Electronic Colloquium on Computational Complexity, Report No. 46 (2017)

Learning Residual Alternating Automata

Sebastian Berndt, Maciej Liśkiewicz*, Matthias Lutter, and Rüdiger Reischuk

Institut für Theoretische Informatik, Universität zu Lübeck, Germany
{berndt,liskiewi,lutter,reischuk}@tcs.uni-luebeck.de

Abstract. Residuality plays an essential role for learning finite automata. While residual deterministic and nondeterministic automata have been understood quite well, fundamental questions concerning alternating automata (AFA) remain open. Recently, Angluin, Eisenstat, and Fisman [3] 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 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 navigation, automated verification, and many others (see e. g. [10], the textbooks [11] and [19], and the references therein). Depending on applications, different types of automata might be selected as desirable learning targets. The list goes from deterministic ones (DFA) over nondeterministic ones (NFA), alternatively the dual of NFAs – universal finite automata (UFA), up to their common generalization – the alternating finite automata (AFA). Though these types all have the same expressive power, they turn out to be different w. r. t. modeling capabilities and succinctness properties. A *minimal* (measured by 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 formal 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 [6].

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 by Angluin in [2] and launched a tremendous amount of research yielding many effective algorithms relevant in machine learning and other areas.

Angluin provided in [2] an algorithm, named L* that based on membership and equivalence queries 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 DFA is minimal.

Beside uniqueness, minimal DFAs have 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.

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

ISSN 1433-8092

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, nondeterministic automata in general do not satisfy the residuality property. Even worse, languages accepted by $\mathfrak{A}_q$ for states q of an NFA $\mathfrak{A}$, have no natural interpretation. Furthermore, minimal NFAs don't have to be 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 formal verification. 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 [12] introduced the class of residual NFA (RNFA), which are perfectly suited for learning algorithms. 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, Habermehl, Kern and Leucker [4] proposed a sophisticated extension of Angluin's algorithm named NL* that learns a canonical RNFA with a polynomial number of membership and equivalence queries. Analogously to RNFA, Kern gave a definition of residual universal automata (RUFA) and their canonical form [20]. Based on this he further proposed an algorithm UL* – a remodeling of NL* – for learning canonical RUFAs.

Since NL* and UL* may infer more succinct residual automata than L* ([12,20]) they were successfully applied to several studies, using e. g. their implementations in the `libalf` learning library [5]. For example, these methods are attractive in the area of formal verification including model checking [1,8,9], where the size of the models of interest is of crucial importance and nondeterminism is a natural abstraction concept. In this area, among others this approach has been were used for the compositional verification of probabilistic systems [14,15] and verification and model synthesis of sequential programs [7]. As verification concerns tasks of alternating nature involving existential and universal statements, investigations of (residual) alternating automata seems to be a natural objective for systematic research.

Recently, Angluin, Eisenstat, and Fisman [3] extended the definition of residual automata to alternating automata 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.

The paper is organized as follows. In Section 2 we provide some backgrounds on automata, learning algorithms and fix notation used in the paper. In Section 3 we describe the UL* algorithm for learning residual UFAs. Next, in Section 4 we present the algorithm AL** and its analysis. Section 5 contains new results on the size of residual AFAs. We finish this paper with a discussion and some conclusions. For sake of readability, we postpone some of the proofs and technical details to the appendix.

2 Preliminaries

Let the symmetric difference of sets be denoted by $\triangle$, the set of all suffixes of a string w denoted by $\mathrm{Suffs}(w)$, and the Boolean values "true" as $\top$ and "false" as $\bot$. For a set S let $\mathcal{F}(S)$ be the set of all formulas over S using the binary operators $\wedge$ and $\vee$ plus the trivial formulas $\top$ and $\bot$ that are always, resp. never satisfied. The restriction $\mathcal{F}_\vee(S)$, resp. $\mathcal{F}_\wedge(S)$ denotes the subset of formulas containing only the $\vee$ operator plus the formula $\bot$ (resp. only $\wedge$ and $\top$).

2.1 Automata

The computational model of alternating finite automata has been introduced by Chandra, Kozen, and Stockmeyer [6].

Definition 1. *Given a finite alphabet Σ, an* alternating finite automaton (AFA) *is a four-tuple (Q, Q_0, F, δ), where Q is the* set of states, *$Q_0 \in \mathcal{F}(Q)$ the* initial configuration, *$F \subseteq Q$ the subset of* accepting states, *and $\delta\colon Q \times \Sigma \to \mathcal{F}(Q)$ 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 then the automaton is called deterministic (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 models a* nondeterministic automaton (NFA). *E. g., if $\delta(q, a) = p_1 \vee p_2$, this describes a nondeterministic 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 automaton is called universal (UFA). *A transition $\delta(q, a) = p_1 \wedge p_2$, for example, leads to state p_1 and state p_2 simultaneously.*

A transition $\delta(q, a)$ of an AFA can be a nested formula of $\vee$ and $\wedge$ operators. Such a formula is difficult to draw pictorially. However, any such formula can equivalently be represented by its disjunctive normal form (DNF) that does not contain any negated variables. Each monomial in such a DNF is represented by an edge from q marked with the letter a leading to a little square. From this square we draw edges to all states that are contained in this monomial. If the monomial consists of a single state only the square can be dropped. For an example, see the AFA in Fig. 1.

The function δ is extended to arbitrary formulas $\varphi \in \mathcal{F}(Q)$ and strings $w \in \Sigma^*$. Let $\varphi_{\mathrm{DNF}} = \bigvee_i M_i$ with $M_i = \bigwedge_j q_{i,j}$ be a DNF-formula equivalent to φ. Then $\delta(\varphi, a) := \bigvee_i \bigwedge_j \delta(q_{i,j}, a)$ for a single symbol $a \in \Sigma$ and $\delta(\varphi, \epsilon) := \varphi$ for the empty string ϵ. For $w \in \Sigma^+$, we define $\delta(\varphi, wa) := \delta(\delta(\varphi, w), a)$. For an NFA, this simply reduces to $\delta(q \vee p, a) = \delta(q, a) \vee \delta(q, b)$.

Definition 2. *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: $(\!|\top|\!) := \top$ and $(\!|\bot|\!) := \bot$. For singletons let $(\!|q|\!) := \top$ if $q \in F$ and equal $\bot$ otherwise. Finally, $(\!|\varphi R \psi|\!) := (\!|\varphi|\!)\ R\ (\!|\psi|\!)$ for $R \in \{\wedge, \vee\}$.*
The automaton $\mathfrak{A}$ accepts *a word w, if $(\!|\delta(Q_0, w)|\!) = \top$. The* language *$L(\mathfrak{A})$ is the set of all accepted strings. For a state $q \in Q$, we write $\mathfrak{A}_q$ for the automaton (Q, q, F, δ) that starts in configuration q instead of Q_0.*

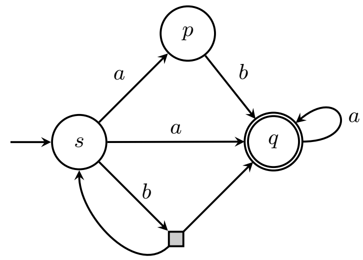

Fig. 1. An AFA for the language $L_1 = a^+ \cup ba^+ \cup aba^*$.
The initial configuration is $Q_0 = s$ and the set of accepting states is $F = \{q\}$.
From state s the automaton has the transitions $\delta(s, a) = p \vee q$ and $\delta(s, b) = s \wedge q$.

For an NFA with $\delta(Q_0, w) = q_1 \vee \ldots \vee q_k$ the evaluation $(\!|\delta(Q_0, w)|\!) = \top$ corresponds to the usual condition $\{q_1, \ldots, q_k\} \cap F \neq \emptyset$, i. e. when starting with initial configuration Q_0 and reading the word w some accepting state is reached. For a UFA with $\delta(Q_0, w) = q_1 \wedge \ldots \wedge q_k$ it requires $\{q_1, \ldots, q_k\} \subseteq F$, i. e. all states reached are accepting.

2.2 Residuality

Definition 3. *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 language 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 set of all $\cup$-prime, resp. $\cap$-prime residuals of L by $\cup$-Primes(L), resp. $\cap$-Primes(L).*
- *An automaton $\mathfrak{A}$ with states Q is* residual, *if $L(\mathfrak{A}_q) \in \mathrm{RES}(L(\mathfrak{A}))$ for all $q \in Q$, i. e. if every state corresponds to a prefix u and its residual language $u^{-1}L(\mathfrak{A})$.*
- *Let RNFA, RUFA and RAFA denote the appropriate residual restrictions.*

For an example, see the residual AFA in Fig. 2 that accepts the same language $L_1 = a^+ \cup ba^+ \cup aba^*$ as the nonresidual AFA illustrated in Fig. 1

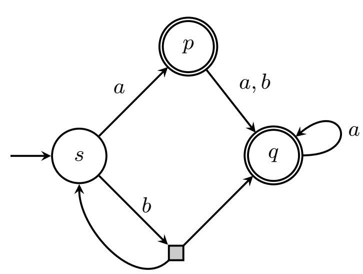

Fig. 2. A residual AFA (RAFA) for the language $L_1 = a^+ \cup ba^+ \cup aba^*$.
State s corresponds to $\varepsilon^{-1}L = a^+ \cup ba^+ \cup aba^*$, state p to $a^{-1}L = a^* \cup ba^*$, and state q to $(ab)^{-1}L = a^*$.
Note that these residual languages $\varepsilon^{-1}L$, $a^{-1}L$ and $(ab)^{-1}L$ are both $\cup$-prime and $\cap$-prime.

2.3 Learning Algorithms

All learning algorithms XL* for automata (i. e. L*, NL*, UL*, and AL*) and the new AL** follow a 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 P of rows (a *basis*) is constructed such that all rows can be built from the elements of P. The specific way to "build" a row depends on the type of automaton. A hypothesized automaton is constructed from this subset P. 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. For this purpose, similar to [4] we introduce the following notion.

Definition 4. *Let L be a regular language. For a prefix-closed set U and a suffix-closed set V, a $|U \cup U\Sigma| \times |V|$* table *$\mathcal{T} = (T, U, V)$ for L with entries in $\{+, -\}$ is determined by a function $T\colon \Sigma^* \to \{+, -, \bot\}$ specified as follows. Let $W(\mathcal{T})$ denote the set $(U \cup U\Sigma)\ V$ described by $\mathcal{T}$. Then for $w \in \Sigma^*$*

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

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. An example of a table is given in Fig. 3.

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

Fig. 3. 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 equals $T(xy)$. For example, the value in row ab and column b is $+$ since $T(abb) = +$ ($abb \in L$) and $abb \in W(\mathcal{T})$. An example for a row is $r_\epsilon = (- + -)$. Furthermore, $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T}) = \{r_\epsilon, r_a\}$.

Definition 5.

- *An automaton $\mathfrak{A}$ and a table $\mathcal{T} = (T, U, V)$ are called* compatible *if for every $w \in W(\mathcal{T})$ holds: $\mathfrak{A}$ 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 $v \in V$, called the* row *of u. The set of all rows is denoted by $\mathrm{Rows}(\mathcal{T})$ and the subset of those r_u with $u \in U$ by $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*
- *A table $\mathcal{T}$ is* consistent *if for every $u, u' \in U$ with $r_u = r_{u'}$ the condition $r_{ua} = r_{u'a}$ is fulfilled for every $a \in \Sigma$.*[1]

[1] This is a weaker requirement than the *RFSA-consistency* of [4], which requires that $r_u \leq r_{u'}$ implies $r_{ua} \leq r_{u'}a$.

– *To simplify the notation, for a consistent table $\mathcal{T}$, row $r \in \mathrm{Rows_{high}}(\mathcal{T})$, and symbol $a \in \Sigma$, let ra denote the vector r_{ua} where $u \in U$ is an arbitrary string with $r_u = r$.*

3 Learning Residual Universal Automata

This section serves two goals. First, restricting our view on universal transitions helps as a warm-up for the general case of AFAs that we will encounter later on. As universal automata are less familiar than their nondeterministic counterparts, we will use them in accustomizing to the setting of universal transitions. Secondly – as already mentioned – we use a weaker form of *consistency* than e. g. [4] or [20]. In [3] it was mentioned that this weaker form is sufficient for NL*, UL* and AL*, but no formal justification has been given yet. Hence, we will use this section to present UL* with the weaker consistency notion and give a detailed analysis of the algorithm. The original UL* that uses the stronger consistency notion was presented in [20].

In order to simplify the notation we use the following convention on formulas over states Q of a residual automaton: to every state $q \in Q$ a language L_q is associated and then the set $\{L_{q_1}, \ldots, L_{q_t}\}$ represents the formula $q_1 \wedge \ldots \wedge q_t$. The following definition helps to conclude that UL* always learns a unique minimal RUFA.

Definition 6 (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 $\delta(L_1, a) = \{L_2 \in Q \mid a^{-1}L_1 \subseteq L_2\}$.*

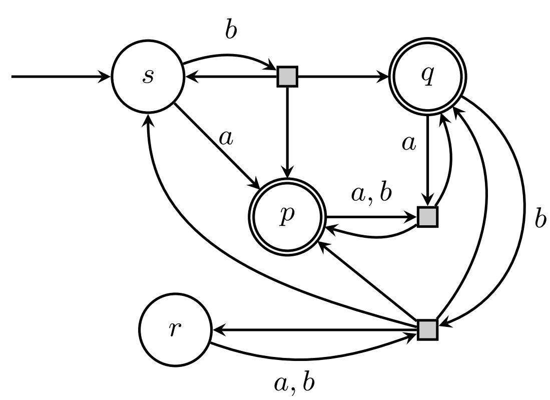

Fig. 4. The canonical RUFA for the language $L_1 = a^+ \cup ba^+ \cup aba^*$.

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.

Definition 7.

- *A row r_u of a table $\mathcal{T}$ is* ⊓-composite *if there are rows $r_{u_1}, \dots, r_{u_k} \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, with $r_{u_i} \neq r_u$, such that $r_u = \bigsqcap_{i=1}^{k} r_{u_i}$. Otherwise, r_u is called* ⊓-prime*. Let $\mathrm{Primes}_{\sqcap}(\mathcal{T})$ be the set of ⊓-prime rows in $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*
- *To simplify 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* ⊓-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 ⊓ operator. A subset of rows that can generate all rows of $\mathcal{T}$ using ⊓ is called a* ⊓-basis *for $\mathcal{T}$.*

For example, in Fig. 3, the row r_b of $\mathcal{T}$ is composite as $r_b = r_\epsilon \sqcap r_a$, whereas r_ϵ, r_a, r_{ab} are ⊓-prime and $\mathrm{Primes}_{\sqcap}(\mathcal{T}) = \{r_\epsilon, r_a\}$.

Thus, $\mathcal{T}$ is ⊓-closed if $\mathrm{Primes}_{\sqcap}(\mathcal{T})$ is a ⊓-basis for $\mathcal{T}$. The table in Fig. 3 is not ⊓-closed as the row $r_{ab} \in \mathrm{Rows}(\mathcal{T})$ is not composable by rows of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.

For a consistent and ⊓-closed table $\mathcal{T}$, define the UFA $\mathfrak{A}(\mathcal{T}) = (Q, Q_0, F, \delta)$ by $Q = \mathrm{Primes}_{\sqcap}(\mathcal{T})$, $Q_0 = \mathbb{B}_{\sqcap}(r_\epsilon) \cap Q$ and $F = \{r \in Q \mid r[\epsilon] = +\}$. For $r \in Q$ and $a \in \Sigma$ let $\delta(r, a) = \mathbb{B}_{\sqcap}(ra) \cap Q$. The following propositions show that $\mathfrak{A}(\mathcal{T})$ is the canonical RUFA.

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. $r_{u'} \leq r_u \iff \forall w\ \delta(r_u, w) \not\subseteq F \Rightarrow \delta(r_{u'}, w) \not\subseteq F$.

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 $r \in Q$ and $u_r \in U$ s. t. $r_{u_r} = r$. 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 ∩-prime since r is ⊓-prime due to Lemma 1. □

$U \leftarrow \{\epsilon\}$; $V \leftarrow \{\epsilon\}$; initialize $\mathcal{T} = (T, U, V)$ with $|\Sigma| + 1$ membership queries;
while *true* **do**
  while $\mathcal{T}$ *is not ⊓-closed* **do**
    find a row $r_{ua} \in \mathrm{Rows}(\mathcal{T})$ s. t. r_{ua} cannot be generated from $\mathrm{Primes}_{\sqcap}(\mathcal{T})$;
    add ua to U; complete $\mathcal{T}$ via membership queries;
  construct the UFA $\mathfrak{A}(\mathcal{T})$;
  if $L(\mathfrak{A}(\mathcal{T})) = L$ **then**
    return $\mathfrak{A}(\mathcal{T})$;
  else
    get a counterexample $w \in L \triangle L(\mathfrak{A}(\mathcal{T}))$; set $V \leftarrow V \cup \mathrm{Suffs}(w)$;
    complete $\mathcal{T}$ via membership queries;

Algorithm 1: UL* applied to a regular language $L \subseteq \Sigma^*$.

In order to learn the canonical RUFA, the learning algorithm UL* presented as Algorithm 1 only needs to construct a suitable table $\mathcal{T}$. The consistency of $\mathcal{T}$ follows from the fact that no duplicate rows are present in $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$. See Lemma 7 for a formal proof of this in the setting of alternating automata.

4 Learning Alternating Automata

This section presents our main result. In [3], an algorithm AL* was presented to learn alternating automata and its running time was analyzed. However, properties of the automata produced remained unclear. We close this gap by establishing several properties of AL* and then disprove the conjecture about residuality. Next, we present a modified algorithm AL** that guarantees residuality. Finally, we discuss how to find a provably good basis for AFAs (defined in the next subsection) and present experimental results demonstrating the performance of AL**.

4.1 Analysis of AL*

Let us review the construction of the automata generated by AL* and analyze the properties of these automata in detail. We use the basic version of AL* (Algorithm 1, in [3]) without further optimizations described there later.

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

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

Definition 8. *In the following P will always denote a subset of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*

- *The set P is a $(\sqcup, \sqcap)$-*basis *for $\mathcal{T}$ (in the following simply called a basis) if $\mathrm{Rows}(\mathcal{T}) \subseteq [\![\mathcal{F}(P)]\!]$ and table $\mathcal{T}$ is then called P-*closed.
- *The table $\mathcal{T}$ is called P-*minimal *if P is a minimal basis for $\mathcal{T}$, i. e. for all $p \in P$, the set $P \setminus \{p\}$ 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 $\left[\!\left[M^P(v)\right]\!\right][v] = +$. If for all $p \in P$ we have $p[v] = -$ then $M^P(v) := \top$.
- *For $r \in \mathrm{Rows}(\mathcal{T})$ of a P-closed table $\mathcal{T}$ 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. If for all $v \in V$ we have $r[v] = -$ then $b^P(r) := \bot$.
- *For a monomial M and $a \in \Sigma$ we define Ma as the monomial derived from M by replacing every row $r \in P$ of M by ra.*

Note that $\llbracket b^P(r) \rrbracket = r$.

Definition 9. *Let φ be a DNF-formula consisting of monomials M_i. We use the notation $M_i \sqsubset \varphi$ and for a monomial $M_i = \bigwedge_j x_j$ 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 $\llbracket \varphi(r_1, r_2, \ldots, r_k) \rrbracket = \llbracket \psi(r_1, \ldots, r_k) \rrbracket$ for all vectors $r_1, \ldots, r_k$ of identical length.*

Now, all necessary tools have been defined to construct an AFA $\mathfrak{A}^P(\mathcal{T})$ from a table a $\mathcal{T}$.

Definition 10. *Let $\mathcal{T}$ be a consistent and P-closed table. The AFA $\mathfrak{A}^P(\mathcal{T}) = (Q, Q_0, F, \delta)$ consists of the following components: $Q = P$, $Q_0 = b^P(r_\epsilon)$ and $F = \{r \in P \mid r[\epsilon] = +\}$. For $r \in Q$ and $a \in \Sigma$ let $\delta(r, a) = b^P(ra)$.*

Recall, that according to our convention, the term ra in the last expression denotes the vector r_{ua} s. t. $u \in U$ is any string with $r_u = r$. Note, moreover, that $\delta(r, a) = b^P(ra)$ is always a DNF-formula.

	ϵ	a	ba
ϵ	$-$	$+$	$+$
a	$+$	$+$	$-$
b	$-$	$+$	$-$
aa	$+$	$+$	$-$
ab	$+$	$+$	$+$

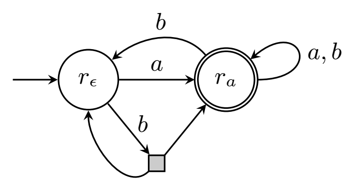

Fig. 5. A consistent and P-closed table $\mathcal{T}$ with $P = \{r_\epsilon, r_a\}$ and the corresponding ATM $\mathfrak{A}^P(\mathcal{T})$.

From this construction one can easily derive

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

Proof. We use induction upon the nesting of φ. For $r \in Q$ it holds $(\!|r|\!) = \top \Leftrightarrow r \in F \Leftrightarrow r[\epsilon] = \llbracket r \rrbracket [\epsilon] = +$. In the inductive step one can conclude

$$(\!|\psi \wedge \xi|\!) = \top \iff (\!|\psi|\!) = \top \wedge (\!|\xi|\!) = \top \iff \llbracket \psi \rrbracket [\epsilon] = + \wedge \llbracket \xi \rrbracket [\epsilon] = + \iff \llbracket \psi \wedge \xi \rrbracket [\epsilon] = + \,,$$
$$(\!|\psi \vee \xi|\!) = \top \iff (\!|\psi|\!) = \top \vee (\!|\xi|\!) = \top \iff \llbracket \psi \rrbracket [\epsilon] = + \vee \llbracket \xi \rrbracket [\epsilon] = + \iff \llbracket \psi \vee \xi \rrbracket [\epsilon] = + \,.$$
□

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. *For all $r \in P$ and $v \in V$ holds $r[v] = \llbracket \delta(r, v) \rrbracket [\epsilon]$.*

The technical proof of this claim is given in Appendix B.

Lemma 4. *For all $\varphi \in \mathcal{F}(P)$ and $v \in V$ we have $\llbracket \varphi \rrbracket [v] = \llbracket \delta(\varphi, v) \rrbracket [\epsilon]$.*

Proof. We may assume that φ is in DNF. If $[\![\varphi]\!]\,[v] = -$ then for every monomial $M \sqsubset \varphi$ it must hold $[\![M]\!]\,[v] = -$. Therefore, there exists some $r \sqsubset M$, such that $r[v] = -$. By Lemma 3, $[\![\delta(r,v)]\!]\,[\epsilon] = -$ and hence $[\![\delta(\varphi,v)]\!]\,[\epsilon] = -$.

Otherwise, if $[\![\varphi]\!]\,[v] = +$ there exists a monomial $M \sqsubset \varphi$ with $[\![M]\!]\,[v] = +$. Hence, for all $r \sqsubset M$ it must hold $r[v] = +$. Lemma 3 implies $[\![\delta(r,v)]\!]\,[\epsilon] = +$ and thus $[\![\delta(\varphi,v)]\!]\,[\epsilon] = +$. $\square$

Using these properties we continue the analysis as follows.

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 $L(\mathfrak{A}^P_{r_u}(\mathcal{T})) \nsubseteq u^{-1}L(\mathfrak{A}^P(\mathcal{T}))$, 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, $[\![\delta(r_u,a)]\!]\,[v] = [\![\delta(r_u,\omega)]\!]\,[\epsilon]$. Further, $[\![\delta(r_u,a)]\!] = r_{ua}$ by definition. Thus

$$r_{ua}[v] = [\![\delta(r_u,a)]\!]\,[v] = [\![\delta(r_u,\omega)]\!]\,[\epsilon] = +,$$

but this contradicts compatibility, as $r_{ua}[v] = +$ implies that $uav = u\omega \in L(\mathfrak{A}^P(\mathcal{T}))$.

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

$$[\![\delta(b^P(r_{ua}),\tilde{\omega})]\!]\,[\epsilon] \;=\; [\![\delta(b^P(-\cdots-),\tilde{\omega})]\!]\,[\epsilon] \;=\; [\![\delta(\bot,\tilde{\omega})]\!]\,[\epsilon] \;=\; [\![\bot]\!]\,[\epsilon] \;=\; -$$

would contradict

$$+ \;=\; [\![\delta(r_u,\omega)]\!]\,[\epsilon] \;=\; [\![\delta(r_u,a\tilde{\omega})]\!]\,[\epsilon] \;=\; [\![\delta(\delta(r_u,a),\tilde{\omega})]\!]\,[\epsilon] \;=\; [\![\delta(b^P(r_{ua}),\tilde{\omega})]\!]\,[\epsilon].$$

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. 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 $\mathfrak{A}^P(\mathcal{T})$. It holds $M^P(v) \sqsubset \delta(r_u,a)$ 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, $[\![\delta(r_u,a\tilde{\omega})]\!]\,[\epsilon] = +$ 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. $\square$

For NFAs and UFAs, the reverse inclusion between the two languages in the statement of Lemma 5 holds in the case of compatibility, too. In [3] it has been conjectured that this is also the case for AFAs since extensive tests of the algorithm AL* never produced 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* $\mathrm{AL}^\star$ *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 $L(\mathfrak{A}^P_r(\mathcal{T}))$.*

Proof. It can be shown that the AFA in Fig. 6 is compatible to a table $\mathcal{T}$ that can be constructed by $\mathrm{AL}^\star$ on a carefully designed language L (see Appendix A). The state labeled nr is not residual. □

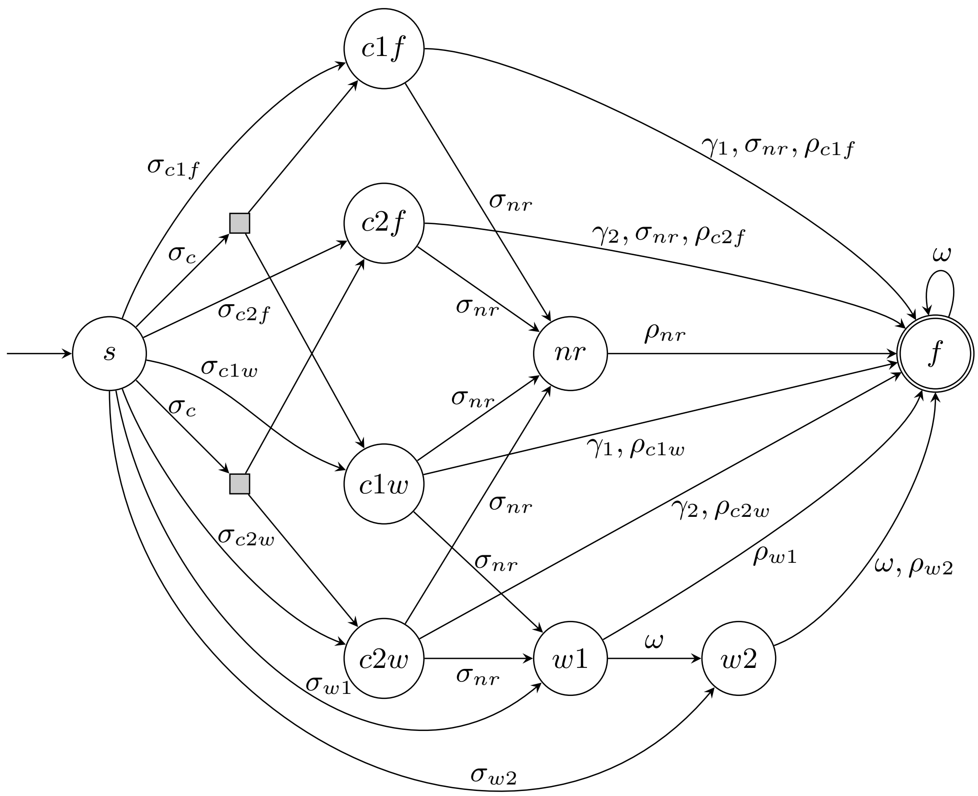

Fig. 6. A non-residual AFA constructed by $\mathrm{AL}^\star$ with initial configuration $Q_0 = s$ and accepting states $F = \{f\}$.

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 $\mathrm{AL}^\star$ and design a new algorithm $\mathrm{AL}^{\star\star}$ presented as Algorithm 2 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 $\mathrm{AL}^\star$ by reducing it to a single equivalence query of a larger automaton.

We start the analysis of $\mathrm{AL}^{\star\star}$ with the following observation which guarantees that the automata constructed successively from tables $\mathcal{T}$ are well defined.

Lemma 7. *Every table $\mathcal{T}$ constructed by $\mathrm{AL}^{\star\star}$ is consistent.*

1 $U \leftarrow \{\epsilon\}$; $V \leftarrow \{\epsilon\}$;
2 initialize $\mathcal{T} = (T, U, V)$ with $|\Sigma| + 1$ membership queries;
3 **while** *true* **do**
4 $\quad P \leftarrow \mathrm{Rows_{high}}(\mathcal{T})$;
5 $\quad$ **while** $\mathcal{T}$ *is not* P*-closed* **do**
6 $\quad\quad$ find a row $r_{ua} \in \mathrm{Rows}(\mathcal{T})$ with $r_{ua} \notin [\![\mathcal{F}(P)]\!]$;
7 $\quad\quad$ add ua to U;
8 $\quad\quad$ complete $\mathcal{T}$ via membership queries;
9 $\quad\quad P \leftarrow \mathrm{Rows_{high}}(\mathcal{T})$;
10 $\quad$ construct a minimal basis P and $\mathfrak{A}^P(\mathcal{T})$ for P;
11 $\quad$ **if** $L(\mathfrak{A}^P(\mathcal{T})) = L$ **then**
12 $\quad\quad$ construct $\mathfrak{A}^{P'}(\mathcal{T})$ with $P' = \mathrm{Rows_{high}}(\mathcal{T})$;
13 $\quad\quad$ **if** $L(\mathfrak{A}^{P'}(\mathcal{T})) = L$ **then**
14 $\quad\quad\quad$ return $\mathfrak{A}^P(\mathcal{T})$;
15 $\quad\quad$ **else**
16 $\quad\quad\quad$ get a counterexample $w \in L \triangle L(\mathfrak{A}^{P'}(\mathcal{T}))$;
17 $\quad\quad\quad$ set $V \leftarrow V \cup \mathrm{Suffs}(w)$;
18 $\quad\quad\quad$ complete $\mathcal{T}$ via membership queries;
19 $\quad$ **else**
20 $\quad\quad$ get a counterexample $w \in L \triangle L(\mathfrak{A}^P(\mathcal{T}))$;
21 $\quad\quad$ set $V \leftarrow V \cup \mathrm{Suffs}(w)$;
22 $\quad\quad$ complete $\mathcal{T}$ via membership queries;

Algorithm 2: AL** applied to a regular language $L \subseteq \Sigma^*$.

Proof. Let $\mathcal{T} = (T, U, V)$ be any table constructed by AL**. It suffices to show that for different $u, u' \in U$ the rows $r_u \neq r_{u'}$ are different, too. Then the precondition for the consistency requirement, namely equal rows, is never fulfilled, and consistency holds trivially. Assume that u' has been added to U after u. This can only happen if the closedness condition is violated in line 5. This, however, contradicts $r_{u'} = r_u \in \mathrm{Rows_{high}}\,\mathcal{T}$. $\square$

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 in 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 automaton $\mathfrak{A}^{P'}(\mathcal{T})$ is residual.*

Proof. Consider some $u \in U$. As $P' = \mathrm{Rows_{high}}(\mathcal{T})$, 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 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. Thus, for an appropriate expression R_u we get

$$\delta(Q_0, u) = \delta(\delta(Q_0, u'), a) \equiv \delta(r_{u'} \wedge R_{u'}, a) \equiv (r_u \wedge R'_u) \wedge R_{u'} \equiv r_u \wedge R_u \ .$$

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, the construction directly gives us a counterexample 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 the condition $L(\mathfrak{A}^P(\mathcal{T})) = L = L(\mathfrak{A}^{P'}(\mathcal{T}))$ 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 monomial $M \sqsubset \delta(q, a)$ 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, $[\![\delta(q, w)]\!] \geq [\![\delta(q', w)]\!]$. From Lemma 8 one gets $u^{-1}L = u^{-1}L(\mathfrak{A}^{P'}(\mathcal{T})) \subseteq L(\mathfrak{A}^{P'}_{q'}(\mathcal{T}))$ and from Lemma 5 $L(\mathfrak{A}^P_q(\mathcal{T})) \subseteq u^{-1}L(\mathfrak{A}^P(\mathcal{T})) = u^{-1}L$. Thus, we get $u^{-1}L \subseteq L(\mathfrak{A}^{P'}_{q'}(\mathcal{T})) \subseteq L(\mathfrak{A}^P_q(\mathcal{T})) \subseteq u^{-1}L$ and $u^{-1}L = u^{-1}L(\mathfrak{A}^P(\mathcal{T})) = L(\mathfrak{A}^P_q(\mathcal{T}))$. Therefore, the automaton $\mathfrak{A}^P(\mathcal{T})$ is residual, too. □

A basis P is called *optimal* for a regular language L if its size is minimal over all bases P for all tables $\mathcal{T}$ for L s. t. $\mathfrak{A}^P(\mathcal{T})$ is an RAFA. The *reverse* of L contains all strings $a_1 \dots a_k \in \Sigma^*$ such that $a_k \dots a_1$ is in L. Now we are ready to state the main result.

Theorem 2. *For every 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 is optimal then $\mathfrak{A}^P$ 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 in [3]. 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. [23]). 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 every P it holds: $\mathcal{M}^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})} = \mathcal{M}^P$ 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 $\mathcal{X} = \{(v,i) \mid v,i \in V \wedge \llbracket M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v) \rrbracket [i] = -\}$ be the groundset and $\mathcal{S} = \{m_u \mid u \in U\}$ with subsets $m_u = \{(v,i) \in \mathcal{X} \mid r_u \geq \llbracket M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v) \rrbracket$ 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 [17] 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.

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 $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.*

It is important to note here that P^* is a minimum basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ and does not necessarily correspond to an *optimal* basis. One can indeed construct tables $\mathcal{T}$ such that *no* basis $P \subseteq \mathrm{Rows}(\mathcal{T})$ is optimal.

4.4 Experimental Results

We ran L*, NL*, and AL** on random AFA targets. The first distribution of these random AFAs (RAT1) was generated similar to the experiments in [3] as told by [16]. For the equivalence oracle we used the probabilistic (non-error free) equivalence oracle (REQ) described in [3] and also implemented an exact version (EEQ). When REQ outputs "equivalent" this was verified by EEQ. In almost every run of L*, NL*, and AL**, at least one wrong answer given by REQ showed up. Thus, the following experiments were obtained by using the exact algorithm EEQ. However, EEQ in about 50% of all non-trivial RAT1 instances required so much computational power that the computation could not be finished. This problem is unlikely to be fixed by a more efficient implementation of EEQ, because AFA-equivalence is PSPACE-hard (NFA-equivalence is already PSPACE-complete [21]).

Therefore, to reduce the computational complexity of the instances we have generated a different set of random AFA targets (RAT2) obtained as follows.

- Every AFA has 6 states over an alphabet of size 3.
- Every state is accepting with probability $1/2$.
- With probability $1/3$, there is exactly one initial state. Otherwise, the initial configuration is a disjunction of two different random states.
- Every transition is a DNF formula, consisting of two monomials. Each monomial is a conjunction of random states. With probability $2/3$, such a monomial is of size 1, otherwise of of size 2.

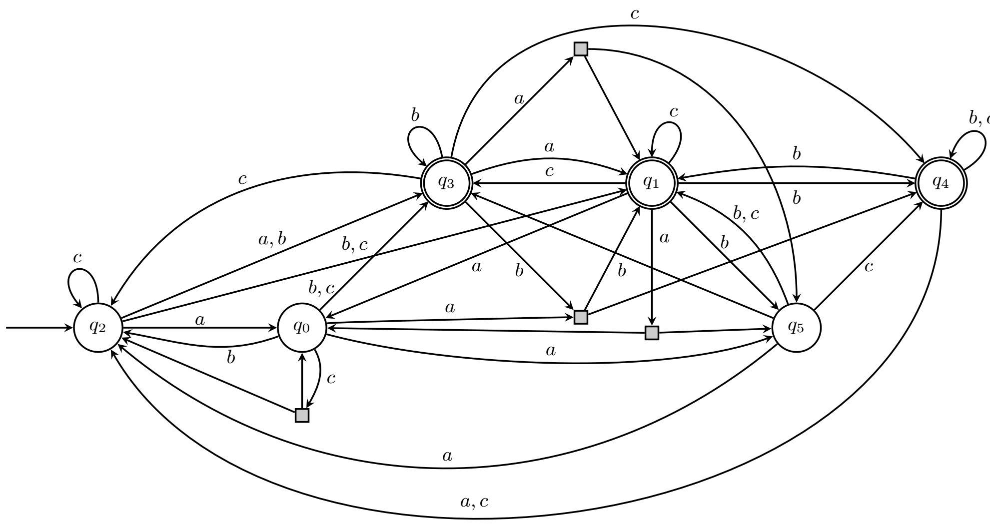

Fig. 7. An example of a RAT2 instance.

Figure 7 shows such a randomly generated target AFA. There were still about 24% non-trivial RAT2 instances we had to abort.

Figure 8 summarizes our experimental results with EEQ for RAT2 comparing the sizes of the automata generated by L*, NL* and AL**. Note that the target instances randomly generated may not be residual, while the AFAs output by AL** are always residual.

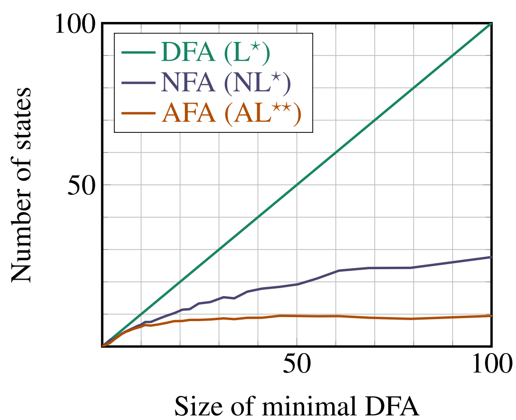

Fig. 8. Comparison of the size of automata learnt by L*, NL* and AL** for random regular languages generated by AFAs.

5 On the Size of Residual AFAs

In [3] it was 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 an RAFA that is double exponentially more succinct than the minimal DFA and uses only 2 nondeterministic (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.

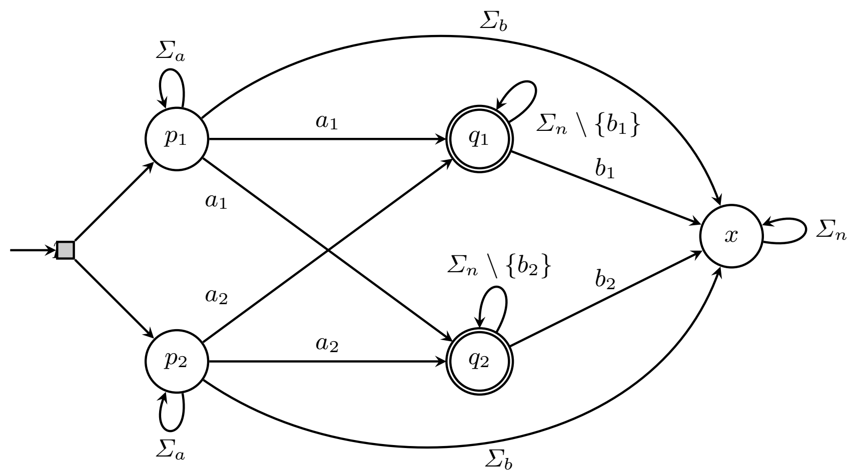

Fig. 9. The residual AFA for the language A_n of Theorem 4 with $n = 2$. The corresponding alphabet is $\Sigma_n = \Sigma_a \cup \Sigma_b$ with $\Sigma_a = \{a_1, a_2\}$ and $\Sigma_b = \{b_1, b_2\}$, the initial configuration is $Q_0 = p_1 \wedge p_2$, and the set of accepting states is $F = \{q_1, q_2\}$.

Theorem 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. The alphabet Σ_n for A_n consists of disjoint subsets $\Sigma_a = \{a_1, a_2, \ldots, a_n\}$ and $\Sigma_b = \{b_1, b_2, \ldots, b_n\}$. The language is defined as

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

We construct a residual AFA with states $\{p_1, \ldots, p_n, q_1, \ldots, q_n, x\}$ that is sketched for $n = 2$ in Fig. 9. A general construction of AFAs $\mathfrak{A}^n$ for A_n is given below:
- $Q^n = \{p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_n, x\},\ Q_0 = \bigwedge_{i=1}^n p_i,\ F = \{q_1, q_2, \ldots q_n\}$
- $\delta(p_i, a_i) = p_i \vee q_1 \vee q_2 \vee \ldots \vee q_n$
- $\delta(p_i, \sigma) = p_i$ for $\sigma \in \Sigma_a \setminus \{a_i\},\ \delta(p_i, \sigma) = x$ for $\sigma \in \Sigma_b,\ \delta(q_i, b_i) = x$
- $\delta(q_i, \sigma) = q_i$ for all $\sigma \in \Sigma_n \setminus \{b_i\},\ \delta(x, \sigma) = x$ for all $\sigma \in \Sigma_n$

Residuality follows from the following strings $u(q)$ for $q \in Q^n$ such that $L(\mathfrak{A}_q) = u(q)^{-1} A_n$:

$u(x) = a_1 a_2 \ldots a_n\ b_1 b_2 \ldots b_n,$
$u(q_i) = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_{i-1} b_{i+1} \ldots b_n,$
$u(p_i) = a_1 a_2 \ldots a_{i-1} a_{i+1} \ldots a_n.$

In order to prove the second property we use permutations on Σ_a. For a permutation π on Σ_a let $S_0(\pi) = \{\pi(1), \pi(2), \ldots, \pi(n/2)\}$, $S_1(\pi) = \{\pi(n/2{+}1), \ldots, \pi(n)\}$ and $w(\pi) = \pi(1)\pi(2)\ldots\pi(n)$. The string $w(\pi)$ contains each letter of Σ_a exactly once and hence belongs to A_n.

Let $\mathfrak{A} = (Q, Q_0, \delta, F)$ be an NFA for A_n. For $w(\pi)$, consider an accepting computation generating the sequence of states $q_\pi^1, q_\pi^2, \ldots, q_\pi^n$ with $q_\pi^n \in F$. There are $\binom{n}{n/2}$ many pairs of permutations π, π' with $S_0(\pi) \neq S_0(\pi')$. If $\mathfrak{A}$ has less than that many states there must exist two such permutations π and π' with $q_\pi^{n/2} = q_{\pi'}^{n/2}$. Hence,

$$q_\pi^1, \ldots, q_\pi^{n/2},\ q_{\pi'}^{n/2+1}, \ldots q_{\pi'}^n$$

is an accepting run of

$$w(\pi, \pi') := \pi(1)\pi(2)\ldots\pi(n/2)\ \pi'(n/2+1)\ldots\pi'(n)\ .$$

But, as $S_0(\pi) \neq S_0(\pi')$, there is some symbol $\sigma \in S_1(\pi)$ with $\sigma \notin S_1(\pi')$. Hence, σ does not occur in $w(\pi, \pi')$ and thus $w \notin A_n$. Thus, $\mathfrak{A}$ does not recognize A_n correctly.

The lower bound proof for UFAs is dual taking permutations on Σ_b. Now a string $w(\pi)$ starts with $a_1 \ldots a_n$ to fulfill the first conditions and then continues with the permutation π of the letters in Σ_b. All these strings do not belong to A_n, but omitting one letter b_i in the second part puts the input into the language. $\square$

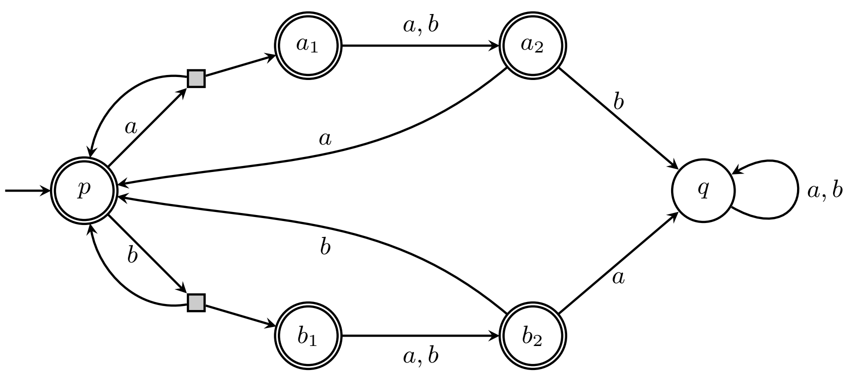

Fig. 10. The (non-residual) AFA for the language B_n of Theorem 5 with $n = 2$.

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 to prove 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 [22]). For $n = 2$, the non-residual AFA $\mathfrak{A} = (Q, Q_0, \delta, F)$ for B_n is sketched in Fig. 10.

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 $\overline{B}_n$ for $n = 1, 2, \ldots$, such that every residual AFA for B_n, resp. $\overline{B}_n$, is exponentially larger than the corresponding minimal UFA, resp. NFA. The details of the proof can be found in Appendix C.

As it has already been noted in [3], RAFAs may be double exponentially smaller than the minimal DFAs. We give a more precise bound inspired by a language defined in [6].

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 construction is given in Appendix C.
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 a $k_1(n)$ state automata of type $\mathfrak{A}_1$ 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, for more than 98% of the non-trivial instances we used in our experiments, our new algorithm AL** only performs a single additional equivalence-query to verify the residuality. Thus, based on the performance experiments 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 nondeterministic automata have been understood quite well [4,12,13,18], fundamental questions concerning residual alternating automata remain open. 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

1. Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *International Conference on Concurrency Theory*, pages 35–48. Springer, 2004.
2. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106, 1987.
3. Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *Proc. 24. IJCAI*, pages 3308–3314, 2015.
4. Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *Proc. 21. IJCAI*, pages 1004–1009, 2009.
5. Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R Piegdon. libalf: The automata learning framework. In *International Conference on Computer Aided Verification*, pages 360–364. Springer, 2010.
6. Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.
7. Yu-Fang Chen, Chiao Hsieh, Ondřej Lengál, Tsung-Ju Lii, Ming-Hsien Tsai, Bow-Yaw Wang, and Farn Wang. PAC learning-based verification and model synthesis. In *Proceedings of the 38th International Conference on Software Engineering*, pages 714–724. ACM, 2016.
8. Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.
9. Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning assumptions for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.
10. Colin De La Higuera. A bibliographical study of grammatical inference. *Pattern recognition*, 38(9):1332–1348, 2005.
11. Colin De la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
12. François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. In *Proc. 18. STACS*, LNCS 2010, pages 144–157. Springer, 2001.
13. François Denis, Aurélien Lemay, and Alain Terlutte. Learning regular languages using RFSAs. *Theoretical Computer Science*, 313(2):267–294, 2004.
14. Lu Feng, Marta Kwiatkowska, and David Parker. Compositional verification of probabilistic systems using learning. In *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*, pages 133–142. IEEE, 2010.
15. Lu Feng, Marta Kwiatkowska, and David Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *International Conference on Fundamental Approaches to Software Engineering*, pages 2–17. Springer, 2011.
16. Dana Fisman. personal communication, 2017.
17. David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.
18. Anna Kasprzik. Learning residual finite-state automata using observation tables. In *Proc. 12. DCFS*, pages 205–212, 2010.
19. Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. MIT press, 1994.
20. Carsten Kern. *Learning communicating and nondeterministic automata*. PhD thesis, RWTH, Fachgruppe Informatik, 2009.
21. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC, pages 1–9, New York, NY, USA, 1973. ACM.
22. Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 238–266. Springer, 1995.
23. David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Appendix A: A Run of AL* that Produces a Non-Residual AFA

Let

$$\begin{aligned}
\Sigma &= \{\sigma_c, \sigma_{nr}, \sigma_{c1f}, \sigma_{c1w}, \sigma_{c2f}, \sigma_{c2w}, \sigma_{w1}, \sigma_{w2}, \rho_{nr}, \rho_{c1f}, \rho_{c1w}, \rho_{c2f}, \rho_{c2w}, \rho_{w1}, \rho_{w2}, \gamma_1, \gamma_2, \omega\},\\
L_x &= \{\sigma_c\gamma_1,\ \sigma_c\gamma_2,\ \sigma_c\sigma_{nr}\rho_{nr},\ \sigma_{c1f}\rho_{c1f}, \sigma_{c1w}\rho_{c1w},\ \sigma_{c2f}\rho_{c2f},\ \sigma_{c2w}\rho_{c2w}, \sigma_{w1}\rho_{w1},\ \sigma_{w2}\rho_{w2}\},\\
L_y &= \{\sigma_{c1f}\gamma_1,\ \sigma_{c1f}\sigma_{nr}\rho_{nr},\ \sigma_{c1w}\gamma_1,\ \sigma_{c1w}\sigma_{nr}\rho_{nr}, \sigma_{c2f}\gamma_2,\ \sigma_{c2f}\sigma_{nr}\rho_{nr},\ \sigma_{c2w}\gamma_2,\ \sigma_{c2w}\sigma_{nr}\rho_{nr}\},\\
L_z &= \{\sigma_{w2}\omega,\ \sigma_{w1}\omega\omega,\ \sigma_{w1}\omega\rho_{w2},\ \sigma_{c1f}\sigma_{nr},\ \sigma_{c2f}\sigma_{nr}, \sigma_{c1w}\sigma_{nr}\rho_{w1},\ \sigma_{c1w}\sigma_{nr}\omega\rho_{w2},\ \sigma_{c1w}\sigma_{nr}\omega\omega,\\
&\qquad \sigma_{c2w}\sigma_{nr}\rho_{w1},\ \sigma_{c2w}\sigma_{nr}\omega\rho_{w2}\sigma_{c2w}\sigma_{nr}\omega\omega\},\\
L_2 &= (L_x \cup L_y \cup L_z \cup \{\sigma_c\sigma_{nr}\omega\omega\})\{\omega\}^*,
\end{aligned}$$

and $\mathfrak{A}$ be the AFA illustrated in Fig. 6. A detailed case analysis shows $L(\mathfrak{A}) = L_2$. This AFA is not residual because of the state labeled nr.

For learning L_2, the following implementation of an equivalence oracle EQ is used based on a total ordering $\prec$ over L_x given by

$$\sigma_c\sigma_{nr}\rho_{nr} \prec \sigma_{c1f}\rho_{c1f} \prec \sigma_{c2f}\rho_{c2f} \prec \sigma_c\gamma_2 \prec \sigma_c\gamma_1 \prec \sigma_{c2w}\rho_{c2w} \prec \sigma_{c1w}\rho_{c1w} \prec \sigma_{w1}\rho_{w1} \prec \sigma_{w2}\rho_{w2}.$$

For a hypothesized AFA $\mathfrak{A}'$ that does not accept L_2, EQ according to $\prec$ searches the smallest element $\xi \in L_x \setminus L(\mathfrak{A}')$. If such a ξ exists EQ returns it as counterexample, otherwise an arbitrary counterexample is chosen.[2]

We have implemented AL* with access to this equivalence oracle. For the language L_2 the non-residual automaton $\mathfrak{A}$ in Fig. 6 has been obtained as final result. The complete table $\mathcal{T}$ of the corresponding run is not presented here since it has thousands of rows.

Some Intuition Concerning the Construction of $\mathfrak{A}$

Let $Q' = \{i, c, nr, c1f, c1w, c2f, c2w, w1, w2, f\}$ be a superset of the states of $\mathfrak{A}$. For $q \in Q'$ the symbol σ_q is used to generate a row of $\mathcal{T}$ of a specific form that will serve as a state of $\mathfrak{A}$. For q to ensure that the corresponding row is prime the symbol ρ_q is used which generates a unique $+$ in this row. In the subtable in Fig. 11 see for example the second row labelled σ_{c1f} and column labelled ρ_{c1f}. This column has a single $+$ at this row that makes this row prime.02

To get a non-residual AFA, Lemma 8 implies that we have to construct a prime row r_u where $u = \sigma_c\sigma_{nr}$ such that r_{σ_c} is not prime. This is achieved by symbols γ_1, γ_2. For the subtable in Fig. 11 one notices that

$$r_{\sigma_c} \;=\; (r_{\sigma_{c1f}} \sqcap r_{\sigma_{c1w}}) \;\sqcup\; (r_{\sigma_{c2f}} \sqcap r_{\sigma_{c2w}})\,.$$

The row with label $\sigma_c\sigma_{nr}$ plays a special role representing the non-residual state nr.

We still have to make sure that the state nr corresponding to r_u is non-residual. For this purpose, we add the string $\sigma_c\sigma_{nr}\omega\omega$ to the language and make sure that the string $\omega\omega$ is not

[2] We cannot provide the sequence of counterexamples exactly because it depends on details of the implementation of AL*. In [3] the authors have suggested some optimizations in order to save membership queries. However, these optimizations may increase the number of (expensive) equivalence queries, because now AL* may produce automata that do not classify already seen counterexamples correctly. In this case, the equivalence oracle defined above would simply provide a previous counterexample again.

	ϵ	γ_1	γ_2	ρ_{c1f}	ρ_{c1w}	ρ_{c2f}	ρ_{c2w}	ρ_{nr}
σ_c	$-$	$+$	$+$	$-$	$-$	$-$	$-$	$-$
σ_{c1f}	$-$	$+$	$-$	$+$	$-$	$-$	$-$	$-$
σ_{c1w}	$-$	$+$	$-$	$-$	$+$	$-$	$-$	$-$
σ_{c2f}	$-$	$-$	$+$	$-$	$-$	$+$	$-$	$-$
σ_{c2w}	$-$	$-$	$+$	$-$	$-$	$-$	$+$	$-$
$\sigma_c\sigma_{nr}$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$+$

Fig. 11. A subtable of $\mathcal{T}$ used to construct non-residual AFA $\mathfrak{A} = \mathfrak{A}(\mathcal{T})$.

accepted while the automata is in state nr, i. e. $\omega\omega \notin L(\mathfrak{A}_{r_u})$. For the automaton to accept $\sigma_c\sigma_{nr}\omega\omega$ the suffix $\sigma_{nr}\omega\omega$ must be accepted from the configuration $(c1f \wedge c1w) \vee (c2f \wedge c2w)$. The non-residuality is achieved by the table not containing information on $\sigma_c\sigma_{nr}\omega\omega$ and $\omega\omega$. To "hide" this information, we have to make sure that ω is never added to V. This is done by two "waiting" states $w1$ and $w2$. As a path from the states $c1w$ and $c2w$ to the accepting state f visits the states $w1$ and $w2$, the automaton has to "wait" for the string $\omega\omega$ to reach f. By construction of r_{σ_c} either $c1w$ or $c2w$ have to be visited in order to accept a word. But, as $\omega \notin V$, this "waiting" behavior cannot be observed by AL* and hence $\omega\omega \notin L(\mathfrak{A}_{r_u})$.

For technical reasons one has to add some words like $\sigma_{w1}\omega\omega$ to the language to get the final version of L_2. Based on these properties, the segmentation of L_2 is as follows. The words in L_x are the counterexamples that let AL* add necessary columns to V and finally necessary rows to U. The words in L_y ensure that row r_{σ_c} gets suitably extended. Finally, the words in L_z correspond to the waiting process before merging the different $\wedge$-branches in the accepting state f.

Appendix B: Additional Claims and Proofs

Proof of Lemma 1

We prove the statements of the lemma separately.

Lemma 12. *For all* $r_u \in Q$ *and* $v \in V$: $\quad r_u[v] = + \iff \delta(r_u, v) \subseteq F.$

Proof. For $v = \epsilon$ it holds $r_u[\epsilon] = +$ iff $r_u \in F$. Since $\delta(r_u, \epsilon) = \{r_u\}$ this implies the claim. Using induction on $|v|$, for $v = av'$ one gets $r_u[v] = r_u[av'] = r_{ua}[v']$ since $u \in U$. By definition, $\delta(r_u, a) = \mathbb{B}_{\sqcap}(r_{ua}) \cap Q$. Now consider the two possible values for $r_{ua}[v']$:

- case $r_{ua}[v'] = +$:
  For every $r \in \delta(r_u, a)$ the property $r_{ua} \leq r$ implies $r[v'] = +$. By induction hypothesis, $\delta(r, v') \subseteq F$. Now $\delta(r_u, av') \subseteq F$ follows from

$$\delta(r_u, av') = \delta(\delta(r_u, a), v') = \bigcup_{r \in \delta(r_u, a)} \delta(r, v') \; .$$

- case $r_{ua}[v'] = -$:
  There must be a row $r \in Q$ with $r_{ua} \leq r$ such that $r[v'] = -$ (this row may be r_{ua} itself if it is $\sqcap$-prime). By definition, $r_{ua} \leq r$ implies $r \in \delta(r_u, a)$. The induction hypothesis gives $\delta(r, v') \not\subseteq F$ and thus $\delta(r_u, av') \not\subseteq F$.

$\square$

Using this lemma one can derive a series of properties.

Claim. For every $v \in V$: $r_\epsilon[v] = + \iff \delta(Q_0, v) \subseteq F$.

Proof. If r_ϵ is prime and thus a member of the state set Q the statement follows directly from the lemma above. Otherwise, $r_\epsilon = \bigsqcap\{r_{u_1}, r_{u_2}, \ldots, r_{u_k}\}$ with states r_{u_i}, and thus

$$Q_0 \;=\; \mathbb{B}_\sqcap(r_\epsilon) \cap Q \;=\; \{r_{u_1}, r_{u_2}, \ldots, r_{u_k}\}\;.$$

Now one can apply the lemma above to the r_{u_i}. □

Claim. If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible then $r_u \in \delta(Q_0, u)$ for all $r_u \in Q$.

Proof. Because r_u is prime, for all $Q' \subseteq Q\backslash\{r_u\}$ there exists some $v \in V$ such that $(\bigsqcap_{r \in Q'} r)[v] \neq r_u[v]$. If we assume $r_u \notin \delta(Q_0, u)$ then there must exists $v \in V$ such that $(\bigsqcap_{r\in\delta(Q_0,u)} r)[v] \neq r_u[v]$. This implies for $r_u[v]$:

- case $r_u[v] = -$:
  $(\bigsqcap_{r\in\delta(Q_0,u)} r)[v] = +$ and we know that $r[v] = +$ for all $r \in \delta(Q_0, u)$. Lemma 12 implies $\delta(r, v) \subseteq F$ for all $r \in \delta(Q_0, u)$. As $\delta(Q_0, uv) = \bigcup_{r\in\delta(Q_0,u)} \delta(r, v)$ one can deduce $\delta(Q_0, uv) \subseteq F$ and thus $uv \in L(\mathfrak{A}(\mathcal{T}))$. Since $r_u[v] = -$ this is a contradiction to the compatibility condition.
- case $r_u[v] = +$:
  $(\bigsqcap_{r\in\delta(Q_0,u)} r)[v] = -$ and there exists a row $r \in \delta(Q_0, u)$ with $r[v] = -$. Again Lemma 12 implies $\delta(r, v) \not\subseteq F$. Then $\delta(Q_0, uv) = \bigcup_{r\in\delta(Q_0,u)} \delta(r, v)$ yields $\delta(Q_0, uv) \not\subseteq F$ and thus $uv \notin L(\mathfrak{A}(\mathcal{T}))$. This leads to a contradiction as well since $r_u[v] = +$. □

Claim. If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible then for all $u \in U$ and $r \in \delta(Q_0, u)$: $r_u \leq r$.

Proof. Suppose that this is not true, i. e. there exists $r \in \delta(Q_0, u)$ such that $r_u \not\leq r$. Hence, there is a $v \in V$ such that $r_u[v] = +$ and $r[v] = -$. Applying Lemma 12 to r we get $\delta(r, v) \not\subseteq F$ and thus $\bigcup_{r\in\delta(Q_0,u)} \delta(r, v) = \delta(Q_0, uv) \not\subseteq F$. Hence, $uv \notin L(\mathfrak{A}(\mathcal{T}))$, which is a contradiction again. □

Claim. If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible then for all $r_u, r_{u'} \in Q$ and $a \in \Sigma$: $r_{u'} \leq r_u \Rightarrow r_{u'a} \leq r_{ua}$.

Proof. Assume $r_{u'} \leq r_u$, but $r_{u'a} \not\leq r_{ua}$, i. e. there exists $v \in V$ such that $r_{u'a}[v] = +$, but $r_{ua}[v] = -$. From the claims above one can deduce $r_{u'} \in \delta(Q_0, u')$. By definition of δ, every prime row that is greater or equal than $r_{u'}$ must be in $\delta(Q_0, u')$, too. Thus we have $r_u \in \delta(Q_0, u')$. Further, $\bigsqcap_{r\in\delta(r_u,a)} r = r_{ua}$, i. e. there exists a prime row $r_{u''} \in \delta(r_u, a)$ with $r_{u''}[v] = -$. By Lemma 12, $\delta(r_{u''}, v) \not\subseteq F$. Hence $r_{u''} \in \delta(\delta(Q_0, u'), a)$ and $\delta(\delta(\delta(Q_0, u'), a), v) \not\subseteq F$. Finally, $u'av \notin L(\mathfrak{A})$. Again this contradicts compatibility as $r_{u'a}[v] = +$. □

Claim. If $\mathcal{T}$ and $\mathfrak{A}(\mathcal{T})$ are compatible then for all $r_u, r_{u'} \in Q$ holds:

$$r_{u'} \leq r_u \iff \forall\, w \in \Sigma^* \quad [\delta(r_u, w) \not\subseteq F \Rightarrow \delta(r_{u'}, w) \not\subseteq F].$$

Proof. Note that w does not necessarily belong to V.

- case $r_{u'} \leq r_u$:
  Consider a w with $\delta(r_u, w) \not\subseteq F$. Using induction on $|w|$, for $w = \epsilon$ we get $w \in V$ and the claims above give $r_u[\epsilon] = -$. $r_{u'} \leq r_u$ implies $r_{u'}[\epsilon] = -$. Hence, $r_{u'} \notin F$ and $\delta(r_{u'}, \epsilon) \not\subseteq F$. For $w = aw'$ one gets $\delta(r_u, w) = \delta(r_u, aw') = \delta(\delta(r_u, a), w')$. As $\delta(r_u, w) \not\subseteq F$, there is a row $r \in \delta(r_u, a)$ such that $\delta(r, w') \not\subseteq F$. $r \in \delta(r_u, a)$ implies $r_{ua} \leq r$. As $r_{u'} \leq r_u$, by the claim above $r_{u'a} \leq r_{ua}$ and thus $r_{u'a} \leq r$. Hence, $r \in \delta(r_{u'}, a)$ and due to $\delta(r_{u'}, aw') = \bigcup_{r \in \delta(r_{u'}, a)} \delta(r, w')$ one can conclude $\delta(r_{u'}, w) \not\subseteq F$.

- case $r_{u'} \not\leq r_u$:
  There is a $v \in V$ such that $r_{u'}[v] = +$ and $r_u[v] = -$ and by Lemma 12 $\delta(r_u, v) \not\subseteq F$ and $\delta(r_{u'}, v) \subseteq F$. Choosing this v as w proves the statement. □

This completes the proof of Lemma 1.

Proof of Lemma 3

We prove the lemma by induction upon the length of v. For $v = \epsilon$ one gets $r[\epsilon] = [\![r]\!]\,[\epsilon] = [\![\delta(r, \epsilon)]\!]\,[\epsilon]$.
For $v = av'$ we have $r[v] = r[av'] = ra[v']$, as $r \in P$ and $v' \in V$ due to its suffix-closedness.

- case $ra[v'] = +$:
  It holds $\delta(r, a) = b^P\,(ra) = \bigvee_{\substack{v \in V \\ ra[v]=+}} M^P(v)$. As $v' \in V$ and $ra[v'] = +$ one can conclude $M^P(v') \sqsubset \delta(r, a)$. For every $r' \sqsubset M^P(v')$ the induction hypothesis implies $[\![\delta(r', v')]\!]\,[\epsilon] = r'[v'] = +$ by the definition of $M^P(v')$. Hence

  $$[\![\delta(M^P(v'), v')]\!]\,[\epsilon] = \left[\!\!\left[\delta\Big(\bigwedge_{r' \in P,\ r'[v']=+} r', v'\Big)\right]\!\!\right][\epsilon] = + \,.$$

  Finally, as $\delta(r, a)$ contains the monomial $M^P(v')$ one can conclude

  $$[\![\delta(r, v)]\!]\,[\epsilon] \;=\; [\![\delta(r, av')]\!]\,[\epsilon] \;=\; [\![\delta(\delta(r, a), v')]\!]\,[\epsilon] \;\geq\; [\![\delta(M^P(v'), v')]\!]\,[\epsilon] = + \,.$$

- case $ra[v'] = -$:
  For every monomial $M \sqsubset \delta(r, a)$ it must hold $[\![M]\!]\,[v'] = -$. Thus, there is a row $r_M \in P$ with $r_M[v'] = -$. The induction hypothesis then implies $[\![\delta(r_M, v')]\!]\,[\epsilon] = -$. So, for every $M \sqsubset \delta(r, a)$ we get $[\![\delta(M, v')]\!]\,[\epsilon] = -$, and finally

  $$[\![\delta(r, v)]\!]\,[\epsilon] \;=\; [\![\delta(\delta(r, a), v')]\!]\,[\epsilon] \;=\; [\![\delta(b^P\,(ra)\,, v')]\!]\,[\epsilon] = - \,.$$

Hence, $ra[v'] = +$ iff $[\![\delta(r, v)]\!]\,[\epsilon] = +$ which implies $r[v] = +$ iff $[\![\delta(r, v)]\!]\,[\epsilon] = +$.

Proof of Theorem 2

Lemma 9 implies that the output of AL** is always residual. The number of states of the final hypothesis equals the size of the basis. Thus, an optimal basis leads to a minimal number of states. The table $\mathcal{T}$ cannot have more different rows than κ_L, the number of states of the minimal DFA for L (compare Lemma 5 of [4]).

Claim. $r_\epsilon[v] = \llbracket \delta(Q_0, v) \rrbracket\, [\epsilon]$ for every $v \in V$.

Proof. Choose $\varphi = b^P(r_\epsilon)$. By construction we have $r_\epsilon = \llbracket b^P(r_\epsilon) \rrbracket$. Now we can apply Lemma 4. □

Claim. If $c \in \Sigma^*$ is classified incorrectly by the AFA then there exists a suffix v of c such that the corresponding column $v \notin V$ is different from all columns in V.

Proof. The claim above shows that every $w \in V$ is classified correctly by $\mathfrak{A}_A^P(\mathcal{T})$ as well as by $\mathfrak{A}_A^{P'}(\mathcal{T})$. So, for every counterexample $c \in \Sigma^*$ we know that c is not classified correctly be the current AFA, but will be classified correctly by every future AFA, which will be constructed from a table $\mathcal{T}' = (T', U', V')$ with c in V'. Hence, δ must be changed. This can only be the case if either one of the new columns (added when seeing the counterexample) differs from all of the old columns, or if a new row is added to $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$. However, to add a new row, the table must have become non-P-closed. Therefore, a column that differs from every old column must have been added before. □

Thus, the maximal number of different columns is bounded by the minimal number of states of a DFA for the reserve language of L denoted by $\hat{\kappa}_L$. Note that $\hat{\kappa}_L \leq 2^{\kappa_L}$. Thus both, $\mathfrak{A}_A^P(\mathcal{T})$ and $\mathfrak{A}_A^{P'}(\mathcal{T})$, must be equivalent to the unknown language L after a finite number of counterexamples. Thus, AL** terminates.

By construction, $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ does not contain a row more than once. So, $|U|$ is bounded by κ_L and $\mathrm{Rows}(\mathcal{T})$ by $(1 + |\Sigma|)\, \kappa_L$. V is bounded by the number of equivalence queries multiplied by the length of the counterexamples. Therefore, $|V| \leq \ell\, \hat{\kappa}_L$.

The size of the final table is thus at most $\kappa_L \hat{\kappa}_L (1 + |\Sigma|)\ell$, and also the number of membership queries. The total running time of AL** is polynomial in the size of the final table.

Proof of Lemma 10

Assume that P is a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$, but $\mathcal{M}^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})} \neq \mathcal{M}^P$. By construction, for every $m = \llbracket M^P(v) \rrbracket \in \mathcal{M}^P$ there must be some $m' = \llbracket M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v) \rrbracket \in \mathcal{M}^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}$ with $m' \leq m$. By assumption, there must be such a pair m, m' with $m' < m$. Now consider $v, v' \in V$ such that $m = \llbracket M^P(v) \rrbracket$, $m' = \llbracket M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v) \rrbracket$ and $m'[v'] < m[v']$. There must be a row $r_u \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ with $r_u[v] = +$ and $r_u[v'] = -$. Note that $r_u \in M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v)$. From $\llbracket M^P(v) \rrbracket\, [v'] = +$ we know that P cannot contain such a row r_u. Thus, every monomial over P that evaluates to $+$ at position v must evaluate to $+$ at position v'. But then, $r_u \in \mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$ cannot be composed from P by a DNF. Thus, P cannot be a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$.

Now assume that P is not a basis of $\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})$. So there is some $u \in U$ such that r_u cannot be expressed by a DNF over P. Thus, there is a $v \in V$ with $u[v] = +$, but $\llbracket M^P(v) \rrbracket > r_u = \llbracket M^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})}(v) \rrbracket$. This implies $\mathcal{M}^{\mathrm{Rows}_{\mathrm{high}}(\mathcal{T})} \neq \mathcal{M}^P$.

Appendix C: Construction of Separating Languages

Proof of Theorem 5

We start with an auxiliary lemma. For an AFA $\mathfrak{A} = (Q, Q_0, F, \delta)$ and $\varphi \in \mathcal{F}(Q)$ let $\mathfrak{A}_\varphi = (Q, \varphi, F, \delta)$ denote the AFA starting with the initial configuration φ.

Lemma 13. *Let L be a regular language and $\mathfrak{A}$ an AFA accepting L. For every $w \in \Sigma^*$, there is a formula $\varphi_w \in \mathcal{F}(Q)$ such that $L(\mathfrak{A}_{\varphi_w}) = w^{-1}L$.*

Proof. Suppose that this is wrong and there exists a $\widehat{w} \in \Sigma^*$ that for every $\varphi \in \mathcal{F}(Q)$, $L(\mathfrak{A}_\varphi) \neq \widehat{w}^{-1}L$. Hence, for every $\varphi \in \mathcal{F}(Q)$, there is a string v_φ such that $v_\varphi \in L(\mathfrak{A}_\varphi) \triangle \widehat{w}^{-1}L$. This means that $\widehat{w}v_\varphi$ is wrongly classified by $\mathfrak{A}$. $\square$

Now we are ready to give the proof of Theorem 5. First, let us construct an AFA for B_n:
- $Q = \{p, q, a_1, \ldots, a_n, b_1, \ldots, b_n\},\ Q_0 = \{p\},\ F = \{p, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}$
- $\delta(p, a) = p \wedge a_1$, $\delta(p, b) = p \wedge b_1$
- $\delta(a_i, \sigma) = a_{i+1}$ for $i < n$ and $\sigma \in \Sigma$
- $\delta(b_i, \sigma) = b_{i+1}$ for $i < n$ and $\sigma \in \Sigma$
- $\delta(a_n, a) = p,\ \delta(b_n, b) = p,\ \delta(a_n, b) = q,\ \delta(b_n, a) = q$
- $\delta(q, \sigma) = q$ for all $\sigma \in \Sigma$

It should not be too difficult to convince oneself that this AFA does the job.
To prove that every residual AFA accepting B_n has at least 2^n states, let $S = \{u \mid u \in \{a, b\}^*, |u| \leq n\}$ be the set of strings of maximal length n. For $w = w_1 w_2 \ldots w_m$ in B_n, with $w_i \in \Sigma$ and $m > n$, we have that $w^{-1}B_n = (w_{m-n+1}w_{m-n+2} \ldots w_m)^{-1}B_n$ by the construction of B_n. Hence, for each residual language L' of B_n, there is a string $u \in S$ such that $L' = u^{-1}B_n$. For $u, u' \in S$ with $u \neq u'$, we have either $u^{-1}B_n \subsetneq u'^{-1}B_n$, or $u'^{-1}B_n \subsetneq u^{-1}B_n$, or $u^{-1}B_n \cap u'^{-1}B_n = \emptyset$, due to the construction of B_n. Hence, there is a bijection between $\mathrm{RES}(B_n)$ and S.

Let $\mathfrak{A}$ be a residual AFA for B_n with states Q. As $\mathfrak{A}$ is residual, every state $q \in Q$ corresponds to a residual language and thus to a string $u_q \in S$. Now consider a string $v \in \Sigma^n$. We have $(v^{-1}B_n) \cap \Sigma^n = \{v\}$. In order to correctly recognize B_n, one can see that there is a configuration $\varphi_v \in \mathcal{F}(Q)$ such that $L(\mathfrak{A}_{\varphi_v}) = v^{-1}B_n$ (see Lemma 13 above). Without loss of generality, suppose that $\varphi_v = \bigvee_{i=1}^{k} M_i$ and $M_i \subseteq Q$. Remember that for all residual languages, they are either disjoint or one of the languages is a subset of the other. Hence, for every M_i, either $L(\mathfrak{A}_{M_i}) = \emptyset$ (and it thus can be removed from φ_v) or there is a state $q_i \sqsubset M_i$ such that $L(\mathfrak{A}_{M_i}) = L(\mathfrak{A}_{q_i})$. Now φ_v can be represented as a conjunction of states, i. e. $\varphi_v \equiv \bigvee_{i=1}^{k} q_i$. But, as $L(\mathfrak{A}_{\varphi_v}) = v^{-1}B_n$, we conclude that there is a single state $q_v \in Q$ such that $\varphi_v \equiv q_v$ (as the language of every other state is either disjoint or a proper superset). As all of these states need to be disjoint, we have $|Q| \geq |\Sigma|^n = 2^n$.

Proof of Theorem 6

Consider the alphabet

$$\Sigma^n = \{a, b, \text{€}, \$\} \cup \{s_i \mid 1 < i \leq n\} \cup \{s_{i,\sigma} \mid 1 \leq i \leq n \wedge \sigma \in \{a, b\}\}.$$

For sake of simplicity, let $\mathcal{I} = \{i, (i, \sigma) \mid 1 \leq i \leq n, \sigma \in \{a, b\}\}$ be the indices of the alphabet symbols s_i (resp. $s_{i,\sigma}$). The AFA $\mathfrak{A}_n = (Q, Q_0, \delta, F)$ is constructed as follows.

– $Q = \{q_i, q_{i,\sigma}, q_{i,\sigma,j} \mid 1 \le i \le n, \sigma \in \{a,b\}, 0 \le j \le n\},\ Q_0 = q_1$
– $F = \{q_{i,\sigma,n} \mid 1 \le i \le n \wedge \sigma \in \{a,b\}\}$
– $\delta(q_1, €) = q_1,\ \delta(q_1, s_I) = q_I$ for all indices $I \in \mathcal{I}$
– $\delta(q_1, a) = (q_{1,a} \wedge q_2) \vee q_1,\ \delta(q_1, b) = (q_{1,b} \wedge q_2) \vee q_1$
– $\delta(q_i, a) = q_{i,a} \wedge q_{i+1},\ \delta(q_i, b) = q_{i,b} \wedge q_{i+1}$ for $1 < i < n$
– $\delta(q_n, a) = q_{n,a},\ \delta(q_n, b) = q_{n,b}$
– $\delta(q_{i,\sigma}, a) = \delta(q_{i,\sigma}, b) = \delta(q_{i,\sigma}, €) = q_{i,\sigma}$ for $1 \le i \le n$ and $\sigma \in \{a,b\}$
– $\delta(q_{i,\sigma}, \$) = q_{i,\sigma,0}$ for $1 \le i \le n$ and $\sigma \in \{a,b\}$
– $\delta(q_{i,\sigma,i-1}, \sigma) = q_{i,\sigma,i}$ for $1 \le i \le n$ and $\sigma \in \{a,b\}$
– $\delta(q_{i,\sigma,j}, a) = \delta(q_{i,\sigma,j}, b) = q_{i,\sigma,j+1}$ for $1 \le i \le n$, $j \ne i-1$, and $\sigma \in \{a,b\}$
– $\delta(q, \sigma) = \bot$ for every $q \in Q$ and $\sigma \in \Sigma^n$ such that $\delta(q,\sigma)$ has not been defined above, where $\bot$ is the empty DNF.

The corresponding automaton $\mathfrak{A}_2$ is shown in Fig. 12.

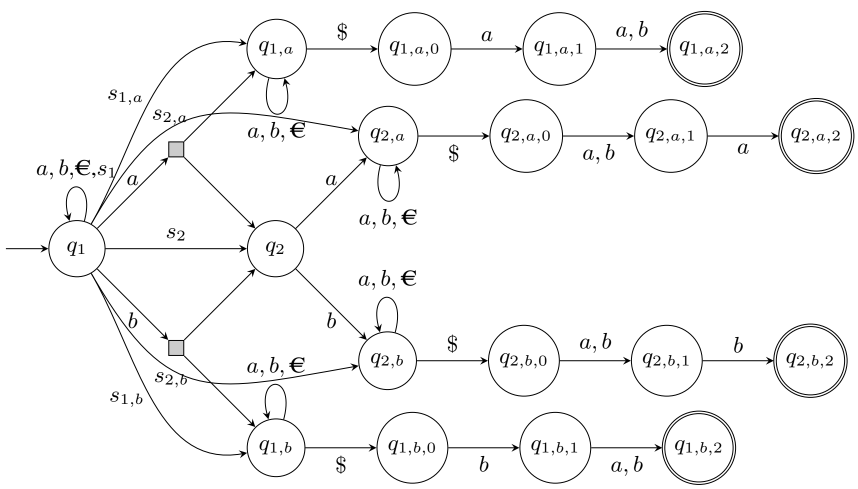

Fig. 12. The AFA $\mathfrak{A}_n$ for $n = 2$.

Note that $\mathfrak{A}_n$ has $n + 2n + 2n(n+1) = 2n^2 + 5n$ states and its transitions are of polynomial size. It accepts the language

$$C_n = \{uwv\$w \mid u, v \in \{a,b,€\}^* \wedge w \in \{a,b\}^n\} \cup \{us_Iv_I \mid\ u \in \{0,1,€\}^* \wedge I \in \mathcal{I} \wedge v_I \in L((\mathfrak{A}_n)_{q_I})\}.$$

This language is inspired by [6]. It remains to show that $\mathfrak{A}_n$ is residual, and that C_n has at least 2^{2^n} different Nerode classes.

1. For every index $I \in \mathcal{I}$ it holds $L\left((\mathfrak{A}_n)_{q_I}\right) = (s_I)^{-1}C_n$, because $\delta(Q_0, s_I) = q_I$. For $q_{i,\sigma,j}$ with $1 \le i \le n$ and $\sigma \in \{a,b\}, 0 \le j \le n$, consider the set $\{w_1, \ldots, w_{2^{n-1}}\} = \{a,b\}^{i-1}\sigma\{a,b\}^{n-i}$. Since $L\left((\mathfrak{A}_n)_{q_{i,\sigma,j}}\right) = (w_1€w_2€\ldots€w_{2^{n-1}}\$\sigma^j)^{-1}C_n$ the AFA $\mathfrak{A}_n$ is residual.
2. For any subset $W = \{w_1, \ldots, w_\ell\}$ of $\{a,b\}^n$ it holds $(w_1€w_2€\ldots€w_\ell\$)^{-1}C_n = W$. Thus, the number of different Nerode classes of C_n is at least 2^{2^n}.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il