\}$, and for all $r \in Q$ and $a \in \Sigma$, let $\delta(r, a) = \mathbb{B}_\sqcap(ra) \cap Q$.

Lemma 1. *For all $u, u' \in U$ with $r_u, r_{u'} \in Q$, $v \in V$ and $r \in \delta(Q_0, u)$ it holds:*

1. $r_u[v] = + \iff \delta(r_u, v) \subseteq F$,
2. $r_\epsilon[v] = + \iff \delta(Q_0, v) \subseteq F$.

If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible then additionally

3. $r_u \in \delta(Q_0, u)$ *and* $r_u \leq r$,
4. $r_{u'} \leq r_u \iff \forall w\ \delta(r_u, w) \nsubseteq F \Rightarrow \delta(r_{u'}, w) \nsubseteq F$.

Theorem 1. *If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible, then $\mathfrak{A}(\mathcal{T})$ is the canonical RUFA.*

Proof. We first show that the automaton is residual. Let $r \in Q$. Thus, $r \in \delta(Q_0, u_r)$ and hence $(u_r)^{-1}L(\mathfrak{A}(\mathcal{T})) \subseteq L(\mathfrak{A}_r(\mathcal{T}))$. Furthermore, for all $r' \in \delta(Q_0, u_r)$, we have $r \leq r'$ and thus $L(\mathfrak{A}_r(\mathcal{T})) \subseteq L(\mathfrak{A}_{r'}(\mathcal{T}))$ by Lemma 1. This implies $L(\mathfrak{A}_r(\mathcal{T})) \subseteq (u_r)^{-1}L(\mathfrak{A}(\mathcal{T}))$. Hence, $L(\mathfrak{A}_r(\mathcal{T})) = (u_r)^{-1}L(\mathfrak{A}(\mathcal{T}))$. The language $L(\mathfrak{A}_r(\mathcal{T}))$ is also $\cap$-prime, as r is $\sqcap$-prime due to Lemma 1. □

4 Learning Alternating Automata

This section contains the main results of our work. In (Angluin, Eisenstat, and Fisman 2015), the algorithm AL* to learn alternating automata has been presented and its running time analyzed. As noted by Angluin, Eisenstat, and Fisman, properties of the automata produced remain unclear. We first prove several properties of AL* and disprove the

conjecture of residuality. Then we present the modified algorithm AL** that guarantees residuality. Finally we discuss how to find a provably good basis for AFAs (defined in the next subsection).

4.1 Analysis of AL* Algorithm

Let us review the construction of the automata generated by AL* and analyze the properties of these automata in detail.

Note that we use the basic version of the algorithm AL* without the optimizations suggested by Angluin, Eisenstat, and Fisman (2015), as some of these optimizations may lead to a non-terminating behaviour of the algorithm.

For a formula $\varphi \in \mathcal{F}(\mathrm{Rows}(\mathcal{T}))$ on the rows of a table, we define the evaluation $[\![\varphi]\!]$ by $[\![r_u]\!] = r_u$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \sqcap [\![\psi]\!]$ and $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \sqcup [\![\psi]\!]$ and extend this to a set B of formulas by $[\![B]\!] = \{[\![\varphi]\!] \mid \varphi \in B\}$. For example,

$$[\![(+--\wedge---+) \vee -++-]\!] = -++.$$

In the following P will always denote a subset of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$. P is a $(\sqcup, \sqcap)$-*basis* for $\mathcal{T}$ (in the following simply called a basis) if $\mathrm{Rows}(\mathcal{T}) \subseteq [\![\mathcal{F}(P)]\!]$. $\mathcal{T}$ is then called *P-closed*. $\mathcal{T}$ is called *P-minimal* if P is a minimal basis for $\mathcal{T}$, i. e. for all $p \in P$, $P \setminus \{p\}$ is not a basis.

For a P-closed table $\mathcal{T}$ and $v \in V$, let $M^P(v)$ be the monomial defined by

$$M^P(v) := \bigwedge_{p \in P, p[v]=+} p\,,$$

which is a maximal one over all monomials in $\mathcal{F}_\wedge(P)$ such that $[\![M^P(v)]\!][v] = +$. For a P-closed table $\mathcal{T}$ and $r \in \mathrm{Rows}(\mathcal{T})$, let $b^P(r) \in \mathcal{F}(P)$ be the expression

$$b^P(r) = \bigvee_{v \in V, r[v]=+} M^P(v)$$

representing r. Note that $[\![b^P(r)]\!] = r$.

Let M be a monomial and $a \in \Sigma$. We define Ma as the monomial derived from M by replacing every row $r \in P$ of M by ra.

For a DNF-formula φ consisting of monomials M_i, we use the notation $M_i \sqsubset \varphi$ and for a monomial $M = \bigwedge_j x_j$ the notation $x_j \sqsubset M_i$ for its literals x_j. For formulas $\varphi(x_1, \ldots, x_k)$ and $\psi(x_1, \ldots, x_k)$ with literals $x_1, \ldots, x_k$ that represent vectors r over $\{+, -\}$, we say that φ and ψ are *equivalent* (in symbols $\varphi \equiv \psi$), if $[\![\varphi(r_1, r_2, \ldots, r_k)]\!] = [\![\psi(r_1, \ldots, r_k)]\!]$ for all vectors $r_1, \ldots, r_k$ of identical length. For a formula φ, let φ_{DNF} denote a DNF-formula that is equivalent to φ.

For a consistent and P-closed table $\mathcal{T}$, let us define the AFA $\mathfrak{A}^P(\mathcal{T}) = (Q, Q_0, F, \delta)$ as follows: $Q = P$, $Q_0 = b^P(r_\epsilon)$, $F = \{r \in P \mid r[\epsilon] = +\}$, and for all $r \in Q$ let $\delta(r, a) = b^P(ra)$.

Note that $\delta(r, a) = b^P(ra)$ is always a DNF-formula.

Lemma 2. *For every $\varphi \in \mathcal{F}(Q)$ and every automaton $\mathfrak{A}^P(\mathcal{T})$: $(\!|\varphi|\!) = \top$ iff $[\![\varphi]\!][\epsilon] = +$.*

In the following, fix a regular language L, a prefix-closed set U, a suffix-closed set V, the corresponding table $\mathcal{T}$ and a minimal basis P of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.

Lemma 3. *$r[v] = [\![\delta(r, v)]\!][\epsilon]$ for all $r \in P$ and $v \in V$.*

Lemma 4. *$[\![\varphi]\!][v] = [\![\delta(\varphi, v)]\!][\epsilon]$ for all $\varphi \in \mathcal{F}(P)$ and $v \in V$.*

Lemma 5. *If $\mathcal{T}$ and $\mathfrak{A}^P(\mathcal{T})$ are compatible, then for every $u \in U$ with $r_u \in P$ it holds $L(\mathfrak{A}^P_{r_u}(\mathcal{T})) \subseteq u^{-1}L(\mathfrak{A}^P(\mathcal{T}))$.*

Proof. Assume $L(\mathfrak{A}^P_{r_u}(\mathcal{T})) \not\subseteq u^{-1}L(\mathfrak{A}^P(\mathcal{T}))$, i. e. there exists a string ω such that $\omega \in L(\mathfrak{A}^P_{r_u}(\mathcal{T}))$ and $\omega \notin u^{-1}L(\mathfrak{A}^P(\mathcal{T}))$. Since $\omega \in L(\mathfrak{A}^P_{r_u}(\mathcal{T}))$, we have $[\![\delta(r_u, \omega)]\!][\epsilon] = +$ by definition. Moreover, $\omega \notin u^{-1}L(\mathfrak{A}^P(\mathcal{T}))$ implies $u\omega \notin L(\mathfrak{A}^P(\mathcal{T}))$ and thus $[\![\delta(\delta(Q_0, u), \omega)]\!][\epsilon] = -$.

We will now prove that such an ω cannot come from V or ΣV by showing that $\omega \notin (\Sigma \cup \{\epsilon\})V$.

Assume that $\omega = av$ with $a \in \Sigma \cup \{\epsilon\}, v \in V$. By Lemma 4, $r_{ua}[v] = [\![\delta(r_u, a)]\!][v] = [\![\delta(r_u, \omega)]\!][\epsilon] = +$. This contradicts compatibility, as $r_{ua}[v] = +$ implies that $uav = u\omega \in L(\mathfrak{A}^P(\mathcal{T}))$.

Now, let $\delta(Q_0, u)_{\mathrm{DNF}} = M_1 \vee M_2 \vee \cdots \vee M_k$ be the formula that is reached in the automaton after reading u. The fact $\omega \notin (\Sigma \cup \{\epsilon\})V$ implies that this formula agrees with r_u, i. e. $[\![M_1 \vee \cdots \vee M_k]\!] = r_u$.

Now let $\omega = a\tilde{\omega}$. From the construction of δ, we know that the row r_{ua} is not completely filled with $-$, since

$$[\![\delta(r_u, \omega)]\!][\epsilon] = [\![\delta(r_u, a\tilde{\omega})]\!][\epsilon] = [\![\delta(\delta(r_u, a), \tilde{\omega})]\!][\epsilon] = \left[\!\left[\delta(b^P(r_{ua}), \tilde{\omega})\right]\!\right][\epsilon] = -$$

would contradict compatibility. For every column $v \in V$ with $r_{ua}[v] = +$, consider all monomials M_i with $[\![M_i a]\!][v] = +$. There must be at least one monomial, because otherwise $uav \notin L(\mathfrak{A}^P(\mathcal{T}))$, which would contradict the compatibility of $\mathcal{T}$ and $\mathfrak{A}^P(\mathcal{T})$. We have $M^P(v) \sqsubset \delta(r_u, a)$ by the construction of $\delta(r_u, a) = b^P(r_{ua})$. For every row $r_{\tilde{u}} \sqsubset M_i$, we have $\delta(r_{\tilde{u}}, a) = b^P(r_{\tilde{u}a}) = \bigvee_{\tilde{v} \in V, r_{\tilde{u}a}[\tilde{v}]=+} M^P(\tilde{v})$. Hence, $M^P(v) \sqsubset \delta(r_{\tilde{u}a})$. Thus, $M^P(v) \sqsubset \delta(M_i, a)_{\mathrm{DNF}}$ and $M^P(v) \sqsubset \delta(M_1 \vee \cdots \vee M_k, a)_{\mathrm{DNF}}$.

So, for every monomial $M^P(v) \sqsubset \delta(r_u, a)$, we have $M^P(v) \sqsubset \delta(M_1 \vee \cdots \vee M_k, a)_{\mathrm{DNF}}$ and thus $M^P(v) \sqsubset \delta(Q_0, u)_{\mathrm{DNF}}$. Hence, if $[\![\delta(r_u, a\tilde{\omega})]\!][\epsilon] = +$, this directly implies $[\![\delta(M_1 \vee \cdots \vee M_k, a\tilde{\omega})]\!][\epsilon] = +$. But $[\![\delta(M_1 \vee \cdots \vee M_k, a\tilde{\omega})]\!][\epsilon] = [\![\delta(\delta(Q_0, u), \omega)]\!][\epsilon] = -$. Hence, this is a contradiction and no such ω exists. □

For NFAs and UFAs, the reverse inclusion between the two languages in the statement of Lemma 5 holds in the case of compatibility, too. Angluin, Eisenstat, and Fisman (2015) have conjectured that this is also the case for AFAs since extensive tests of their algorithm AL* never gave a non-residual AFA. With the help of specially developed software that simulates and visualizes the run of AL* interactively, we have been able to construct a counterexample.

Lemma 6. *There exists a regular language L for which the algorithm AL* constructs a table $\mathcal{T}$ defining a compatible AFA $\mathfrak{A}^P(\mathcal{T})$, with $L(\mathfrak{A}^P(\mathcal{T})) = L$, such that for some $r \in P$ and all $\omega \in \Sigma^*$ the residual language $\omega^{-1}L$ is not contained in $L(\mathfrak{A}^P_r(\mathcal{T}))$.*

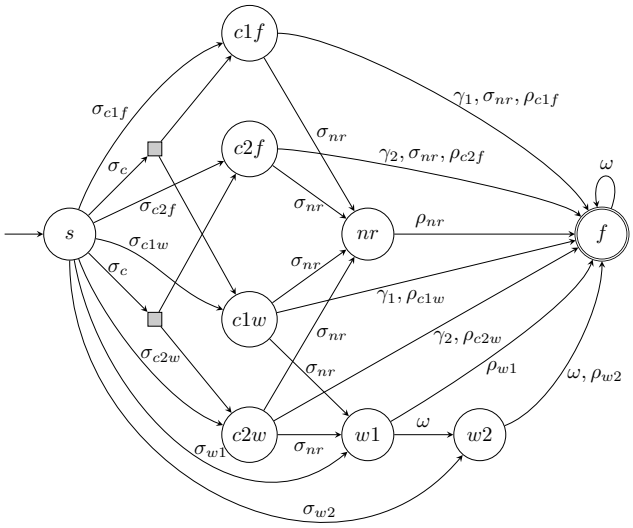

Figure 2: A non-residual AFA constructed by AL*. The initial configuration is $Q_0 = s$ and the set of accepting states $F = \{f\}$. A filled square indicates a conjunction of its successors.

Proof. It can be shown that the AFA in Fig. 2 is compatible to a table $\mathcal{T}$ that can be constructed by AL* on a carefully designed language L. The state labeled nr is not residual. □

4.2 Learning Residual Alternating Automata

Let L be a given regular language. In order to construct only residual AFAs for L we build on AL* and design a new algorithm AL** presented as Algorithm 1 that solves this problem. The main obstacle that one encounters is the test of residuality of the constructed automaton. We use the power of the equivalence-oracle to incorporate this task into AL* by reducing it to a single equivalence query of a larger automaton.

Let $\mathrm{Suffs}(w)$ denote the set of all suffixes of a string w. We start the analysis of AL** with the following observation which guarantees that the automata constructed successively from tables $\mathcal{T}$ are well defined.

Lemma 7. *In* AL** *algorithm table* $\mathcal{T}$ *is always consistent.*

The main difference between AL* and AL** lies in the construction of the automaton $\mathfrak{A}^{P'}(\mathcal{T})$ in line 12. This modification of AL* allows us to guarantee the residuality of the generated automaton. As shown in the previous section, the reason for the possible non-residuality of the automaton produced by AL* is that the reverse statement of Lemma 5 does not hold for AFAs. As we perform no basis reduction at the construction of $\mathfrak{A}^{P'}(\mathcal{T})$, compatibility of the table and the automaton guarantees residuality of the automaton.

Lemma 8. *If the AFA* $\mathfrak{A}^{P'}(\mathcal{T})$ *constructed in line 12 is compatible with* $\mathcal{T}$ *then* $\mathfrak{A}^{P'}(\mathcal{T})$ *is residual.*

Proof. Consider some $u \in U$. As $P' = \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, we have $r_u \in P'$ and thus $L(\mathfrak{A}^{P'}_{r_u}(\mathcal{T})) \subseteq u^{-1}L(\mathfrak{A}^{P'}(\mathcal{T}))$ by

```
1  U ← {ϵ}; V ← {ϵ};
2  initialize T = (T, U, V) with |Σ| + 1 membership queries;
3  while true do
4      P ← Rows_high(T);
5      while T is not P-closed do
6          find a row r_ua ∈ Rows(T) with r_ua ∉ [[F(P)]];
7          add ua to U;
8          complete T via membership queries;
9          P ← Rows_high(T);
10     construct a minimal basis P and A^P(T) for P;
11     if L(A^P(T)) = L then
12         construct A^P'(T) with P' = Rows_high(T);
13         if L(A^P'(T)) = L then
14             return A^P(T);
15         else
16             get counterexample w ∈ LΔL(A^P'(T));
17             set V ← V ∪ Suffs(w);
18             complete T via membership queries;
19     else
20         get counterexample w ∈ LΔL(A^P(T));
21         set V ← V ∪ Suffs(w);
22         complete T via membership queries;
```

Algorithm 1: AL** for the target language L.

Lemma 5. It remains to prove the inclusion in the other direction. Iterating over the length of u one can show that for every configuration of the AFA $\delta(Q_0, u) \equiv r_u \wedge R_u$, where R_u is some expression.

By construction, every monomial of $Q_0 = b^P(r_\epsilon)$ contains r_ϵ. Therefore, $Q_0 \equiv r_\epsilon \wedge R_\epsilon$ for some expression R_ϵ. Hence, $\delta(Q_0, \epsilon) = Q_0 \equiv r_\epsilon \wedge R_\epsilon$.

As U is prefix-closed, every prefix of u is also in U. If $u = u'a$, every monomial of $\delta(u', a)$ contains $r_{u'a} = r_u \in P'$ by the induction hypothesis. Therefore, $\delta(u', a) \equiv r_u \wedge R'_u$, where R'_u is an expression. We thus have $\delta(Q_0, u) = \delta(\delta(Q_0, u'), a) \equiv \delta(r_{u'} \wedge R_{u'}, a) \equiv (r_u \wedge R'_u) \wedge R_{u'} \equiv r_u \wedge R_u$ for some expression R_u.

Therefore, $L(\mathfrak{A}^{P'}_{r_u}(\mathcal{T})) \supseteq u^{-1}L(\mathfrak{A}^{P'}(\mathcal{T}))$. □

Computing the large residual automaton $\mathfrak{A}^{P'}(\mathcal{T})$ in line 12 upon the trivial basis P' allows us to test the smaller automaton $\mathfrak{A}^{P}(\mathcal{T})$ for residuality via the following lemma. If $\mathfrak{A}^{P'}(\mathcal{T})$ passes the equivalency test, it certificates the residuality of $\mathfrak{A}^{P}(\mathcal{T})$. Otherwise, our construction directly gives us a counter-example that helps $\mathfrak{A}^{P'}(\mathcal{T})$ to pass the equivalence test the next time.

Lemma 9. *If the two AFAs* $\mathfrak{A}^{P}(\mathcal{T})$ *and* $\mathfrak{A}^{P'}(\mathcal{T})$ *constructed in line 10, resp. 12 satisfy* $L(\mathfrak{A}^{P}(\mathcal{T})) = L = L(\mathfrak{A}^{P'}(\mathcal{T}))$ *then* $\mathfrak{A}^{P}(\mathcal{T})$ *is residual.*

Proof. Assume $L(\mathfrak{A}^{P}(\mathcal{T})) = L = L(\mathfrak{A}^{P'}(\mathcal{T}))$. Lemma 8 states that $\mathfrak{A}^{P'}(\mathcal{T})$ is residual. Consider a state $q = r_u$ of $\mathfrak{A}^{P}(\mathcal{T})$ with corresponding state q' of $\mathfrak{A}^{P'}(\mathcal{T})$. As $r_u \in P \subseteq \mathrm{Rows}_{\mathrm{high}}(\mathcal{T}) = P'$, there is always such a corresponding state. Let $a \in \Sigma$ be any alphabet symbol. For every monomial $M' \sqsubset \delta(q', a)$, there is a mono-

mial $M \sqsubset \delta(q,a)$ such that every literal of M is in M' (with the corresponding v we have $M = M^P(v)$ and $M' = M^{P'}(v)$ and $M^{P'}(v)$ may consist of states not in P). Hence we have $[\![\delta(q,w)]\!] \geq [\![\delta(q',w)]\!]$. We know $u^{-1}L = u^{-1}L(\mathfrak{A}^{P'}(\mathcal{T})) \subseteq L(\mathfrak{A}^{P'}_{q'}(\mathcal{T}))$ from Lemma 8. We also know $L(\mathfrak{A}^P_q(\mathcal{T})) \subseteq u^{-1}L(\mathfrak{A}^P(\mathcal{T})) = u^{-1}L$ from Lemma 5. So we have $u^{-1}L \subseteq L(\mathfrak{A}^{P'}_{q'}(\mathcal{T})) \subseteq L(\mathfrak{A}^P_q(\mathcal{T})) \subseteq u^{-1}L$. Thus, $u^{-1}L = u^{-1}L(\mathfrak{A}^P(\mathcal{T})) = L(\mathfrak{A}^P_q(\mathcal{T}))$. Thus, automaton $\mathfrak{A}^P(\mathcal{T})$ is also residual. □

For a language L the reverse of L contains all strings $a_1 \dots a_\ell \in \Sigma^*$ such that $a_\ell \dots a_1$ is in L. Now we are ready to give the main result of this section.

Theorem 2. *For any given regular language L, the algorithm* AL** *always generates an RAFA $\mathfrak{A}^P$ such that $L(\mathfrak{A}^P) = L$. Moreover, if the basis P constructed in the run of* AL** *is of minimal size, $\mathfrak{A}^P$ has the minimal number of states over all RAFAs for L. The algorithm terminates after at most κ_L equivalence queries and $\kappa_L\hat{\kappa}_L(1+|\Sigma|)\ell$ membership queries, where κ_L and $\hat{\kappa}_L$ denote the number of states of the minimal DFA for L, resp. the reverse of L and ℓ is the size of the longest counterexample obtained from the equivalence oracle.*

4.3 Approximating the Minimum Basis

Assume $\mathcal{T} = (T,U,V)$ is a table for a regular language. Note that algorithm AL** constructs a minimal basis P (of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$) because computing a minimum basis (i. e. of minimal cardinality) is $\mathcal{NP}$-hard, as shown by Angluin, Eisenstat, and Fisman (2015). In order to guarantee that the basis (and hence the set of states) used by the algorithm is small enough, we give an approximation algorithm for this problem. In the optimization problem MIN-SET-COVER, one is given a groundset $\mathcal{X}$ and a set $\mathcal{S}$ of subsets of $\mathcal{X}$ and searches the smallest $S \subseteq \mathcal{S}$ with $\bigcup_{s\in S} s = \mathcal{X}$ (see e. g. (Williamson and Shmoys 2011)). If $\mathcal{M}^P := \{[\![M^P(v)]\!] \mid v \in V\}$ for $P \subseteq \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, we obtain the following lemma.

Lemma 10. *For any P it is true that $\mathcal{M}^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})} = \mathcal{M}^P$, iff P is a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*

We will now reduce the problem of finding a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ to the problem of finding a solution to a SET-COVER instance.

Lemma 11. *Let $\mathcal{X} = \{(v,i) \mid v,i \in V \wedge [\![M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v)]\!][i] = -\}$ be the groundset and $\mathcal{S} = \{m_u \mid u \in U\}$ with subsets $m_u = \{(v,i) \in \mathcal{X} \mid r_u \geq [\![M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v)]\!]$ and $r_u[i] = -\}$ be an instance of* SET-COVER. *The set P is a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, iff there exists a feasible solution $\mathcal{C}$ of the set cover instance above such that $P = \{r_u \mid m_u \in \mathcal{C}\}$.*

Proof. Every vector of $\mathcal{M}^P$ can be composed by the vectors of P by intersection, so requiring these compositions does not increase P. Now we apply the lemma above. □

We can now use the well known algorithm for the optimization problem MIN-SET-COVER due to (Johnson 1974) that, on input $(\mathcal{X},\mathcal{S})$ produces a feasible solution $S \subseteq \mathcal{S}$ with $|S| \leq (\ln(|\mathcal{X}|)+1)|S^*|$ in polynomial time, where S^* is an optimal solution to the instance. We get the following result.

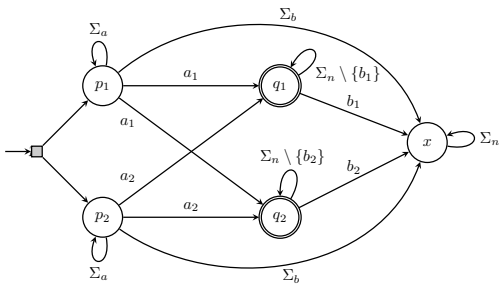

Figure 3: The residual AFA for the language A_n of Theorem 4 with $n = 2$. The corresponding alphabet is $\Sigma_n = \Sigma_a \cup \Sigma_b$ with $\Sigma_a = \{a_1, a_2\}$ and $\Sigma_b = \{b_1, b_2\}$, the initial configuration is $Q_0 = p_1 \wedge p_2$, and the set of accepting states is $F = \{q_1, q_2\}$.

Theorem 3. *There exists a polynomial time algorithm that for a given table $\mathcal{T} = (T,U,V)$ returns a basis P of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ with $|P| \leq (2\ln(|V|)+1)\cdot|P^*|$, where P^* is a minimum basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*

5 On the Size of Residual AFAs

In (Angluin, Eisenstat, and Fisman 2015) it has been shown that RAFAs may be exponentially more succinct than RNFAs and RUFAs and double exponentially more succinct than DFAs. We strengthen these results by proving that RAFAs may be exponentially more succinct than every equivalent *non-residual* NFAs or UFAs. Furthermore, there exists a RAFA that is double exponentially more succinct than the minimal DFA and uses only 2 non-deterministic (i. e. $\vee$) transitions and only a linear number of universal (i. e. $\wedge$) transitions. Thus, the restriction to residual automata still allows a very compact representation. On the other hand, we give an example where the residuality of an automata demands an exponentially larger state set.

Theorem 4. *For every even $n \in \mathbb{N}$, there exists a language A_n that can be accepted by a* residual *AFA with $2n+1$ states and every NFA or UFA for A_n needs at least $\binom{n}{n/2}$ states.*

Proof Sketch. The alphabet Σ_n for A_n consists of disjoint subsets $\Sigma_a = \{a_1, a_2, \dots, a_n\}$ and $\Sigma_b = \{b_1, b_2, \dots, b_n\}$.

$$A_n = \{w_1 w_2 \mid w_1 \in \Sigma_a^*, w_2 \in \Sigma_n^*, \\ w_1 \textit{ contains all } \text{symbols from } \Sigma_a, \\ w_2 \text{ does } \textit{not contain all } \text{symbols from } \Sigma_b\}.$$

We construct a residual AFA with states $\{p_1, \dots, p_n, q_1, \dots, q_n, x\}$ that is sketched for $n = 2$ in Fig. 3. To prove the second property we show that every NFA has to "remember" all $\binom{n}{n/2}$ subsets of size $n/2$ of Σ_a, while UFAs need to "remember" all $\binom{n}{n/2}$ subsets of size $n/2$ of Σ_b. □

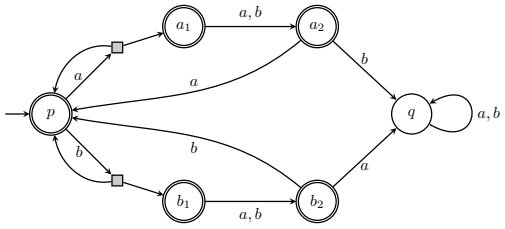

Figure 4: The (non-residual) AFA for the language B_n of Theorem 5 with $n = 2$.

Theorem 5. *For every $n \in \mathbb{N}$ there exists a language B_n over a binary alphabet that can be accepted by a (non-residual) AFA with $2n + 2$ states, but every residual AFA for B_n requires at least 2^n states.*

One can construct the succinct AFAs of Theorem 5 as follows. Let $\Sigma = \{a, b\}$ and consider $B_n = \{w^* w' \mid w \in \Sigma^n, w' \text{ is a prefix of } w\}$ (based on the construction of (Vardi 1995)). For $n = 2$, the non-residual AFA $\mathfrak{A} = (Q, Q_0, \delta, F)$ for B_n is sketched in Fig. 4.

A closer look at the constructions of succinct automata for B_n reveals that the resulting AFAs are in fact UFAs. Dually, $\overline{B}_n = \Sigma^* \setminus B_n$ can be accepted by an NFA with the same number of states $2n + 2$. Thus, we obtain families of languages B_n and $\overline{B}_n$, $n = 1, 2, \ldots$, such that every residual AFA for B_n, resp. $\overline{B}_n$, is exponentially larger than the corresponding minimal UFA, resp. NFA.

As it was already noted in (Angluin, Eisenstat, and Fisman 2015), RAFAs may be double exponentially smaller than the minimal DFAs. We give a more precise bound inspired by a language defined by Chandra, Kozen, and Stockmeyer (1981).

Theorem 6. *For every $n \in \mathbb{N}$ there exists a language C_n such that the minimal DFA for C_n needs at least 2^{2^n} states and there is a* residual *AFA with $2n^2 + 5n$ states for C_n.*

The tables below summarize the results presented in this section. Here

$\mathfrak{A}_1$	$\mathfrak{A}_2$
$k_1(n)$	$k_2(n)$

has the following meaning: For every n there exists a language L_n with $k_1(n)$ state automata of type $\mathfrak{A}_1$ for L_n and every automaton of type $\mathfrak{A}_2$ for L_n needs at least $k_2(n)$ states.

RAFA	NFA/UFA
$2n+1$	$\binom{n}{n/2}$

NFA/UFA	RAFA
$2n+2$	2^n

RAFA	DFA
$2n^2+5n$	2^{2^n}

6 Discussion

We have disproved the conjecture that the algorithm AL* outputs residual AFAs only and designed a modified algorithm AL** that achieves this property. This algorithm has almost the same complexity as AL*. In fact, as all automata produced by AL* in the experiments in (Angluin, Eisenstat, and Fisman 2015) were residual, we expect that our new algorithm AL** only performs a single additional equivalence-query to verify the residuality compared to AL*. Thus, based on the performance experiments reported for randomly generated automata or regular expressions AL** outperforms the algorithms L* and NL* w. r. t. the number of membership queries. Simultaneously AL** infers an (approximately minimal) RAFA which is always smaller than (or equal to) the corresponding minimal DFA generated by L* and RNFA produced by NL*. Typically, AL** generates automata which are significantly more succinct than DFAs and RNFAs. Theoretical analysis shows that residual AFAs can be exponentially smaller than NFAs and even double exponentially more succinct than DFAs. This makes RAFAs an attractive choice for language representations in the design of learning algorithms.

While residual non-deterministic automata have been understood quite well (Denis, Lemay, and Terlutte 2001; 2004; Bollig et al. 2009; Kasprzik 2010), fundamental questions concerning residual alternating automata remain open. In our paper we introduced a complementary notion to the canonical RNFA – the canonical RUFA. Recently, we have exhibited languages for which the canonical RNFA and RUFA differ, but both automata are minimal AFAs. Thus, a meaningful notion for canonical AFAs would be desirable, but this seems to be a difficult problem, which we leave for future work.

References

Angluin, D.; Eisenstat, S.; and Fisman, D. 2015. Learning regular languages via alternating automata. In *Proc. 24. IJCAI*, 3308–3314.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75(2):87 – 106.

Bollig, B.; Habermehl, P.; Kern, C.; and Leucker, M. 2009. Angluin-style learning of NFA. In *Proc. 21. IJCAI*, 1004–1009.

Chandra, A. K.; Kozen, D. C.; and Stockmeyer, L. J. 1981. Alternation. *J. ACM* 28(1):114–133.

De la Higuera, C. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Denis, F.; Lemay, A.; and Terlutte, A. 2001. Residual finite state automata. In *Proc. 18. STACS*, LNCS 2010, 144–157. Springer.

Denis, F.; Lemay, A.; and Terlutte, A. 2004. Learning regular languages using RFSAs. *Theoretical Computer Science* 313(2):267–294.

Johnson, D. S. 1974. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.* 9(3):256–278.

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

Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, 238–266. Springer.

Williamson, D. P., and Shmoys, D. B. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.