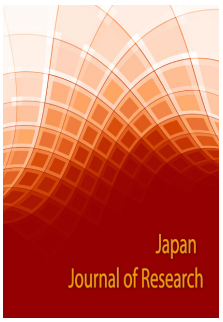
Japan Journal of Research

Original Article



Exploring a Residual Language Learning Algorithm Through the Lens of L* and Reversal Alternation

Aziz Fellah

School of Computer Science and Information Systems, Northwest Missouri State University, Maryville, MO 64468, USA

Correspondence

Aziz Fellah

School of Computer Science and Information Systems, Northwest Missouri State University, Maryville, MO 64468, USA

Abstract

Residuality Theory has recently emerged as a powerful framework for understanding the learning of formal languages. It enriches regular languages with linguistic meaning and reveals deeper semantic layers inherent in their structure. Alternation Theory, where existential and universal quantifiers interchange during computation, offers a succinct and expressive representation of regular languages.

In this paper, we investigate how residuality and reversal alternation influence the learning of regular languages, with a foundation grounded in Angluin's L algorithm. Building on these theoretical perspectives, we introduce a trilateral canonical framework called Learner-Teacher-Expert (LTEx), which incorporates an extended and diverse query set. This leads to the development of a new polynomial-time learning algorithm, the Residual Reversal-Alternating (RAL*) for learning regular languages.*

We demonstrate that the integration of residuality, reversal alternation, and L enables the learning of extended regular languages and facilitates their representation as a family of structured finite-state machines called Residual Alternating Finite Automata (RAFA). Finally, we reflect on these constructs as conceptual metaphors, proposing them as potential avenues for further research in formal language learning.*

- Received Date: 18 June 2025
- Accepted Date: 20 July 2025
- Publication Date: 20 Aug 2025

Keywords

Residuality, residual languages, alternation, residual alternating finite state automata, learning algorithms, Angluin's algorithm L.*

Introduction

The concept of *Residuality* has been introduced by Denis et al [1,2] in the context of finite state automata. It is considered a natural distillation of the essence of the automaton's states language recognition. Residuality adds foundational linguistic meaning to the automaton's states in the context of regular languages and discerns significant facts from the semantics of each state of the automaton. In the context of regular languages, Residual Finite Automata (RFA) are a subclass of Nondeterministic Finite Automata (NFA) where each state represents a language called *residual language* of the language recognized by the NFA. An automaton $\mathcal{A}$ accepting a language $\mathcal{L}$ is *residual* if every state q of $\mathcal{A}$ represents a residual language. That is, if for every state q of $\mathcal{A}$, there exists a word u such that the language accepted by $\mathcal{A}_q$, the automaton $\mathcal{A}$ that starts in state q, is the set of all words v such that uv is in $\mathcal{L}$. In terms of derivatives, an automaton accepting a language $\mathcal{L}$ is residual if the language of each state is a derivative of $\mathcal{L}$. In addition, residuality plays an important role in the context of machine learning inference, especially in areas of computer science such as inference in finite state machines, regular languages, and grammars, see [3-6].

RFA are introduced as a solution to the well-known problem of NFA not having unique minimal (in terms of the number of states) representatives. The class of RFA lies between Deterministic Finite Automata (DFA) and nondeterministic Finite Automata (NFA). With their own specific properties, such a refined class of residual automata allows one to eventually observe each state independently and describe its formal semantic subsequently. Additionally, RFA share in common a number of significant properties in the context of determinism and nondeterminism settings. For instance, RFA share with NFA the existence of automata that are exponentially smaller, in the number of states, than the corresponding minimal DFA for the same language. Importantly, every DFA exhibits the property of residuality, which underlines several active learning algorithm techniques for finite state automata, such as the seminal algorithm L* of [3] for learning DFA and the generalized algorithm NL of [7] for learning NFA. Such algorithms have provoked a tremendous amount of research in several areas of computer science such as machine learning, artificial intelligence (AI), and software verification [8-11]. Broadly speaking, formal

Citation: Fellah A. Exploring a Residual Language Learning Algorithm Through the Lens of L* and Reversal Alternation. Japan J Res. 2025;6(10):156.

languages play a significant role in shaping the perspective of natural language processing, machine learning algorithms, and data processing. Moreover, regular languages are used in many applications of AI from pattern matching to image recognition.

The notion of *alternation* is a natural generalization of nondeterminism, receiving its historical and formal treatment in the seminal paper by [12]. Alternation provides a succinct representation of regular languages, while residuality adds a natural meaning to the automaton's states. This seminal paper and subsequent research [6,13-15] have focused on various types, sizes, languages, and computational complexities of Alternating Finite Automata (AFA). Thus, alternation has played an important role in understanding many questions in complexity theory and model checking. All these automata (*i.e.,* DFA, NFA, AFA, RFA) share the same expressive power in terms of language recognition − they all accept regular languages but differ in efficiency. In terms of the number of states, a minimal DFA might be exponentially larger than an NFA and double-exponentially larger than an AFA. Furthermore, the presence of alternation can lead to simplified construction in finite automata [5,13-21]. AFA have particularly emerged as practical tools in a wide variety of applications, such as the extended version of AFA for learning AL* [22,23], query learning for regular languages [3,7,22,24], and software model checking [25-27].

These two prominent metaphors, residuality and alternation, are considered a stepping stone towards machine learning algorithms. Enlightening and discerning the residual facts from the semantics of each state of a compacted AFA is a step towards a better understanding of the learnability of finite state machines when inferring unknown regular languages. Angluin's L* algorithm for inferring an unknown regular language using membership and equivalence queries has provoked a tremendous challenge of research in various directions [5-7,11,23,28,29].

Broadly speaking, we present and extend the framework of L* as introduced in the original algorithm of [3]. That is, we extend L* by developing a model that we refer to as Learner-Teacher-Expert (LTEx). First, our goal is to provide foundational insights on finite state automata learning. Then, for learning regular languages, we develop a framework that is based on L* algorithm by exploring reversal alternation and residuality techniques, which subsequently infer finite state automata models from traces. Such a formalism is composed of three entities, a learner, a teacher, and an expert, each with a different role.

The learner, who initially knows nothing about the regular language $\mathcal{L}$, attempts to learn $\mathcal{L}$ by interacting with the teacher. The learner repeatedly makes queries to the teacher, who typically works in a black-box fashion and has access to the language in question through the expert. The teacher can answer straightforward membership and equivalence queries by further exploiting additional information and knowledge from the expert. The teacher plays an intermediate role by refining and forwarding the learner's residuality and reversal alternation queries to the expert. What is more important is the expert's knowledge of the relationship between different classes of automata and related languages (*i.e.*, determinism, nondeterminism, alternation, residual, and reversal). For example, the states of DFA and all versions of AFA have the property of residuality, which infers the existence of residual languages [1,2,29].

The teacher provides an interface (membership and equivalence queries) for the learner. In turn, the expert subsequently provides an interface (reversal membership and equivalence queries) to the teacher. In a membership query (MQ), the learner chooses a word w and asks the teacher, "Is the word $w \epsilon \Sigma^*$ in the language $\mathcal{L}$?", where Σ^* is the set of all words over the alphabet Σ. In an equivalence query (EQ) the learner selects a hypothesis, DFA $\mathcal{H}$, and the teacher answers whether $\mathcal{H}$ recognizes the language $\mathcal{L}$. That is, whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}$. The teacher returns either *"yes"* if the equivalence query $\mathcal{H}$ and the inferred model are equivalent, otherwise, it submits a counterexample, *i.e.*, a word in which $\mathcal{L}$ differs from the language of $\mathcal{H}$. In comparison to the teacher, the expert answers two types of queries, the reversal membership query (r-MQ) and the reversal equivalence query, (r-EQ). The expert acts in conformance with the teacher's potential requests. Our framework forms a trilateral interaction between the learner, teacher, and expert, which is summarized in four different query mappings: *(a)* Membership Query (MQ), *(b) Re*versal Membership Query (r-MQ), *(c)* Equivalence Query (EQ), and *(d)* Reversal Equivalence Query (r-EQ). On the other hand, L* is based on the classical bilateral interaction between the learner and teacher.

Furthermore, we encapsulate the two-mode properties, residuality and alternation, into a more expressive automaton model that we refer to as Residual Alternating Finite Automata (RAFA). Then, following the analogical lines of L*, we reexamine L* algorithm and expend it with residuality and reversal alternation. That is, we introduce a new canonical framework that we refer to as Learner-Teacher-Expert (LTEx), which is regulated with a variety of queries. Subsequently, this canonical paradigm has led to an efficient residual alternating learning algorithm that we refer to as the Residual Reversal-Alternating (RAL*). As a notational convention in this paper, we denote by the letter "R" has a double meaning residuality and reversal. Also, we use the "*" by analogy to the seminal learning algorithm L*.

The remainder of the paper is organized as follows: In Section 2, we introduce preliminary concepts, notations, and definitions. Language interpretation, nullable regular expressions, and languages are described in Section 3. Section 4 formulates an algebraic language approach and establishes an underlying algebraic approach. Section 5 covers alternating finite automata (AFA) with straightforward details, showing that AFA are a suitable framework for active automata learning algorithms. Furthermore, AFA are strengthened even more by expressing such automata in terms of a system of language equations and solving such equations in Section 6. We also introduce residual language equations that parallel the solution of algebraic equations. Section 7 puts emphasize on the related work and background of derivatives of regular expressions and languages. Section 8 discusses residuality and describes such a property as a desirable feature in language algorithmic learning. Section 9 extends the aforementioned results of the native AFA in two directions, language equations over Boolean operations and residual languages. In Section 10, we introduce two related frameworks, s-RAFA and rs-RAFA, which lay the groundwork for the residual-alternating learning algorithm and establish some fundamental results. In Section 11, we briefly review the classical learning algorithm L* for DFA by Angluin, the first of its kind that encapsulates the essence

of innovation. Along the analogical lines of L* which is augmented with residuality and reversal alternation, we introduce in Section 12 a new framework that we refer to as LTEx, equipped by residuality, reversal alternation, and a set of queries. We also demonstrate the importance of reversal alternation and residuality properties, leading to a new algorithm for learning regular languages called Residual Reversal-Alternating (RAL*). Finally, in Section 13 we conclude the paper with a summary and highlight looming future work and potential research directions.

Preliminaries

In this section, we briefly recall some relevant definitions. An alphabet is a finite, nonempty set. The elements of an alphabet are called symbols or letters. A string over an alphabet Σ is a finite sequence consisting of zero or more symbols of Σ. Without loss of generality, we assume in the sequel that alphabets do not contain any of the "special" symbols: $(\varnothing,\varepsilon,\cup,\cap,*,\vee,\wedge,+,\cdot,-,(\ ,))$. The string consisting of zero letters is called the empty string, denoted by ϵ. The length of a string *w,* denoted by $|w|$. is the number of symbols in *w*. By definition, $|\epsilon| = 0$. The set of all strings (respectively, all nonempty strings) over an alphabet Σ is denoted by Σ* (respectively, Σ⁺). A language is said to be *nullable* if it contains the empty string, ϵ, that is, a language $\mathcal{L}$ is nullable if $\epsilon \in \mathcal{L}$. A language $\mathcal{L}$ over Σ is a (possibly infinite) set of finite strings $\mathcal{L} \subseteq \Sigma^*$. We denote the language of an automaton $\mathcal{A}$ by $\mathcal{L}(\mathcal{A})$ and the language accepted by a state $q \in Q$ by $\mathcal{L}_q$ Regular languages are a special class of languages used in many applications, ranging from compilers and recent modern languages to web services. Given a language $\mathcal{L}$ over an alphabet Σ, the Kleene star closure ("∗") of $\mathcal{L}$ is the set $\mathcal{L}^* = U_{i=0}^{\infty}\mathcal{L}^i$ and the positive Kleene plus ("+") of $\mathcal{L}$ is the set $\mathcal{L}^+ = U_{i=1}^{\infty}\mathcal{L}^i$. The language $\overline{\mathcal{L}} = \Sigma^* \setminus \mathcal{L}$ is the complement of $\mathcal{L}$. The concatenation of two strings *u* and *v* is the string consisting of the symbols of *u* followed by the symbols of *v*, denoted *u.v.* (also often written as *uv*). We use the symbol "·" to show the concatenation operation, which we sometimes omit in this work. We denote the reversal of a string *w* by w^r, while the reversal of a language $\mathcal{L}$. denoted $\mathcal{L}^r$, is defined as $\mathcal{L}^r = \{ w^r \mid w \in \mathcal{L}\}$. A prefix-closed set is a set where every prefix of every member is also a member of the set. For example*: {aba, ab, a, baa, b}.* A suffix-closed set is a set where every suffix of every member is also a member of the set. For example: *{abb, bb, b, baa, aa, a}*. Finite state automata, typically deterministic and nondeterministic versions, are the two fundamental representations of regular languages. In this paper, we equivalently refer to finite state automata as finite automata.

A nondeterministic finite state automaton (NFA) is a quintuple $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ where Σ is the alphabet, *Q* is a finite set of states, Q_0 is a set of initial states $Q_0 \subseteq Q$, δ: $Q \times \Sigma \rightarrow 2^Q$ is the transition function, and $F \subseteq Q$ is a set of final states. $\mathcal{A}$ is called deterministic finite-state automaton (DFA) if $|Q_0| = 1$ and δ: $Q \times \Sigma \rightarrow Q$. The transition function δ can always be extended to δ: $Q \times \Sigma^* \rightarrow 2^Q$ defined as $\delta(q, \epsilon) = \{q\}$ and $\delta(q, wa) = \delta(\delta(q,w),a)$ for $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$. The language accepted by $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(Q_0, w) \cap F \neq \varnothing\}$$

Semantic Interpretation and Emptiness Analysis of Regular Languages and Expressions

Regular expressions are formal notations for describing regular languages. Let *e* denote a regular expression, we define the regular language of *e* to be $\mathcal{L}(e)$. Furthermore, we define the set of regular expressions by *E* over Σ as the subset of $(\Sigma \cup \{\epsilon,\emptyset,+,\cdot,*,-,(\ ,)\})^*$ that recursively satisfies the following conditions:

(i) $\Sigma \cup \{\varepsilon, \varnothing\} \subseteq E$

(ii) If $e_1, e_2 \in E$ then

(a) $(e_1 + e_2), (e_1 \cdot e_2), (e^*) \in E$

(b) $(e_1 \cap e_2), \ \overline{e} \in E$

In addition to the above operation of union, concatenation, and star as defined in (a), regular expressions which include the intersection (∩) and complement (-) operations as shown in (b) are called *extended regular expressions*.

Definition 1: *The language interpretation ι of an extended regular expression e is defined recursively as follows:*

$$\iota(e) = \begin{cases} \varnothing & \text{if } e = \varnothing \\ \{\varepsilon\} & \text{if } e \text{ is nullable} \\ \{a\} & \text{if } e = a \text{ and } a \in \Sigma \\ \iota(e_1) + \iota(e_2) & \text{if } e = (e_1 + e_2) \\ \iota(e_1) \cdot \iota(e_2) & \text{if } e = (e_1 \cdot e_2) \\ \iota(e_1) \cap \iota(e_2) & \text{if } e = (e_1 \cap e_2) \\ \iota(e_1)^* & \text{if } e = e_1^* \\ \iota(e_1) & \text{if } e = \overline{e_1} \end{cases}$$

where e, e_1 and $e_2 \in E$.

We say that an extended regular expression *e* is *nullable* if the language it represents contains the empty string, that is if $\epsilon \in \mathcal{L}(e)$. Furthermore, a language $\mathcal{L}$ is said to be nullable if $\epsilon \in \mathcal{L}$. Formally, the following is a direct consequence of the nullable definition of $\mathcal{L}$. Thus, we define the nullable interpretation η of extended regular expressions *e*, e_1, and e_2 as follows:

We can also apply η to *E*.

$$\eta(e) = \begin{cases} \{\varepsilon\} & \text{if } \varepsilon \in L(e) \\ \varnothing & \text{otherwise} \end{cases}$$

$$\eta(\varnothing) = \varnothing$$
$$\eta(\varepsilon) = \{\varepsilon\}$$
$$\eta(e_1 + e_2) = \eta(\mathcal{L}(e_1)) + \eta(\mathcal{L}(e_2))$$
$$\eta(e_1 \bullet e_2) = \eta(\mathcal{L}(e_1)) \bullet \eta(\mathcal{L}(e_2))$$
$$\eta(e_1 \cap e_2) = \eta(\mathcal{L}(e_1)) \cap \eta(\mathcal{L}(e_2))$$
$$\eta((e)^*) = \varepsilon$$

$$\eta(\bar{e}) = \begin{cases} \{\varepsilon\} & \text{if } \varnothing \in \mathcal{L}(e) \\ \varnothing & \text{if } \varepsilon \in \mathcal{L}(e) \end{cases}$$

An Algebraic Approach Language to Language Theory

Let define by the symbol $\mathcal{B}$ the Boolean semiring, $\mathcal{B} = \{0, 1\}$. Let Q be a set. Then $\mathcal{B}^Q$ is the set of all mappings of Q into $\mathcal{B}$. Note that $\hat{u} \in B^Q$ can also be considered as a Q-vector over $\mathcal{B}$. Let $\mathcal{B}(X)$ be the set of Boolean expressions over X with the usual operation symbols, that is, with $\vee$, $\wedge$,–, including the constants 0 and 1. We now introduce two additional operation symbols "+" and "·" to form an algebra a $\mathcal{T}(X, \Sigma)$ of terms over X and Σ with the following properties:

(T1) $\mathcal{T}(X, \Sigma)$ contains the two "special" elements denoted by $\emptyset$ and ϵ.

(T2) For every $w \in \Sigma^+$ and every $\hat{e} \in \mathcal{B}(X)$, $w \cdot \hat{e}$ is a term in $\mathcal{T}(X, \Sigma)$.

(T3) For every $t_1, t_2 \in \mathcal{T}(X, \Sigma)$ also $t_1 + t_2$ is a term in $\mathcal{T}(X, \Sigma)$.

(T4) For every $w \in \Sigma^*$ and every $t \in \mathcal{T}(X, \Sigma)$, $w \cdot t$ also a term in $\mathcal{T}(X, \Sigma)$.

These operation symbols satisfy the following conditions:

(T5) With respect to $+, \mathcal{T}(X, \Sigma)$ is a commutative band with $\emptyset$ as an identity element.

With respect to $\cdot, \mathcal{T}(X, \Sigma)$ is a commutative band with ϵ as an identity element

(T6) For any $w_1, w_2 \in \Sigma^*$ and any $\hat{e} \in \mathcal{B}(X)$ one has $w_1 \cdot (w_2 \cdot \hat{e}) = (w_1 \cdot w_2) \cdot (\hat{e})$ Moreover, $w_1 \cdot \varnothing = \varnothing$ and $w_1 \cdot \varepsilon = w_1$.

(T7) One $w \cdot (\hat{e}_1 \cdot \hat{e}_2) = (w \cdot \hat{e}_1) + (w \cdot \hat{e}_2)$ has and $w \cdot (t_1 + t_2) = w.t_1 + w.t_2$ for all $w \in \Sigma$ all $\hat{e}_1, \hat{e}_2 \in \mathcal{B}(X)$, and all $t_1, t_2 \in \mathcal{T}(X, \Sigma)$.

The terms in $\mathcal{T}(X, \Sigma)$ have a simple normal form as shown in the following result:

Lemma 1 *Every term t in* $\mathcal{T}(X, \Sigma)$ *can be effectively transformed into the form:*

$$\sum_{i=1}^{n} t_i$$

for some $n \in \mathrm{N}$ such that every t_i is a term of one of the forms $t_i = \varepsilon$, $t_i = w_i \cdot \varepsilon$, $t_i = \varnothing$, *or* $t_i = w_i \cdot \hat{e}_i$.

With $w_i \in \Sigma^*$ and $\hat{e}_i \in \mathcal{B}(X)$; in addition, one may assume that each $\hat{e}_i$ is a Boolean constant or a conjunction of variables or their negations. Moreover, the terms t_i are distinct and, if $t_i = \emptyset$ for some i, then $i = n = 1$.

Proof: By the construction of $\mathcal{T}(X, \Sigma)$, every term has the form Σt_i where every t_i is of one of the above forms. Assume that $t_i = w_i \cdot \hat{e}_i$ with $\hat{e}_i$ not a constant or a conjunction of variables or their negations. Then $\hat{e}_i$ can be re-written as a disjunction of conjunctions of variables and their negations. The application of the property (T7) then yields the required form. By (T5), no term occurs twice, and $\emptyset$ can be omitted as a term unless it is the only one.

Thus, we have constructed a sorted algebra as follows:

$$\left(\mathcal{T}(X, \Sigma),\ \Sigma^*, \mathcal{B}(X), \cdot, +,\ \varepsilon,\ \varnothing, \wedge, \vee, \bar{\ }, 1, 0\right)$$

which we also denote by $\mathcal{T}(X, \Sigma)$. The underlying sets of this algebra are $\mathcal{T}(X, \Sigma)$, Σ^*, and $\mathcal{B}(X)$. In this algebra one has the binary operations

$$\cdot:\ \Sigma^* \times \left(\mathcal{T}(X, \Sigma)\right) \cup \mathcal{B}(X) \to \mathcal{T}(X, \Sigma)$$
$$+ : \mathcal{T}(X, \Sigma) \times \mathcal{T}(X, \Sigma) \to \mathcal{T}(X, \Sigma)$$
$$\wedge:\ \mathcal{B}(X) \times \mathcal{B}(X) \to \mathcal{B}(X)$$
$$\vee:\ \mathcal{B}(X) \times \mathcal{B}(X) \to \mathcal{B}(X)$$

The unary operation: $\bar{\ }:\ \mathcal{B}(X) \to \mathcal{B}(X)$

and the nullary operations:

$$\varepsilon, \varnothing \in\ \mathcal{T}(X, \Sigma)\ \text{ and } 0, 1 \in\ \mathcal{B}(X)$$

We now introduce language interpretations ι_λ of elements of this algebra. Again, we use the symbol ι because we are extending the interpretation of regular expressions. Let λ be a homomorphism of the algebra $(\mathcal{B}(X), \wedge, \vee, -, 1, 0)$ into the algebra $(2^{(\Sigma^*)}, \cap, \cup, \bar{\ }, \Sigma^*, \varnothing)$ where the symbol "–" denotes complement with respect to Σ^*. The interpretation ι_λ is a homomorphism of the algebra $\mathcal{T}(X,\Sigma)$ with the operations

$\cdot, +,\ \varepsilon,\ \varnothing, \wedge, \vee, \bar{\ }, 1, 0$

into the set 2^{Σ^*} with the operations

$\cdot, \cup, \{\varepsilon\},\ \varnothing, \cap, \cup, \bar{\ }, \Sigma^*, \varnothing$

which satisfies the following:

$\iota_\lambda(w \cdot \hat{e}) = \{w\}\lambda\{\hat{e}\}$

for all w $\in \Sigma^*$ and all $\hat{e} \in \mathcal{B}(X)$. Note that any mapping $\lambda : X \to 2^\Sigma$ can be uniquely extended to a homomorphism of $\mathcal{B}(X)$ into 2^{Σ^*}, and hence, gives rise to a unique language interpretation, again denoted by ι_λ.

Alternating Finite Automata (AFA)

Alternating finite automata (AFA) exhibits the property of alternation in the following sense: If in a given state q, the automaton reads an input symbol a, it activates all states of the automaton to work on the remaining part of the input in parallel. Once the states have completed their tasks, q evaluates their results using a Boolean function and passes on the resulting value by which it was activated. A string w is accepted by an AFA if there exists some path that leads to an accepting state. More precisely, a string w is accepted if the starting state computes the values of 1. Otherwise, it is rejected. In a nondeterministic computation, all configurations are *existantial* in the sense that there exists at least one successful path that leads to acceptance. An AFA may also have *universal* configurations from which the computation branches into a number of parallel computations that must all lead to acceptance. We represent existential and universal choices by a Boolean formula. Formally, let Q be a finite set of states, we use $\mathcal{B}^Q$ to be the

set of all Boolean formulas over Q. That is, $\mathcal{B}^Q$ is built from the elements $q \in Q$, *1* and 0 using the binary operations or $(\vee)$, and $(\wedge)$, and not $(-)$.We now formalize this idea.

Definition 2 *An alternating finite automaton (AFA) is a quintuple $\mathcal{A} = (\Sigma, Q, s, F, g)$ where (a) Σ is an alphabet, the input alphabet; (b) Q is a finite set, the set of states; (c) $s \in Q$ is the starting state; (d) $F \subseteq Q$ is the set of final states; (e) g is a mapping of Q into the set of all mappings of $\Sigma \times \mathcal{B}^Q$ into $\mathcal{B}$.*

Now, we turn to defining the sequential behavior of an AFA. For $q \in Q$ and $a \in \Sigma$, let $g_q(a)$ be the Boolean function defined as:

$$g_q\ (a,\hat{u}) : \Sigma \times \mathcal{B}^Q \to \mathcal{B}$$

where $a \in \Sigma$ and $\hat{u} \in \mathcal{B}^Q$. Also, for $a \in \Sigma$, $q \in Q$ and $\hat{u} \in \mathcal{B}^Q$, $g_q(a, \hat{u}), = g_q(a)\ (\hat{u})$ is equal to either 0 or 1. Later, we also need the mappings $g\ (a)$ of Q into the set of all mappings of $\mathcal{B}^Q$ into $\mathcal{B}$ and the mappings g_p (a) of $\mathcal{B}^Q$ into $\mathcal{B}$ defined by

$$g(a)(q)(\hat{u}) = g_p(a)(\hat{u}) = g_p(a,\hat{u})$$

for $a \in \Sigma$, $q \in Q$, and $\hat{u} \in \mathcal{B}^Q$

Now define $f \in \mathcal{B}^Q$ by the condition

$$f_q = 1 \Leftrightarrow\ q \in F$$

f is called the characteristic vector of F. We extend g to a mapping of Q into the set of all mappings of $\Sigma^* \times \mathcal{B}^Q$ into $\mathcal{B}$ as follows:

$$g_q(w,\ \hat{u}) = \begin{cases} u_q & \text{if } w = \varepsilon \\ g_q(a, g(v,\hat{u}) & \text{if } w = av \text{ with } a \in \Sigma, v \in \Sigma^* \end{cases}$$

where $w \in \Sigma^*$ and $\hat{u} \in \mathcal{B}^Q$

Definition 3 *Let $\mathcal{A} = (Q, \Sigma, s, F, g)$ be an AFA. $\mathcal{A}$ string* w *$\in \Sigma^*$ is accepted by $\mathcal{A}$ if and only if $g_s(w, f)=1$. The language accepted by $\mathcal{A}$ is the set*

$$\mathcal{L}(A) = \{w \mid w \in \Sigma^* \wedge g_S(w,f) = 1\}.$$

We denote the language of $\mathcal{A}$ by $\mathcal{L}(\mathcal{A})$ and the language accepted by a state $q \in Q$ by $\mathcal{L}_q$. Note that in the same spirit as the characteristic vector of F, we extend g to languages. Thus, we define the characteristic output of $\mathcal{A}, g_{\mathcal{A}}(w,\hat{u})$, as follows:

Definition 4 Let *be $\mathcal{A} = (\Sigma, Q, s, F, g)$ an AFA and $\mathcal{L}(\mathcal{A})$ the language accepted by $\mathcal{A}$. Then, the characteristic output of Ais defined as:*

$$g_A(w,\ \hat{u}) = \begin{cases} 1 & \text{if } g_q(w,\ \hat{u}) = 1 \text{ for all } w \in L(A) \\ 0 & \text{otherwise} \end{cases}$$

Example 1 *Consider the following AFA $\mathcal{A} = (Q, \Sigma, s, F, g)$ where $Q = \{q_0, q_1, q_2\}$, Σ {a, b}, $s = \{q_0\}$, $F = \{q_2\}$, and g is given by the following Table 1.*

The existential and universal states as represented as $\vee$ *and* $\wedge$, respectively. The AFA can have multiple runs on a given input where both choices coexist. Notice that the run branches in parallel to the two states q_0 and q_2 on the second input symbol b from q_1 as shown in Table 1 and graphically in Figure 1. In addition, there are three separate mappings of $\mathcal{B}^Q$ into $\mathcal{B}$. That is, $g(q_0)$, $g(q_1)$, and $g(q_2)$ represent Boolean

Table 1. *AFA's state table*

States	a	b
q_0	$q_0 \wedge q_1$	$q_1 \vee \overline{q_2}$
q_1	q_0	$q_0 \wedge q_2$
q_2	$q_0\ \overline{q_1}$	1

Table 2: g (q_0), g (q_1), *and* g (q_2)

states			g (q_0)		g (q_1)		g (q_2)	
q_0	q_1	q_2	*a*	*b*	*a*	*b*	*a*	*b*
0	0	0	0	①	0	0	①	①
0	0	1	0	0	0	①	①	①
0	1	0	0	①	0	0	0	①
0	1	1	0	①	0	①	0	①
1	0	0	0	①	①	0	①	①
1	0	1	0	0	①	0	①	①
1	1	0	①	①	①	0	①	①
1	1	1	①	①	①	0	①	①

value of true 1 or false 0 as shown in Table 2.

Example 2 Let $w = bab$ be a string. We will check whether the input w is accepted by the above AFA $\mathcal{A}$.

$$\begin{aligned}
&g_{q0}(bab, f)\\
&= g_{q1}(ab, f) \vee \overline{g_{q2}(ab, f)}\\
&= g_{q0}(b, f) \vee \overline{\left(g_{q0}(b, f) \vee \overline{(g_{q1}(b, f))}\right)}\\
&= \left(g_{q1}(\epsilon, f) \vee \overline{g_{q2}(\epsilon, f)}\right) \vee \overline{\left(\left(g_{q1}(\epsilon, f) \vee \overline{g_{q2}(\epsilon, f)}\right) \vee \overline{\left(\left(\overline{g_{q0}(\epsilon, f)}\right) \wedge g_{q2}(\epsilon, f)\right)}\right)}\\
&= \left(0 \vee \bar{1}\right) \vee \overline{\left(\left(0 \vee \bar{1}\right) \vee \overline{\left(\left(\overline{\bar{0}} \wedge 1\right)\right)}\right)}\\
&= (0 \vee 0) \vee \overline{\left((0 \vee 0) \vee \overline{(\bar{1} \wedge 1)}\right)}\\
&= (0) \vee \overline{\left(0 \vee (\bar{1})\right)}\\
&= 0 \vee \overline{(0 \vee 0)}\\
&= 0 \vee (\bar{0})\\
&= 0 \vee 1\\
&= 1
\end{aligned}$$

By Definition 3, the characteristic output of the computation is equal to 1 and the string *bab* is accepted by the AFA $\mathcal{A}$.

We now introduce the notion of reversal AFA (r-AFA) which can be seen as an AFA except that it reads its input in reverse order. Moreover, such automata are usually used for implementing regular languages and their operations efficiently, as summarized in the next theorem. We will adapt and extend r-AFA far beyond their original scope [6, 19, 21]. That is, we will exploit such results in the context of learning

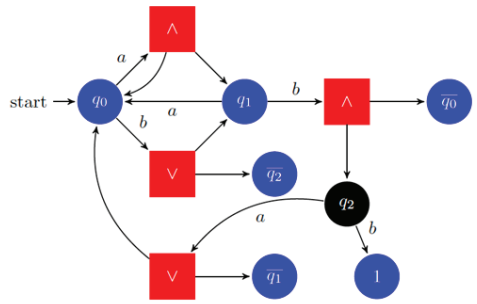


Figure 1: An alternating finite automaton (AFA).

algorithms of regular languages and automata inference as in Sections 10 and 12.

Theorem 1 (DFA, r-AFA) ([6, 21]).

If $\mathcal{L}$ is accepted by an n-state complete DFA then $\mathcal{L}$ is accepted by an r-AFA with at most $[log_2 n]$ states.

System of Language Equations: An Interpretation of AFA

Language equations are equations defined over languages where both the constants and variables are formal languages. Usually, they are formalized through various classes of finite state automata such as deterministic, nondeterministic, and universal finite automata. It is well-known that regular languages can be described as the solutions of systems of one-sided linear equations in an appropriate semiring [25, 30, 31]. These systems of equations can be easily generated by DFA or one-sided linear grammars. In this section we follow a "similar" idea and show that AFA can be readily represented by systems of equations. However, the systems of equations to be considered involve Boolean expressions over a finite set X of variables and the symbols of an alphabet $\mathcal{B}$. The main result of this section is that the solutions of such systems of equations are precisely the regular languages and that, indeed, there is a natural correspondence between AFA and such systems of equations. In the sequel, we associate with each AFA a system of equations such that the languages accepted by the AFA with various start states constitute the unique fixpoint of the system of equations. Let $\mathcal{A} = (\mathcal{B}, Q, q, F, g)$ be an AFA. For $q \in Q$, we use x_q to denote a Boolean variable associated with the state q and $\bar{x}_q$ to denote its negation. Let Xq = $\{x_q \mid q \in Q\}$. Then the following system) of equations can be used to describe $\mathcal{A}$:

$$\mathcal{L}\left(\widehat{\mathcal{A}_q}\right)=\left\{X_q=\sum_{a\in\Sigma} a\bullet\hat{g}_q(a,X)+\bar{\varepsilon}\left(f_q\right)\right\}_{q\in Q} \qquad (1)$$

where X is the vector of variables $x_q, q\varepsilon Q, g_p(a,X)$ is being given by a Boolean expression in $\mathcal{B}(X_q)$, and

$$\varepsilon\left(f_q\right)=\begin{cases}\varepsilon & if\ f_q=1\\ \varnothing & otherwise\end{cases}$$

The solution of this canonical representation of $\mathcal{L}(\mathcal{A})$, which can always be constructed via several systems of language equations, exists and uniquely defined [21,32].

The Uniqueness Solution of Language Equations

Deterministic, nondeterministic, and universal finite automata can be represented by system of language equations with two operations, union and concatenation. The relevant properties of these equations, such that existence and uniqueness of their solutions have been established in the literature [21, 32].

Theorem 2 *Let $x = e_1x + e_2$ be an equation in the x, where e_1 and e_2 are regular expression.*
Then $x = e_1^* e_2$ is the solution of the equation. Furthermore, this solution is regular if e_1 and e_2 are regular.

Proof: If $x = e_1^* e_2$ is the solution, then we have to show that

$$e_1^* e_2 = e_1 e_1^* e_2 + e_2 \qquad (2)$$

The problem of deciding whether two regular expressions are equivalent is reduced of deciding whether the two language interpretations denoted by the two sides of the equation (2) are the same. That is, $\iota\left(e_1^* e_2\right) = \iota(e_1 e_1^* e_2 + e_2)$. Assume that $w \in e_1^* e_2$, then w can be written as $w = w_1 w_2$ where $w_1 \in e_1^*$ and $w_2 \in e_2$. If $w_1 \neq \epsilon$ then $w_1 \in e_1 e_1^*$ where $e_1 e_1^* \subset e_1 e_1^* + e_2$. If $w_1 = \epsilon$ then $w = w_2 \in e_2$, where $e_2 \subset e_1 e_1^* + e_2$. Thus, the left-side hand of the equation (2) is a subset of the right-side hand.
Conversely, assume that $w = e_1 e_1^* e_2 + e_2$. If $w = e_1 e_1^* e_2$, then $w = e_1^* e_2$, and if $w \in e_2$, then w is also in $e_1^* e_2$. Therefore, in (2) the right-side hand is a subset of the left-side hand. Thus, we have proved the theorem.
The following theorem gives a sufficient condition for the uniqueness of the solution of Theorem 3.

Theorem 3 *Let $x = e_1 x + e_2$ be an equation, and $x = e_1^* e_2$ be its solution. Let $\iota(e_1)$ be the language interpretation of e_1. If $\epsilon \notin \iota(e_1)$, then the solution $x = e_1^* e_2$ is unique.*

Proof: $\epsilon \notin \iota(e_1)$ is a sufficient condition for the uniqueness of the solution $x = e_1^* e_2$. Assume that there exist two solutions x_1 and x_2 such that $x_1 \neq x_2$ which satisfy equation $x = e_1 x + e_2$. Then, we have

$$x_1 = e_1 x_1 + e_2 \qquad (3)$$

$$x_2 = e_1 x_2 + e_2 \qquad (4)$$

Since $x_1 \neq x_2$, then $\iota(x_1) \neq \iota(x_2)$. Therefore, there exists a word w such that ($w \in x_1$ *and* $\notin x_2$) or ($w \notin x_1$ *and* $w \in x_2$). Without loss of generality, assume that $w \in x_1$ and $w \in x_2$, let w be such word of the shortest length (w need not to be unique). If there exists w' $\in x_1$ such that $|w'| \langle |w|$, then $w' \in x_2$. Since $w \in x_1$ then from (3), w $\in e_1 x_1$ or $w \in e_2$. But w shouldn't belong to e_2, because, by (4) w would belong to x_2. Thus, w can be written as $w = w_1 w_2$, where $w_1 \in e_1$ and $w_2 \in x_1$. Since $\epsilon \notin e_1$, then the word $w_1 \neq \epsilon$, therefore the length of w_2 is strictly less than the length of w, in addition, we also know that $w_2 \in x_2$ since $|w_2| \langle |w|$. This implies that $w_1 w_2$, and consequently w belongs to $e_1 x_2$. Thus, w_1, w_2, and w belong to x_2 by (4). This is a contradiction with our hypothesis and the choice of w.

Corollary 1 *Let X be the set of Boolean variables, and $\iota(e_1)$ and $\iota(e_2) \subseteq \Sigma^*$ are language interpretations generated by e_1 and e_2 respectively. The language equation h as a solution $X = \iota(e_1) X + \iota(e_2)$. Furthermore, the solution is unique if $\epsilon \notin \iota(e_1)$.*

The above Corollary is an extension of Arden's rule [33] and Theorem 3 to language equations.

Example 3 *Given the* NFA $\mathcal{A} = (Q, \Sigma, Q_0, F, \delta)$, *where* $Q = \{x_1, x_2, x_3\}$, $\Sigma = \{a, b\}$, $Q_0 = x_1$, $F = \{x_3\}$ and *is given by the following system of language equations. The regular language* $\mathcal{L}(\mathcal{A})$ *generated by* $\mathcal{A}$ *is obtained by solving the following system of language equations. Let* $\mathcal{A}$ *be an* NFA *as depicted below.*

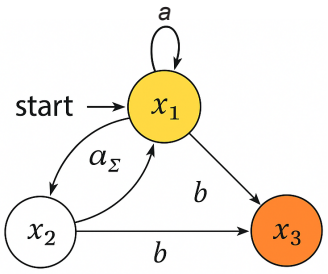


Figure 2: $\mathcal{L}(\mathcal{A}) = \mathcal{L}_x = (a + a\Sigma)^*(ab + b)$

$$\begin{aligned}\mathcal{L}_{x1} &= a\mathcal{L}_{x1} - a\mathcal{L}_{x2} + b\mathcal{L}_{x3}\\ \mathcal{L}_{x2} &= \Sigma\mathcal{L}_{x1} + b\mathcal{L}_{x3}\\ \mathcal{L}_{x3} &= \epsilon\end{aligned}$$

Using backward propagation yields the solution $\mathcal{L}_{x1}$ which is obtained by using a series of substitutions of Theorem 3 and Corollary 1 as follows:

$$\begin{aligned}\mathcal{L}_{x1} &= a\mathcal{L}_{x1} + a\mathcal{L}_{x2} + b\\ \mathcal{L}_{x2} &= \Sigma\mathcal{L}_{x1} + b\\ \mathcal{L}_{x1} &= a\mathcal{L}_{x1} + a(\Sigma\mathcal{L}_{x1} + b) + b\\ \mathcal{L}_{x1} &= a\mathcal{L}_{x1} + a\Sigma\mathcal{L}_{x1} + ab + b\\ \mathcal{L}_{x1} &= (a + a\Sigma)\mathcal{L}_{x1} + ab + b\\ \mathcal{L}_{x1} &= (a + a\Sigma)^*(ab + b)\end{aligned}$$

Derivatives of Regular Expressions and Regular Languages

The notion of derivative regular expressions has been introduced by Brzozowski [34] for finding the quotient of regular expressions and providing corresponding derivatives and their auxiliary functions. Let e be an extended regular expression, and u is a string over Σ^*. We denote by $\partial_u(e)$ the derivative of e with respect to u, which is formally defined as follows:

Definition 5 *The derivative of an extended regular expression* e *with respect to a string* $u \in \Sigma^*$ *is defined to as:* $\partial_u(e) = v \in \Sigma^* \mid uv \in e$ *and* $\partial_u(\mathcal{L}) = \{v \in \Sigma^* \mid wv \in \mathcal{L}\}$.

Intuitively, $\partial_u(e)$ is the set of all remaining strings obtainable from e by taking off the prefix u, if possible. The derivatives of an extended regular expression e with respect to a symbol $a \in \Sigma$ are defined as follows:

Theorem 4

$$\begin{aligned}\partial_a(\phi) &= \varnothing\\ \partial_a(\varepsilon) &= \varnothing\\ \partial_a(a) &= \varepsilon\\ \partial_a(b) &= \varnothing \text{ if } b \neq a\\ \partial_a(e_1 + e_2) &= \partial_a(e_1) + \partial_a(e_2)\\ \partial_a(e_1^*) &= \partial_a(e_1)e_1^*\\ \partial_a(e_1 \cap e_2) &= \partial_a(e_1) \cap \partial_a(e_2)\\ \partial_a(\overline{e_1}) &= \overline{\partial_a(e_1)}\end{aligned}$$

$$\partial_a(e_1 \bullet e_2) = \begin{cases}\partial_a(e_1)e_2 + \partial_a(e_2) & \text{if } e_2 \text{ is nullable}\\ \partial_a(e_1)e_2 & \text{otherwise}\end{cases}$$

The last equation takes the symbol a off of the first expression e_1 or from the second regular expression e_2 if $\mathcal{L}(e_1)$ is *nullable* (*i.e.,* the empty string $\epsilon \in \mathcal{L}(e_1)$.

Example 4 *Let* $\Sigma = \{a, b\}$.

- *Let* $e = a(ab)^*$. Then the derivatives of e with respect to a and b respectively are $\partial_a(e) = (ab)^*$ *and* $\partial_b(e) = \phi$.
- *Let* $e = a(ab + a)^*ba$. Then the derivative of e with respect to a using auxiliary properties and calculations is:

$$\begin{aligned}\partial_a(e) &= \partial_a\left((ab + a)^* ba\right)\\ &= \partial_a\left((ab + a)^*\right)ba + \partial_a(ba)\\ &= \partial_a((ab + a))(ab + b)^* ba + \partial_a(b)a\\ &= (\partial_a(ab) + \partial_a(a))(ab + a)^* ba + \partial_a(b)a\\ &= (b + \varepsilon)(ab + a)^* ba + \partial_a(b)a\\ &= (b + \varepsilon)(ab + a)^* ba + \phi\\ &= (b + \varepsilon)(ab + a)^* ba.\end{aligned}$$

In a similar manner, we can compute the derivative of e with respect to b, that is, $_b(e)$. In addition, the concept of derivatives applies to languages. For a language $\mathcal{L}$, the derivative of $\mathcal{L}$ with respect to a string w is the set of remaining strings after having read w from any string in $\mathcal{L}$, formally defined as follows:

Definition 6 *The derivative of a language* $\mathcal{L} \subseteq \Sigma^*$ *with respect to a string* $w \in \Sigma^*$ *is defined by* $\partial_w(\mathcal{L}) = \{v \in \Sigma^* \mid wv \in \mathcal{L}\}$.

The following two properties are a consequence from the definition of the derivative.

$$\begin{aligned}\partial_\delta(\mathcal{L}) &= \mathcal{L}\\ \partial_{ua}(\mathcal{L}) &= \partial_a(\partial_u(\mathcal{L}))\end{aligned}$$

where $w \in \Sigma^*$ and $\in \Sigma$.

Proposition 1 *Let* $a \in \Sigma, w \in \Sigma^*$, *and* $e \in E$. *Then* $aw \in \mathcal{L}(e)$ *if* $w \in \mathcal{L}(\partial_a(e))$ *and* $\epsilon \in \mathcal{L}(e_1)$. *if and only if nullable* (e).

Learning Regular Languages Through Residuality

In this section, we show that residualizing canonical AFA through a generalized reversal alternation methodology leads to the construction of DFA that recognizes regular languages. Residuality is considered a natural distillation of the essence

of the automaton's states language recognition. Importantly, it adds a foundational meaning for a better understanding of regular language learning algorithms and computational learning theory. Residual finite state automata (RFA) [1] are a subclass of NFA where each state represents a residual language of the language that is accepted by the automaton. The class of RFA lies between DFA and NFA, and they share in common a number of significant properties. For instance, RFA share with NFA the existence of automata that are exponentially smaller, in terms of the number of states, than the corresponding minimal DFA for the same language. For more details, see Introduction, Section 1.

Definition 7 *A residual finite automaton* (*RFA*) $\mathcal{R} = (\Sigma, Q, s, F, \delta)$ *is a nondeterministic finite automaton* (*NFA*) *where for every state* $q \in Q$, $\mathcal{L}(\mathcal{R}_q)$ is a residual language $\in \mathcal{L}(\mathcal{R})$.

The class of RFA lies between DFA and NFA, and they share in common a number of significant properties. For instance, RFA share with NFA the existence of automata that are exponentially smaller, in terms of the number of states, than the corresponding minimal DFA for the same language. These properties make RFA particularly appealing in severalareas of computer science such as pattern recognition, computational biology, and software verification. Moreover, residual languages play an important role in many state machine inference algorithms, particularly in identifying residual languages and actively learning regular languages from queries and counterexamples.

Let $\mathcal{A} = (Q, \Sigma, q, F, \delta)$ be a finite state automaton. The language $\mathcal{L}(\mathcal{A})$ is the set of all accepted strings by $\mathcal{A}$. For a state $q \in Q$, we write $\mathcal{A}_q$ for the automaton that starts in the configuration (i.e., state) q.

Definition 8 *A language* $\mathcal{L}_q \in \Sigma^*$ *is a residual language of* $\mathcal{L}$ *if there is* $w \in \Sigma^*$ *such that* $\mathcal{L}_q = \partial_w(\mathcal{L})$, where $\partial_w(\mathcal{L}) = \{v \in \Sigma^* \mid wv \in \mathcal{L}\}$.

For notational purpose, we use $\mathcal{L}_q$ to indicate the residual

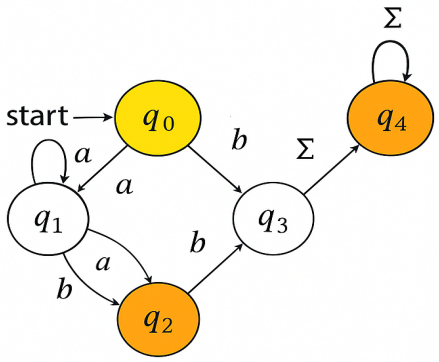


Figure 3: $\mathcal{L}_q(\mathcal{A}) = \mathcal{L}_{q_0} = a^+b + (a^+b)^+(\varepsilon + b\Sigma^+) + b\Sigma^+$

language of state q. Let $\mathcal{A} = (Q, \Sigma, Q_0, F, \delta)$ be a DFA.

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $Q_0 = \{q_0\}$, $F = \{q_2, q_3\}$ and δ as shown in the following graph in Figure 3.

Now, we derive a simple set of residual languages of $\mathcal{L}$, denoted by

$Res(\mathcal{L})$, $\partial\varepsilon(\mathcal{L}) = \mathcal{L}_{q_0}$, $\partial_a(\mathcal{L}) = \mathcal{L}_{q_1}$, $\partial_\alpha(\mathcal{L}) = \mathcal{L}_{q_2}$, $\partial_\beta(\mathcal{L}) = \mathcal{L}_{q_4}$.

where α and $\beta \in \Sigma^*, \alpha = a^+b$ and β

An automaton $\mathcal{A}$ accepting a language $\mathcal{L}$ is residual if every state q of $\mathcal{A}$ corresponds to a residual language (equivalent to residual in this paper). However, the reverse is not always true. That is, not every residual language should be accepted by a unique state. To this end, several states may accept the same residual language. Consequently, we categorize the set of residual languages as prime and composed residual languages, formally defined as follows:

Definition 9 *A residual language* $\mathcal{L}'$ *with respect to a string* $w \in \Sigma^*$ *is called prime* $\{\partial_v(\mathcal{L}) \mid \partial_v(\mathcal{L}) \subsetneq \partial_w(\mathcal{L})\} \subsetneq \partial_w(\mathcal{L})$. *Otherwise,* $\mathcal{L}'$ *is called composed. In other words* $\mathcal{L}'$ *is a residual prime if there are* $\mathcal{L}_1, \mathcal{L}_2 \ldots \mathcal{L}_n \in Res(\mathcal{L}) \setminus \{\mathcal{L}'\}$ *such that* $\mathcal{L}' \neq \mathcal{L}_1 \cup \mathcal{L}_2 \cup \ldots \cup \mathcal{L}_n$.

Definition 10 *A residual alternating finite* automata (*RAFA*) $\hat{} = (\Sigma, Q, s, F, g)$. *is an alternating finite automaton* (*AFA*) *such that for every state* $q \in Q$, $\mathcal{L}(\widehat{\mathcal{A}_q})$ *is a residual language* $\in \mathcal{L}(\hat{\mathcal{A}})$.

Similar to AFA and better suited than RFA, RAFA (Residual Alternating Finite Automata) are double-exponentially more succinct than DFA, making them the preferable automaton model to work with in practical learning algorithms. We will show in the remaining of the paper that RAFA is double exponentially than the size of the corresponding minimal DFA but are usually even considerably smaller and easier to learn.

Given a RAFA $\hat{\mathcal{A}} = (\Sigma, Q, s, F, g)$. recognizing the set of regular languages, $Res(\mathcal{L})$, a state $q \in Q$ is called *prime* if it recognizes a prime residual language of $\mathcal{L}$. That is, $\mathcal{L}(\mathcal{A}) \in$ Prime ($\mathcal{L}$). A residual is called composed, if it is the non-trivial union of other residuals. Otherwise, it is called prime residual language.

Lemma 2 *Let* $\hat{\mathcal{A}} = (\Sigma, Q, s, F, g)$ *be a RAFA. For every prime residual language* $\mathcal{L}\partial_w(\mathcal{L}(\hat{\mathcal{A}})$, *there exists a state* $q \in Q$ *such that* $\mathcal{L}(\widehat{\mathcal{A}_q}) = \partial_w(\mathcal{L}(\hat{\mathcal{A}})$.

Residual Language Equations

In this section, we extend the aforementioned results of the native AFA in two directions: First, we consider language equations over Boolean operations and show the existence of solutions. Second, of special interest are the semantics of residual alternating finite automata in terms of residual languages over finite words. Let $\hat{\mathcal{A}} = (\Sigma, Q, s, F, g)$ be a residual alternating finite automaton (RAFA) and let $X = (X_0, \ldots, X_q)$. We use X_q to denote a Boolean variable associated with the state q and $\overline{x_q}$ to denote its negation. Let $X_q = \{x_q \mid q \in Q\}$. Then the following collection of language equations, $\mathcal{L}(\widehat{\mathcal{A}_q})$, identify the set of residual languages of $\mathcal{L}$.

$$\mathcal{L}(\widehat{\mathcal{A}_q}) = \left\{ X_q = \sum_{a \in \Sigma} a \bullet \hat{g}_q(a, X) + \hat{\varepsilon}(f_q) \right\}_{q \varepsilon Q} \quad (5)$$

$$\hat{\epsilon}(f_q)=\begin{cases}\varepsilon & if\ (f_q)=1\\ \varnothing & otherwise\end{cases}$$

where $\hat{g}_q(a,X)$ is the set-operation interpretation of the Boolean function $g(a,X), q\in Q,$ and $a\in\Sigma$.

Similarly to $\mathcal{L}(\mathcal{A})$ the language $\mathcal{L}(\hat{\mathcal{A}})$ is the set of all accepted strings by $\mathcal{A}$. For a state $q\in Q,$, we write $\mathcal{A}_q$ for the automaton $\mathcal{A}=(\Sigma, Q,q,F,g)$ that starts in the configuration q instead of s. Let $\mathcal{L}$ and $w\in\Sigma^*$, we denote by the derivative of $\mathcal{L}$ with respect to w. $L'\in\Sigma^*$ is a residual language of $\mathcal{L}$ if there is $w\in\Sigma^*$ such that $\mathcal{L}'=\partial_w(\mathcal{L})$. Then the following theorem describes the system of residual language equations:

Theorem 5 *Let $\hat{\mathcal{A}}=(\Sigma,Q,s,F,g)$ be a residual alternating finite automaton (RAFA), $Res(\mathcal{L})$ the set of residual languages of $\hat{\mathcal{A}}$, and $\{X_q\}_{q\in Q}$ be the solution of the language equations. Then the following claims hold:*

$$(i)\ \mathcal{L}(\hat{\mathcal{A}})=X_s$$
$$(ii)\ \mathcal{L}(\hat{\mathcal{A}}_q)=X_q$$
$$(iii)\ \mathcal{L}(\hat{\mathcal{A}}_q)=\partial_w(\mathcal{L}(\hat{\mathcal{A}}))=\partial_w(X_q)$$
$$(iv)\ Res(\mathcal{L})=\partial_w(\mathcal{L})=\partial_w(\mathcal{L}(\hat{\mathcal{A}}))=\partial_w(X_s)$$

Proof: (i) Let $\hat{\mathcal{A}}_q=(\Sigma,Q,s,F,g),\ q\in Q$ We prove that $\mathcal{L}(\hat{\mathcal{A}}_q)=X_q$ for all $q\in Q$. Since $\hat{\mathcal{A}}_s=\hat{\mathcal{A}},\ \mathcal{L}(\hat{\mathcal{A}}_s)=X_s$. We prove this for any $w\in\Sigma^*$, if and only if $g_q(\epsilon,f)=1$ by induction on the length of w. If $|w|=0$. i.e., $w=\epsilon$, then $\epsilon\in X_q$ if and only if $g_q(\epsilon,f)=1$ because $g_q(\epsilon,f)=f_q$ and by definition $\hat{\varepsilon}(fq)=\varepsilon$ if and only if $f_q=1$.

Assume that any $w\in\Sigma^*$ such that $|w|<k, w\in X_q$ if and only if $g_q(\epsilon,f)=1$ Consider $w\in\Sigma^*$ such that $|w|<k$, where $k>0$. Let $w=aw_0$. $g_q(w,f)=1$ if and only if $g_q(a,u)=1$ and $u=g(w_0,f)$. By induction hypothesis, $u_p=1$ if and only if $w_0\in X_q$ for all $p\in Q$. Therefore, $g_q(w,f)=1$ if and only if $w_0\in g_q(a,u)$ and if and only if $w\in X_q$.

The proof of each of the above claims, (ii), (iii), and (iv) follow in turn the same induction pattern than that of (i). That is, on the length of the string w and in terms of the derivative of a language. That is, the set $\partial_w(\mathcal{L}(\hat{\mathcal{A}}_q))$ which is the set of stings in L with the prefix w. The last claim asserts a set of characteristic equations where X_s is exactly the language accepted by $\hat{\mathcal{A}}$.

The most significant property of residual automata (RFA) is that it performs the semantics of each state independently, which makes RFA appealing in several areas of research in computer science. In the grammatical inference and finite state automata settings, residuality underlies the seminal algorithm L* for learning deterministic automata [3] and building other efficient algorithms for learning nondeterministic and alternating automata, NL and AL*. Now, we illustrate and compile the system of residual language equations for the AFA from Table 1 and Figure 1.

$$\mathcal{L}_q=a(\mathcal{L}_{q0}\wedge\mathcal{L}_{q1})+b(\mathcal{L}_{q1}\vee\mathcal{L}_{\overline{q}})$$
$$\mathcal{L}_{q1}=a\mathcal{L}_{q0}+b(\mathcal{L}_{\overline{q_0}}\wedge\mathcal{L}_{q2})$$
$$\mathcal{L}_{q2}=a(\mathcal{L}_{q0}\vee\mathcal{L}_{\overline{q_1}})+b1+\varepsilon$$

Reversal-Alternating State Machine Framework for Learning Residual Languages

Learning regular sets from queries and counterexamples by [3] forms the basis of many modern state machine inference algorithms. With respect to the minimization of RAFA, we consider a special kind of RAFA that we call s-RAFA. An s-RAFA is an RAFA $\hat{\mathcal{A}}=(\Sigma,Q,s,F,g)$ such as every a $a\in\Sigma$ and every $\hat{u}\in B^Q,$, $g_q(a,\hat{u})$, does not depend on bus. Intuitively, this means that the starting states cannot be reached in any computation. Obviously, for every RAFA one construct an equivalent s-RAFA which has just one more state. On the other hand, if $\hat{\mathcal{A}}$ is $a(k+1)$ state s-RAFA then there exists an equivalent k-state RAFA. For example, the language $\{\varepsilon,a,a^2\}$ is accepted by a 3-state s-RAFA but not by any 2-state RAFA. The s-RAFA are particularly useful to simplify certain constructions of regular language learning algorithms.

Theorem 6 *Let $\mathcal{L}$ be a language and $\mathcal{L}^r$ be the reverse of $\mathcal{L}$. $\mathcal{L}$ is accepted by an s-RAFA* with $k+1$ states if and only if $\mathcal{L}^r$ is accepted by a *DFA* with 2^k states.

Proof: We show the proof by construction. Let $\mathcal{D}=(Q_{\mathcal{D}},\Sigma,s_{\mathcal{D}},F_{\mathcal{D}},\delta)$. be a 2^k-state DFA. Let $K=\{1,2,\ldots,k$ and $K_0=K\cup\{0\}$. Without loss of generality, we assume that $Q_{\mathcal{D}}=\mathcal{B}^K$ and $s_{\mathcal{D}}=\{0,\ldots,0\}$. For $\hat{u}\in\mathcal{B}_0^K$, let $\widehat{u'}\in\mathcal{B}^K$ be defined by $\widehat{u'}=\hat{u}$ for all $i\in K$. Now, we define a (k + 1)-state s-RAFA $\hat{\mathcal{A}}=(Q_{\hat{\mathcal{A}}},\Sigma,s_{\hat{\mathcal{A}}},F_{\hat{\mathcal{A}}},g)$ by

$$Q_{\hat{\mathcal{A}}}=K_0$$
$$s_{\hat{\mathcal{A}}}=0$$
$$F_{\hat{\mathcal{A}}}=\begin{cases}\{0\} & \text{if } s_{\mathcal{D}}\in F_{\hat{\mathcal{A}}}\\ 0 & \text{otherwise}\end{cases}$$
$$f(x)=\begin{cases}\delta(\hat{u}',a)_i & \text{if } i=0 \text{ and } \hat{u}'\in F_{\mathcal{D}}\\ 1 & \text{if } i=0 \text{ and } \hat{u}'\notin F_{\mathcal{D}}\end{cases}$$

For $i\in K_0$, $a\in\Sigma$ and $\hat{u}\in\mathcal{B}_0^K$. The function g is well defined since $\hat{\mathcal{A}}$ is an alternating residual finite automaton. By induction on the length of $w\in\Sigma^*$, one shows that $\hat{u}=g(w,f)$ if and only if $\delta_{\mathcal{D}}(s_{\mathcal{D}},w^r)=\widehat{u'}$. Since $\widehat{u_0}=1$ if and only if $\widehat{u'}\in F_{\mathcal{D}}$, and $w\in\mathcal{L}(\hat{\mathcal{A}})$ if only if $w^r\in\mathcal{L}(\mathcal{D})$.

Corollary 2 *For any RAFA, there exists an equivalent s-RAFA having at most one additional state.*

For RAFA, we adopt the concept of r-AFA introduced in Section 5, inspired by the work of [19,21,32], and extend to RAFA in this paper. Now, we introduce a variant of s-RAFA called rs-RAFA (with "r" indicating reversal and "s" indicating that the starting state cannot be reached in any computation). The rs-RAFA operates similarly to an s-RAFA, but with the input string read in reverse The rs-RAFA framework serves as the foundation for the learning algorithm designed for RAFA. The following theorem naturally follows from the results presented earlier in Theorem 1, Theorem 6, and Corollary 2.

Theorem 7 *For each language $\mathcal{L}$ that is accepted by a DFA with n states, there exists an equivalent rs-RAFA with at most* $1+[(\log_2 n)]$ *states.*

We now formulate a new paradigm for inferring a state machine model learning algorithm, residual alternating finite automata represented as a system of residual language equations, which uses active state machine learning algorithms for learning regular sets. rs-RAFA $\widehat{R}=(\Sigma,Q,s,F,g)$ can be described naturally as a set of residual language equations that parallels the solutions of algebraic equations. Moreover, the solution of such systems of residual equations is the class of regular languages. Then the following system $\mathcal{L}(\hat{\mathcal{R}})$ of residual language equations can be used to describe $\hat{\mathcal{R}}$:

$$Res(\mathcal{L})=\left\{X_q=\sum_{a\varepsilon\Sigma} a.g_q(a,X)+\varepsilon(f_q)\right\}_{q\varepsilon Q}$$
$$\varepsilon(f_q)=\begin{cases}\varepsilon & if\ (f_q)=1\\ \varnothing & otherwise\end{cases} \quad (6)$$

In the system $Res(\mathcal{L})$ of equations, $g_q(a,X)$ and have been defined earlier. The Boolean function $g_q(a,X)$ is considered as being given by a Boolean expression in B^{X_q}. Any system of residual language equations of the above form has a unique solution for each X_q, $q\in Q$. Furthermore, the solution for each X_q is regular. The system of equations (6) corresponds to the set of residual language equations of $\mathcal{L}$. That is, each residual language equation exactly corresponds to the states of ˆ . That is, there is a bijection between residual language equations of $\mathcal{L}$ and the states of the minimal rs-RAFA.

Angluin's-Style L* Learning algorithm

We briefly review the classical automata learning algorithm L* by Angluin [3]. The algorithm L* demonstrates that the class of regular language could be learned efficiently by fully constructing the minimal DFA. $\mathcal{A}$ for a given regular language $\mathcal{L}$ such that $\mathcal{L}(\mathcal{A})=\mathcal{L}$. Such a minimal DFA is learned using membership and equivalence queries between a learner and teacher. To do this, the learner may ask the teacher, who knows about the language, two types of classical queries:

Membership Queries: The learner selects a word $w\in\Sigma^*$ in the target language and the teacher replies whether or not $w\in\mathcal{L}$.

Equivalence Queries: Does a given hypothesis automata ($\mathcal{H}$) recognize the target language? That is, whether $\mathcal{L}$=$\mathcal{L}(\mathcal{H})$? The learner selects a hypothesis automaton $\mathcal{H}$, and the teacher answers whether or not $\mathcal{L}$ is the language of $\mathcal{H}$. If yes, then the algorithms terminate; otherwise, the teachers provides a counterexample, which is a word in the symmetric difference of $\mathcal{L}$ and $\mathcal{L}(\mathcal{H})$.

The learner maintains an observation table T (*rows, cols*) over two finite sets $S, E\subseteq\Sigma^*$ respectively. For any $u\in$ and $u\ a\in\Sigma$, $T(u,\ a)=1$ if and only if the word $ua\in\mathcal{L}$. In other words, T has one complete row of derivatives. Intuitively, each row of T approximates a derivative of the target language $\mathcal{L}$. However, the content of T may be incomplete because no membership queries have not been asked yet for some words w. The table T is *closed* if one-letter derivatives are in the table. Intuitively, membership query words are used by the learner to identify the different derivatives of the target language $\mathcal{L}$, enabling the construction of an automaton from T. Broadly, based on the observed behavior, the learner can infer a model of the canonical DFA for $\mathcal{L}$, by formulating a polynomial set of membership queries. It is important to highlight that the L* algorithm encapsulates the essence of innovation, and it is the first of its kind [3]. The class of regular languages could be learned efficiently (*i.e.,* in time polynomial in the size of the canonical DFA for the language $\mathcal{L}$. Many implementation details are omitted in this section, and we will be discussed and exploited in the next sections.

Revisiting L* through Residual Language Learning and Reversal Alternation

Indeed, while regular languages may pose challenges for learning using residual alternating finite automata (RAFA), this paper focuses on a particular subtype of RAFA known as rs-RAFA. This specialized form of RAFA, with its properties of reversal alternation and residuality, can be efficiently learned along the analogical lines of L*. The combination of reversal alternation and residuality in rs-RAFA renders them exponentially more concise than DFA making them valuable tools for learning regular languages.

Definition 11 *Given a learner-teacher-expert (LTEx) framework capable of answering classical and reversal membership and equivalence queries, the active learning task is to construct the minimal RAFA (rs-RAFA) and consequently deriving the minimal DFA for an unknown regular language $\mathcal{L}$ over $\sum$ in polynomial time.*

Now, we describe an enrichment of the standard framework L* for learning regular languages by introducing additional information and various types of queries. Specifically, we introduce an extension of membership and equivalence queries known as reversal membership and equivalence queries. In these queries, the input is consumed in reverse, and the correct- ness of the reversed hypothesized RAFA $\mathcal{H}$ is assessed. Theorems 6 and 7 provide further details and insights into these extensions. A learner wants to learn a regular language $\mathcal{L}$ over a fixed alphabet $\sum$ represented by a specific type of finite automata, namely residual alternating finite automata (RAFA). This learning framework model is featured by a teacher and an expert who define the active learnability according to their capabilities and access to information about the target language. The RAL* active learning algorithm is distinguished by its capability for query refinement, facilitated by the teacher's ability to seek assistance from the expert to fulfill the request of the learner.

Importantly, we distinguish between the roles of the teacher and expert. The teacher plays an intermediate role by refining and forwarding the learner's queries to the expert. While the teacher can address basic membership and equivalence queries, it also has access to supplementary knowledge from the expert. In comparison to the teacher, the expert answers reversal

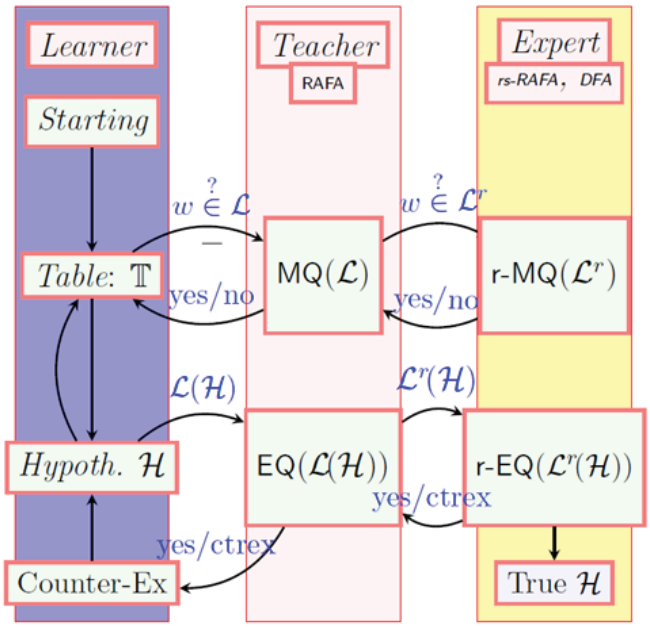


Figure 4: *The LTEx trilateral learning framework.*

membership query r-MQ and reversal equivalence query (r-EQ). The expert provides accurate and insightful responses to queries related to reverse operations and equivalence between different types of automata, namely rs-RAFA and DFA.

Ask To this end, the learner repeatedly makes queries to the teacher, typically operating in a black-box fashion and accessing the language in question through the expert. Unlike the standard version of Angluin's algorithm, which forms the basis of L* the active learning process in our framework involves a trilateral interaction between three components: the learner, teacher, and expert. This interaction is summarized in four mappings: (a) a membership query between the learner and teacher, (b) a reversal membership query between the teacher and expert, (c) an equivalence query between the learner and teacher, and (d) a reversal equivalence query between the teacher and expert. The teacher has access to the language in question through the expert and can answer two different types of queries: membership query (MQ) and equivalence query (EQ). The expert has access to the language in question through the conversion of rs-RAFA and DFA and can answer two different types of queries: reversal membership query (r-MQ) and reversal equivalence query (r-EQ). These results are stated in Theorems 6 and 7, depicted in Figure 4, and highlighted in Algorithm 1.

Algorithm 1 Learner-Teacher-Expert (LTEx).

(a) Learner-Teacher: The membership query $MQ(\mathcal{L})$ consists in asking the teacher if a word $w \in \Sigma^* \in \mathcal{L}$ (RAFA). The teacher reformulates the request and forwards it to the expert who replies *"yes" or "no"* depending on whether $w \in \mathcal{L}^r$ or not.

(b) Teacher-Expert: The reversal membership query is r-MQ consists in asking the expert if a word $w \text{ in } \Sigma^* \in \mathcal{L}$ (RAFA) (Theorems 6 and 7). The experts *"yes" or "no"* depending on whether $w \in L^r$ (RAFA) or *"not"*

(c) Learner-Teacher consists in asking the teacher whether a hypothesis RAFA $\mathcal{L}(\mathcal{H})$ is correct, i.e., whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}$ The teacher answers *"yes"* $\mathcal{L}(\mathcal{H})$ of is correct or returns a counterexample.

(d) Teacher-Expert: The reversal equivalence query r-EQ consists in asking the expert to use the relationship between RAFA (*i.e.*, rs-RAFA) and DFA). That is, whether $\mathcal{L}^r(H) = \mathcal{L}$ (RAFA). The expert answers *"yes"* if $\mathcal{H}$ is correct or returns a counter example.

The learner interactively refines the learner's request by querying the expert. The expert, possessing complete knowledge, helps generalize the partial knowledge of the teacher, who guides the learner. Consequently, the RAL* algorithm produces an RAFA that is isomorphic to the canonical minimal RAFA of the target language $\mathcal{L}$. Throughout the learning process, the learner maintains and updates two sets: a prefix-closed set $\mathcal{U} \subseteq \Sigma^*$ containing candidate words for identifying states, and a suffix-closed set $\mathcal{V} \subseteq \Sigma^*$ containing words used to distinguish these states. Both sets, u and v, contain the empty string ϵ. By issuing membership queries, the learner determines whether all strings $uv \in \mathcal{U}\mathcal{V}$ or $ua v \in \mathcal{U}\Sigma\mathcal{V}$ belong to the language $\mathcal{L}$. The results are organized into an observation table $T = (\mathcal{U}, \mathcal{V}, \mathcal{T})$ for $\mathcal{L}$, where $\mathcal{T}$ is a mapping function defined as follows:

$$\mathcal{T}(x) = \begin{cases} (U \cup U\Sigma) \times V \to \{\text{"yes"}\}) & \text{if } w \text{ is accepted} \\ (U \cup U\Sigma) \times V \to \{\text{"no"}\} & \text{if } w \text{ is not accepted} \end{cases}$$

where $w \in (\mathcal{U} \cup \mathcal{U}\Sigma)\mathcal{V}$. The table $\mathbb{T}$ is of dimension $|(\mathcal{U} \cup \mathcal{U}\Sigma)| \times |\mathcal{V}|$ where the elements entries of both rows and columns are {"yes", "no"}. The values of each entry are the outcome of a membership query for the concatenation of the row and column strings. Let $u \in (\mathcal{U} \cup \mathcal{U}\Sigma)$, we associate with every u a function $row(u): \mathcal{V} \to \{\text{"yes"}, \text{"no"}\}$ such that $row(u)(v) = \mathcal{T}(uv)$. We call such a function row of u and the set of all rows is denoted by *Rows* $(\mathbb{T})$. Now, define by $Rows_{upp}(\mathbb{T})$ *and* $Rows_{low}(\mathbb{T})$ the upper and lower parts of $\mathbb{T}$, respectively. That is, $Rows_{upp}(\mathrm{T}) = \{row(u) \mid u \in \mathcal{U}\}$ and $Rows_{low}(\mathrm{T}) = \{row(u) \mid u \in \mathcal{U}\Sigma\}$. Rows labeled by the elements of C are the candidates for states of the automaton being constructed, and columns labeled by the elements of $\mathcal{V}$ correspond to distinguishing experiments for these states. Rows labeled by elements of Σ are used to construct the transition function.

Closedness and Consistency

The table $\mathbb{T}$ is closed if for all $u \in \mathcal{U}$ and $a \in \Sigma$, there is $u' \in \mathcal{U}$ such that $row(ua) = row(u')$

The table $\mathbb{T}$ is consistent if for all $u, u' \in \mathcal{U}$ and $a \in \Sigma$, there is $u' \in \mathcal{U}$ such that $row(u) = row(u')$ which implies that $row(ua) = row(u'a)$.

The primary objective of active learning is to find a rs-RAFA of minimal size that adequately satisfies the learner's membership and equivalence queries. Below the algorithm 2, is the pseudo-code outlining the learning algorithm for RAL*.

Algorithm 2 RAL* Algorithm

Input: $MQ(\mathcal{L})$ and $EQ(\mathcal{L})$, target language: $\mathcal{L} \subseteq \Sigma^*$
Output: RAFA
Prefix-closed Set: $\mathcal{U} \subseteq \Sigma^* = \{\varepsilon\}$;
Suffix-closed Set: $\mathcal{V} \subseteq \Sigma^* = \{\varepsilon\}$;
Initialize the observation table: $\mathbb{T} = (\mathcal{U}, \mathcal{V}, \mathcal{T})$ for $\mathcal{L}$;
repeat
  while T is *not closed* and *not consistent* **do**
    (i) The *learner-teacher* perform $MQ(\mathcal{L})$
    (ii)The *teacher-expert* perform $r-MQ(\mathcal{L}')$
    if $\mathbb{T}$ *is not closed* **then**
      find $u, u' \in \mathcal{U}, v \in \mathcal{V}, a \in \Sigma$ such that
      $row(u) = row(u')$ and
      $row(ua) \neq row(u'a)(v)$;
      $\mathcal{V} \rightarrow \mathcal{V} \cup \{av\}$;;
      update($\mathbb{T}$);
    end if
    if $\mathbb{T}$ is *not consistent* **then**
      find $u \in \mathcal{U}$ and $a \in \Sigma$ such that
      $row(ua) \neq row(u')$;
      $\mathcal{U} \rightarrow \mathcal{U} \cup \{ua\}$;
      update($\mathbb{T}$);
    end if
  end while
until Equivalence Query Succeeds
Comments: /* $\mathbb{T}$ is both closed and consistent; */
Comments: /* $\mathcal{H}$ Construct $\mathcal{H}$ */;
Comments: /* Perform the equivalence queries with the hypothesis $\mathcal{H}$ */;
(i) The *learner-teacher* perform $EQ(\mathcal{L}(\mathcal{H}))$;
(i i)The *teacher-expert* perform $r-EQ\mathcal{L}'((\mathcal{H}))$;

if the reverse equivalence query, $r-EQ\mathcal{L}'$, fails **then**
  the *expert-teacher* return a *counterexample* u;
  $\mathcal{U} \rightarrow \mathcal{U} \cup prefix(u)$;
  update($\mathbb{T}$);
end if

If the table $\mathbb{T}$ is closed and consistent the learner constructs hypothesized RAFA $\mathcal{H} = (\Sigma, Q, s, F, g)$ where

- Q is the set of states, $Q = \{row(u) \mid u \in \mathcal{U}\}$
- q_0 is the starting states, $q_0 = \{row(\lambda)\}$
- F is the set of final states, $F = \{q \in Q \mid q(\lambda) = +\}$
- g is the transition function, $(row(u), a) = row(ua), row(u) \in Q \text{ and } a \in \Sigma)$

In the RAL*, the learner eventually submits $\mathcal{H}$ as an equivalence query to the teacher inquiring whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}$. It is imperative that the observation table maintains the properties of closeness and consistency throughout the learning process. If at some step $\mathbb{T}$ violates closeness, this is solved by adding the string u' into the set $\mathcal{U}\Sigma$. Similarly, if $\mathbb{T}$ is inconsistent, this is solved by adding a into the set $\mathcal{V}$. If an automaton fails an equivalence query, the teacher must provide a counterexample — a word that highlights a discrepancy between the conjectured automaton and the target language.

Conclusion

In this paper, we have investigated a new framework called LTEx which incorporates two theoretical metaphors known as reversal alternation and residuality. The first is a generalization of nondeterminism and the second is a discernment of linguistic facts from the semantics of the state of RAFA where each state represents a residual language. This has led to a new online active algorithm called residual reversal-alternating (RAL*), viewed through the lens of L*. Furthermore, we exploit the succinct mappings between rs-RAFA, RAFA and DFA to develop RAL*.

The logic of such mappings involves a set of membership and equivalence queries, which are referred to as MQ, EQ, r-MQ and r-EQ, between the learner, teacher, and expert. By utilizing the concept of residuality, we also introduced residual language equations that precisely correspond to the states of rs-RAFA. That is, there is a bijection between the residual language equations of L and the states of the minimal rs-RAFA. Such models can be naturally described as a set of residual language equations that mirror the solutions of algebraic equations. Furthermore, the solution of such systems of residual language equations belongs to the class of regular languages. The results of this paper extend the current research on learning regular languages and have highlighted some issues for further investigation, such as grammatical inference learning.

Conflicts of Interest

The authors declare no conflicts of interest.

References

1. Denis F, Lemay A, Terlutte A. Residual finite state automata. In: Middeldorp A, Deussen O, eds. Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS '10). Lecture Notes in Computer Science, vol 5xx. Springer; 2010:144–157.
2. Denis F, Lemay A, Terlutte A. Learning regular languages using RFSAs. Theoretical Computer Science. 2004;313(2):267–294.
3. Angluin D. Learning regular sets from queries and counterexamples. Information and Computation. 1987;75(2):87–106.
4. An J, Zhan B, Zhan N, Zhang M. Learning nondeterministic real-time automata. ACM Transactions on Embedded Computing Systems. 2021;20(5):1–26. doi:[insert DOI].
5. Berndt S, Liskiewicz M, Lutter M, Reischuk R. Learning residual alternating automata. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence; 2017:1749–1755.
6. Fellah A. Revising and reexamining Angluin's algorithm: implications for unified regular language learning algorithms. In: Proceedings of the 5th International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI); 2023:178–184.
7. Bolling B, Habermehl P, Kern C, Leucker M. Angluin-style learning of NFA. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI); 2019:1004–1009.
8. Chubachi K, Hendrian D, Yoshinaka R, Shinohara A. Query learning algorithm for residual symbolic finite automata. In: Proceedings of the 10th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF); 2019:140–153.
9. Ganty P, Gutiérrez PE, Valero P. A quasiorder based perspective in residual automata. In: Esparza J, Krail D, eds. Proceedings of the

45th International Symposium on Mathematical Foundations of Computer Science (MFCS ’20); 2020: [page numbers]. Springer.

10. Moerman J, Sammartino M. Residual nominal automata: residuality and learning for nondeterministic nominal automata. Journal of Logical and Algebraic Methods in Programming. 2022;18(1):1–28.
11. Tsunoo E, Kashiwagi Y, Narisetty C, Watanabe S. Residual language model for end to end speech recognition. In: Proceedings of the International Speech Communication Association Conference; 2022:1–5.
12. Chandra AK, Kozen DC, Stockmeyer LJ. Alternation. Journal of the ACM. 1981;28(1):114–133.
13. Kavitha J, Jeganathan L, Sethuraman G. Descriptional complexity of alternating finite automata. In: Proceedings of the 8th International Workshop on Descriptional Complexity of Formal Systems (DCFS). 2006;44:188–198.
14. Antoni LD, Kincaid Z, Wang F. A symbolic decision procedure for symbolic alternating finite automata. Electronic Notes in Theoretical Computer Science. 2018;336:790–799.
15. Yu S. Regular languages. In: Salaomaa A, ed. Handbook of Formal Languages. Vol I. Springer-Verlag; 2021:41–110.
16. Place T, Zeitoun M. Going higher in first order quantifier alternation hierarchies on words. Journal of the ACM. 2019;66(2):1–33.
17. Barloy C, Cadilhac M, Paperman C, Zeume T. The regular languages of first order logic with one alternation. In: Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS ’22). ACM/IEEE; 2022:1–11.
18. Dan S, Bastani O, Dan R. Understanding robust generalization in learning regular languages. In: Proceedings of the 39th International Conference on Machine Learning (ICML ’22); 2022:1–14.
19. Fellah A, Jurgensen H, Yu S. Constructions for alternating finite automata. International Journal of Computational Mathematics. 1990;35(1–4):117–132.
20. Fellah A, Bandi A. Learning language equations and regular languages using alternating finite automata. Journal of Computing Sciences in Colleges. 2019;35(2):19–28.
21. Yu S. State complexity of regular languages. Journal of Automata, Languages and Combinatorics. 2001;6:221–234.
22. Angluin D, Eisenstat S, Fisman D. Learning regular languages via alternating automata. International Journal of Foundations of Computer Science. 2015;25(6):781–802.
23. Berndt S, Liskiewicz M, Lutter M, Reischuk R. Learning residual finite automata. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI ’17); 2017:1749–1755.
24. Angluin D. Learning regular sets from queries and counterexamles. Info. Comput. 1987;75(2):87–106. [Duplicate of #3—consider removing one]
25. Fellah A. Real time languages, timed alternating automata, and timed temporal logics: relationships and specifications. Procedia Computer Science. 2015;62:47–54.
26. Kupferman D. Automata theory and model checking. In: Baier C, Katoen JP, eds. Handbook of Model Checking. Springer; 2018:157–172.
27. Chiari M, Mandrioli D, Pontiggia P, Pradella M. A model checker for operator precedence languages. ACM Transactions on Programming Languages and Systems. 2023;45(3):1–32.
28. Vaandrager FW. Model learning. Communications of the ACM. 2017;60(2):86–95.
29. Moerman J, Sammartino M. Residual nominal automata. In: Proceedings of the 31st International Conference on Concurrency Theory (CONCUR ’20). Lecture Notes in Computer Science, vol 171. Schloss Dagstuhl–Leibniz-Zentrum für Informatik; 2020:44:1–44:21.
30. Brzozowski JA, Leiss E. On equations for regular languages, finite automata, and sequential networks. Theoretical Computer Science. 1980;10:19–35.
31. Baader F, Okhotin A. On language equations with one sided concatenation. Fundamenta Informaticae. 2013;126(1):1–34.
32. Fellah A. Equations and regular like expressions for AFA. International Journal of Computational Mathematics. 1994;51(3–4):157–172.
33. Arden DN. Delayed logic and finite state machines. In: Proceedings of the 2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT); 1961:133–151.
34. Brzozowski JA. Derivatives of regular expressions. Journal of the ACM. 1964;11(4):481–494.