Angluin-Style Learning of NFA*

Benedikt Bollig
LSV, ENS Cachan, CNRS
bollig@lsv.ens-cachan.fr

Peter Habermehl
LIAFA, U. Paris Diderot (Paris 7)
Peter.Habermehl@liafa.jussieu.fr

Carsten Kern
MOVES, RWTH Aachen
kern@cs.rwth-aachen.de

Martin Leucker
Institut für Informatik, TU Munich
leucker@in.tum.de

Abstract

We introduce NL*, a learning algorithm for inferring non-deterministic finite-state automata using membership and equivalence queries. More specifically, residual finite-state automata (RFSA) are learned similarly as in Angluin's popular L* algorithm, which, however, learns deterministic finite-state automata (DFA). Like in a DFA, the states of an RFSA represent residual languages. Unlike a DFA, an RFSA restricts to prime residual languages, which cannot be described as the union of other residual languages. In doing so, RFSA can be exponentially more succinct than DFA. They are, therefore, the preferable choice for many learning applications. The implementation of our algorithm is applied to a collection of examples and confirms the expected advantage of NL* over L*.

1 Introduction

Learning automata has a wide field of applications ranging over robotics and control systems, pattern recognition, computational linguistics, computational biology, data compression, data mining, etc. (see [de la Higuera, 2005] for an excellent survey). Recently, learning techniques have also become popular in the area of automatic verification. They have been used [Leucker, 2007] for *minimizing* (partially) specified systems and for model checking *black-box systems*, proved helpful in *compositional model checking* and in *regular model checking*. To put it bluntly, automata learning is en vogue.

The general goal of learning algorithms employed in verification is to identify a *machine*, usually of *minimal* size, that *conforms* with an *a priori fixed* set of strings or a given machine. Nearly all algorithms learn *deterministic finite-state automata* (DFA) or deterministic finite-state machines (Mealy-/Moore machines), as the class of DFA has preferable properties in the setting of learning. For every regular language, there is a unique minimal DFA accepting it, which can be characterized thanks to Nerode's right congruence. This characterization is at the base of most learning algorithms.

In general, two types of learning algorithms for DFA can be distinguished, so-called *online* and *offline* algorithms. Offline algorithms get a fixed set of *positive* and *negative* examples, comprising strings that should be *accepted* and, respectively, strings that should be *rejected* by the automaton in question. The learning algorithm now has to provide a (minimal) automaton that accepts the positive examples and rejects the negative ones. For deriving minimal automata, major achievements are due to Biermann [Biermann and Feldman, 1972], Trakhtenbrot and Barzdin [Trakhtenbrot and Barzdin, 1973]. Efficient algorithms inferring a *not necessarily minimal* DFA are given in [Lang, 1992] and [Oncina and Garcia, 1992] under the name *RPNI*.

Online algorithms have the possibility to ask further *queries*, i.e., whether some string is in the language of the automaton to learn or not. In this way, an online algorithm can enlarge the set of examples as needed.

A popular setup for an online approach is that of Angluin's L* algorithm [Angluin, 1987]: A minimal DFA is learned based on *membership* and *equivalence queries*. We have a *learner* whose job is to come up with the automaton to learn, a *teacher* who may answer if a given string is in the language as well as an *oracle* answering if the automaton hypothesis currently proposed by the learner is correct or not.

DFA have a serious drawback for applications in verification. In general, a DFA might be exponentially bigger than a *non-deterministic finite-state automaton* (NFA). For many applications, it would be a huge improvement to work with an exponentially more succinct NFA rather than the corresponding DFA. As such, learning algorithms for NFA are needed. However, the class of NFA lacks important properties that are essential for current learning algorithms: There is no unique minimal NFA for a given regular language and there is no characterization of NFA in terms of right-congruence classes.

In a seminal paper, Denis et al. [Denis *et al.*, 2002] introduce the class of *residual finite-state automata* (RFSA). It is a subclass of NFA that shares important properties with the class of DFA: For every regular language, there is a unique minimal canonical RFSA accepting it. The states of this automaton correspond to right-congruence classes, or, equivalently, to *residuals* of the accepted language. At the same time, the RFSA can be exponentially more succinct than the corresponding DFA. As such, RFSA are the preferable class for learning regular languages. In [Denis *et al.*, 2004], Denis et al. provided an offline algorithm, called DeLeTe2, which works in the spirit of RPNI. Alternatives and extensions to

*Work supported by DAAD/EGIDE (Procope 2008/2009).

this algorithms have then been presented, most recently in [García *et al.*, 2008], which also gives a nice overview on offline algorithms for learning NFA.

In this paper, we introduce NL* as an online learning algorithm for RFSA, patterned after L*. Using membership and equivalence queries, our algorithm infers a (minimal) canonical RFSA for the language in question, which is always smaller than or equal to the corresponding DFA. Note that [Yokomori, 1994] presents an online learning algorithm for NFA based on *contradiction backtracking*. According to [Denis *et al.*, 2004], the resulting NFA are actually RFSA. They are, however, not canonical and can even be larger than the corresponding minimal DFA, as their size crucially depends on the size of counterexamples provided by the *oracle*.

We have implemented our algorithm and studied its efficiency on a collection of regular languages described by regular expressions. It turns out that, for most examples, the resulting RFSA is much smaller than the corresponding DFA. While our current upper bounds for NL* on the number of membership and equivalence queries are slightly worse than those for L*, our implementation shows that the resulting RFSA are typically obtained after far fewer queries. Summarizing, we provide a new learning algorithm for regular languages. Together with a practically efficient oracle realization based on *anti-chains* [De Wulf *et al.*, 2006], we expect it to enhance, e.g., associated verification tasks considerably.

Full proofs can be found in [Bollig *et al.*, 2008].

2 Preliminaries

We fix a finite set Σ of letters, called *alphabet*. Finite sequences of letters are elements of Σ^*, called *words*. Subsets of Σ^* are termed *languages*. For $w \in \Sigma^*$, we denote by $Pref(w)$ (resp. $Suff(w)$) the set of its prefixes (resp. suffixes) including w itself and the empty word ϵ. A *non-deterministic finite-state automaton* (NFA) $\mathcal{A} = (Q, Q_0, F, \delta)$ has a finite set of *states* Q, a set of *initial states* $Q_0 \subseteq Q$, a set of *final states* $F \subseteq Q$, and a *transition function* $\delta : Q \times \Sigma \to 2^Q$. $\mathcal{A}$ is a *deterministic finite-state automaton* (DFA) if $|Q_0| = 1$ and $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. As usual, δ is extended to $\bar{\delta} : Q \times \Sigma^* \to 2^Q$ by $\bar{\delta}(q, \epsilon) = \{q\}$ and $\bar{\delta}(q, aw) = \bigcup_{q' \in \delta(q,a)} \bar{\delta}(q', w)$, and to sets $Q' \subseteq Q$ by $\hat{\delta}(Q', w) = \bigcup_{q \in Q'} \bar{\delta}(q, w)$. We use δ to denote both $\bar{\delta}$ and $\hat{\delta}$. For $q \in Q$, let $L_q = \{w \in \Sigma^* \mid \delta(q, w) \cap F \neq \emptyset\}$. The *language* $L(\mathcal{A})$ *accepted* by $\mathcal{A}$ is $\bigcup_{q \in Q_0} L_q$. Two automata are *equivalent* if they accept the same language. It is folklore that the languages of finite-state automata are the *regular* ones. We call a DFA *minimal* if there is no equivalent DFA with strictly fewer states. In contrast to NFA, a DFA has always a unique minimal representative, as shown by Myhill/Nerode.

Residual Finite-State Automata RFSA, introduced in the seminal work [Denis *et al.*, 2002], are a subclass of NFA inheriting some desirable features of DFA. Important for learning, every regular language is accepted by a canonical RFSA with a minimal number of states. As this does not hold for arbitrary NFA, it seems difficult to come up with learning algorithms for the whole class of NFA. At the same time, like for NFA, RFSA can be exponentially more succinct than DFA.

Technically, RFSA and DFA have the property that the states of the automata correspond to so-called *residual languages* defined below. This is in general not true for NFA.

Definition 1 (Residual Language) *For $L \subseteq \Sigma^*$ and $u \in \Sigma^*$, we denote by $u^{-1}L$ the set $\{v \in \Sigma^* \mid uv \in L\}$. A language $L' \subseteq \Sigma^*$ is a* residual language *(or, simply,* residual*) of L if there is $u \in \Sigma^*$ with $L' = u^{-1}L$. We denote by $Res(L)$ the set of residual languages of L.*

The Myhill-Nerode theorem states that the number of residuals of a language is finite iff this language is regular [Nerode, 1958]. Moreover, for a minimal DFA $\mathcal{A}$, there is a natural bijection between its states and the residual languages of $L(\mathcal{A})$.

Definition 2 (Residual Finite-State Automaton) *A* residual finite-state automaton *(RFSA) is an NFA $\mathcal{R} = (Q, Q_0, F, \delta)$ such that, for each $q \in Q$, $L_q \in Res(L(\mathcal{R}))$.*

In other words, each state accepts a residual language of $L(\mathcal{R})$, but not every residual language must be accepted by a *single* state. Intuitively, the states of an RFSA form a class of states of the corresponding minimal DFA. Yet, using nondeterminism, certain states of a minimal DFA are not needed as they correspond to the union of languages of other states. To this end, we distinguish *prime* and *composed* residuals:

Definition 3 (Prime and Composed Residuals) *Let $L \subseteq \Sigma^*$. A residual $L' \in Res(L)$ is called* composed *if there are $L_1, \ldots, L_n \in Res(L) \setminus \{L'\}$ such that $L' = L_1 \cup \ldots \cup L_n$. Otherwise, it is called* prime*. The set of prime residuals of L is denoted by $Primes(L)$.*

We can now define the *canonical* RFSA of a regular language. The idea is that its set of states corresponds exactly to its prime residuals. Moreover, the transition function should be *saturated* in the sense that a transition to a state should exist if it does not change the accepted language.

Definition 4 (Canonical RFSA) *Let $L \subseteq \Sigma^*$ be a regular language. The canonical RFSA of L, denoted by $\mathcal{R}(L)$, is the NFA (Q, Q_0, F, δ) where $Q = Primes(L)$, $Q_0 = \{L' \in Q \mid L' \subseteq L\}$, $F = \{L' \in Q \mid \epsilon \in L'\}$, and $\delta(L_1, a) = \{L_2 \in Q \mid L_2 \subseteq a^{-1}L_1\}$.*

Note that $\mathcal{R}(L)$ is indeed an RFSA and we actually have $L(\mathcal{R}(L)) = L$. We say that an RFSA $\mathcal{R}$ is *canonical* if it is the canonical RFSA of $L(\mathcal{R})$.

Angluin's Learning Algorithm L* The algorithm from [Angluin, 1987] learns or infers a minimal DFA for a given regular language L. A so-called *Learner*, who initially knows nothing about L, is trying to learn a DFA $\mathcal{A}$ such that $L(\mathcal{A}) = L$. To this end, it repeatedly asks queries to a *Teacher* and an *Oracle*, who both know L. There are two kinds of queries: A *membership query* consists of asking the *Teacher* if a string $w \in \Sigma^*$ is in L, and an *equivalence query* consists of asking the *Oracle* whether a *hypothesized* DFA $\mathcal{H}$ is correct, i.e., whether $L(\mathcal{H}) = L$. The *Oracle* answers *yes* if $\mathcal{H}$ is correct, or, otherwise, supplies a counterexample w, drawn from the symmetric difference of L and $L(\mathcal{H})$.

The *Learner* maintains a prefix-closed set $U \subseteq \Sigma^*$ of words that are candidates for identifying states, and a suffix-closed set $V \subseteq \Sigma^*$ of words used to distinguish those

states. Words of U are called *access strings* and words of V *experiments*. U and V are both initialized to $\{\epsilon\}$ and increased when needed. The *Learner* makes membership queries for all words in $(U \cup U\Sigma)V$, and organizes the results into a *table* $\mathcal{T} = (T, U, V)$ where function T maps each $w \in (U \cup U\Sigma)V$ to an element from $\{+, -\}$. Here, $+$ means *accepted* and $-$ *not accepted*. To $u \in U \cup U\Sigma$, we assign a function $row(u) : V \to \{+, -\}$ given by $row(u)(v) = T(uv)$. Any such function is called a *row* of $\mathcal{T}$, and the set of all rows of a table is denoted by $Rows(\mathcal{T})$. We let $Rows_{\text{upp}}(\mathcal{T}) = \{row(u) \mid u \in U\}$ denote the set of rows representing the "upper" part of the table. The rows from $Rows_{\text{low}}(\mathcal{T}) = \{row(u) \mid u \in U\Sigma\}$ occur in its "lower" part. Table $\mathcal{T}$ is *closed* if, for all $u \in U$ and $a \in \Sigma$, there is $u' \in U$ such that $row(ua) = row(u')$. It is *consistent* if, for all $u, u' \in U$ and $a \in \Sigma$, $row(u) = row(u')$ implies $row(ua) = row(u'a)$.

If $\mathcal{T}$ is not closed, then we find $u' \in U\Sigma$ such that $row(u) \neq row(u')$ for all $u \in U$. We move u' to U and ask membership queries for every $u'av$ where $a \in \Sigma$ and $v \in V$. Likewise, if $\mathcal{T}$ is not consistent, then we find $u, u' \in U$, $a \in \Sigma$, and $v \in V$ such that $row(u) = row(u')$ and $row(ua)(v) \neq row(u'a)(v)$. In that case, we add av to V and ask membership queries for every $u''av$ with $u'' \in U \cup U\Sigma$. When $\mathcal{T}$ is closed and consistent, the *Learner* constructs a hypothesized DFA $\mathcal{H} = (Q, Q_0, \delta, F)$ where $Q = Rows_{\text{upp}}(\mathcal{T}) = \{row(u) \mid u \in U\}$, $Q_0 = \{row(\epsilon)\}$, δ is defined by $\delta(row(u), a) = \{row(ua)\}$, and $F = \{r \in Q \mid r(\epsilon) = +\}$. Then, the *Learner* submits $\mathcal{H}$ to an equivalence query (asking whether $L(\mathcal{H}) = L$). If the answer is *yes*, the learning procedure is completed. Otherwise, the returned counterexample u is processed by adding every prefix of u (including u) to U, extending $U\Sigma$ accordingly, and performing membership queries to make the table closed and consistent, whereupon a new hypothesized DFA is constructed, etc.

Remark 1 L* *can be modified by changing the treatment of counterexamples. Instead of adding the counterexample and its prefixes to U, one can add the counterexample and all its suffixes to V ensuring that the table is* always *consistent [Maler and Pnueli, 1995].*

3 Learning of Residual Finite-State Automata

We will now modify Angluin's learning algorithm L*, which infers DFA, towards learning of NFA in terms of RFSA.

From Tables to RFSA To simplify our presentation, we follow Angluin's notions and notation. We use tables $\mathcal{T} = (T, U, V)$ with a prefix-closed set of words U, a suffix-closed set V, and a mapping $T : (U \cup U\Sigma)V \to \{+, -\}$. We associate with a word $u \in U \cup U\Sigma$ a mapping $row(u) : V \to \{+, -\}$. Members of U are used to reach states and members of V to distinguish states. We adopt notations introduced before such as $Rows(\mathcal{T})$, $Rows_{\text{upp}}(\mathcal{T})$, and $Rows_{\text{low}}(\mathcal{T})$.

The main difference in the new approach is that *not all rows* of the table will correspond to states of the hypothesized automaton, but only certain *prime* rows. Essentially, we have to define for rows what corresponds to *'union'*, *'composed'*, *'prime'*, and *'subset'* previously introduced for languages.

Definition 5 (Join Operator) *Let $\mathcal{T} = (T, U, V)$ be a table. The* join *$(r_1 \sqcup r_2) : V \to \{+, -\}$ of rows $r_1, r_2 \in Rows(\mathcal{T})$ is defined, for $v \in V$, by $(r_1 \sqcup r_2)(v) = r_1(v) \sqcup r_2(v)$ where $- \sqcup - = -$ and $+ \sqcup + = + \sqcup - = - \sqcup + = +$.*

Note that the join operator is associative, commutative, and idempotent, yet that the join of two rows is *not* necessarily a row of table $\mathcal{T}$.

Definition 6 (Composed, Prime Rows) *Let $\mathcal{T} = (T, U, V)$ be a table. A row $r \in Rows(\mathcal{T})$ is called* composed *if there are rows $r_1, \ldots, r_n \in Rows(\mathcal{T}) \setminus \{r\}$ such that $r = r_1 \sqcup \ldots \sqcup r_n$. Otherwise, r is called* prime*. The set of prime rows in $\mathcal{T}$ is denoted by $Primes(\mathcal{T})$. Moreover, we let $Primes_{\text{upp}}(\mathcal{T}) = Primes(\mathcal{T}) \cap Rows_{\text{upp}}(\mathcal{T})$.*

Definition 7 (Covering Relation) *Let $\mathcal{T} = (T, U, V)$ be a table. A row $r \in Rows(\mathcal{T})$ is* covered *by row $r' \in Rows(\mathcal{T})$, denoted by $r \sqsubseteq r'$, if, for all $v \in V$, $r(v) = +$ implies $r'(v) = +$. If, moreover, $r' \neq r$, then r is* strictly covered by *r', denoted by $r \sqsubset r'$.*

As for L*, we now define concepts comparable to closedness and consistency called RFSA-closedness and RFSA-consistency. For DFA, closedness ensures that every row in the lower part also occurs in the upper part. For RFSA, this translates to the idea that each row of the lower part of the table is composed of (prime) rows from the upper part.

Definition 8 (RFSA-Closedness) *A table $\mathcal{T} = (T, U, V)$ is called* RFSA-closed *if, for each $r \in Rows_{\text{low}}(\mathcal{T})$, $r = \bigsqcup\{r' \in Primes_{\text{upp}}(\mathcal{T}) \mid r' \sqsubseteq r\}$.*

Note that a table is RFSA-closed iff any prime row of the lower part is a prime row of the upper part of the table.

The idea of consistency in the DFA case was as follows: Assume that two words u and u' of the table have the same row. This suggests that both words lead to the same state of the automaton to learn as they cannot be distinguished by words $v \in V$. Hence, they induce the same residuals. Then, however, ua and $u'a$ have to induce equal residuals as well, for any $a \in \Sigma$. In other words, if there is some $a \in \Sigma$ and $v \in V$ such that $T(uav) \neq T(u'av)$, then the residuals induced by u and u' cannot be the same and must be distinguishable by the suffix av to be added to V.

For RFSA, if there are u and u' with $row(u) \sqsubseteq row(u')$, then this suggests that the residual induced by u is a subset of the residual induced by u'. If indeed so, then the same relation must hold for the successors ua and $u'a$.

Definition 9 (RFSA-Consistency) *A table $\mathcal{T} = (T, U, V)$ is called* RFSA-consistent *if, for all $u, u' \in U$ and $a \in \Sigma$, $row(u') \sqsubseteq row(u)$ implies $row(u'a) \sqsubseteq row(ua)$.*

One might expect that the previous simple definitions smoothly lead to a straightforward extension of L* towards learning RFSA. This is, however, not the case. We now list some major difficulties that arise. First, an RFSA-closed and RFSA-consistent table does generally not represent a canonical RFSA, not even an RFSA. To put things right, we come up with an algorithm that produces NFA as intermediate hypotheses and outputs a canonical RFSA only in the last step. Second, termination of our algorithm crucially depends on the treatment of counterexamples, which need to be added to the

set of suffixes held in a table. A third subtle point is that the number of states may *decrease* during a run of the algorithm. Thus, the termination proof requires reasoning on a measure that relates four indicators of a table such as the number of distinct rows or the number of upper prime rows.

So let us first associate an NFA to an RFSA-closed and RFSA-consistent table. As above-mentioned, we show that this NFA corresponds to a canonical RFSA only after our learning algorithm has terminated (Theorems 1 and 2).

For the rest of this subsection, we fix an RFSA-closed and RFSA-consistent table $\mathcal{T} = (T, U, V)$.

Definition 10 (NFA of a Table) *We define $\mathcal{R}_\mathcal{T}$ to be the NFA (Q, Q_0, F, δ) with $Q = Primes_{\text{upp}}(\mathcal{T})$, $Q_0 = \{r \in Q \mid r \sqsubseteq row(\epsilon)\}$, $F = \{r \in Q \mid r(\epsilon) = +\}$, and $\delta(row(u), a) = \{r \in Q \mid r \sqsubseteq row(ua)\}$ for $u \in U$ with $row(u) \in Q$, $a \in \Sigma$.*

Note that $Primes_{\text{upp}}(\mathcal{T}) = Primes(\mathcal{T})$, as $\mathcal{T}$ is RFSA-closed. Also, δ is well-defined: Take u, u' with $row(u) = row(u')$. Then, $row(u) \sqsubseteq row(u')$ and $row(u') \sqsubseteq row(u)$. Consistency implies $row(ua) \sqsubseteq row(u'a)$ and $row(u'a) \sqsubseteq row(ua)$ so that both resulting rows are the same.

Let us establish some elementary properties of $\mathcal{R}_\mathcal{T}$. The proof of the first two lemmas is by simple inductions.

Lemma 1 *Let $\mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta)$. For all $u \in U$ and $r \in \delta(Q_0, u)$, we have $r \sqsubseteq row(u)$.*

Lemma 2 *Let $\mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta)$. For each $r \in Q$ and $v \in V$, we have (1) $r(v) = -$ iff $v \notin L_r$, and (2) $row(\epsilon)(v) = -$ iff $v \notin L(\mathcal{R}_\mathcal{T})$.*

Lemma 2 may be summarized by saying that each *state* of $\mathcal{R}_\mathcal{T}$ correctly classifies strings of V. This fact will enable us to prove that the covering relation precisely reflects language inclusion, as stated in the next lemma.

Lemma 3 *Let $\mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta)$. For every $r_1, r_2 \in Q$, $r_1 \sqsubseteq r_2$ iff $L_{r_1} \subseteq L_{r_2}$.*

Proof: Let $r_1, r_2 \in Q$ and assume $u_1, u_2 \in U$ with $row(u_1) = r_1$ and $row(u_2) = r_2$. "only if": Suppose $r_1 \sqsubseteq r_2$ and $w \in L_{r_1}$. We distinguish two cases. Assume first $w = \epsilon$. Then, $row(u_1)(\epsilon) = +$ and, due to $r_1 \sqsubseteq r_2$, $row(u_2)(\epsilon) = +$. Thus, $r_2 \in F$ so that $\epsilon \in L_{r_2}$. Now let $w = aw'$ with $a \in \Sigma$. We have $\delta(r_1, aw') \cap F \neq \emptyset$. Thus, there is $r \in \delta(r_1, a)$ such that $\delta(r, w') \cap F \neq \emptyset$. From $r_1 \sqsubseteq r_2$, we obtain, by RFSA-consistency, $row(u_1a) \sqsubseteq row(u_2a)$. By definition of δ, $r \sqsubseteq row(u_1a)$, which implies $r \sqsubseteq row(u_2a)$. Thus, $r \in \delta(r_2, a)$ and we have $aw' \in L_{r_2}$. "if": Assume $r_1 \not\sqsubseteq r_2$. We show that $L_{r_1} \not\subseteq L_{r_2}$. By definition of $\sqsubseteq$, there exists $v \in V$ with $row(u_1)(v) = +$ but $row(u_2)(v) = -$. By Lemma 2, $v \in L_{r_1}$ and $v \notin L_{r_2}$. Therefore, $L_{r_1} \not\subseteq L_{r_2}$. □

The automaton $\mathcal{R}_\mathcal{T}$ constructed from $\mathcal{T}$ is not necessarily an RFSA [Bollig *et al.*, 2008]. However, we show that $\mathcal{R}_\mathcal{T}$ is a canonical RFSA *if it is consistent with* $\mathcal{T}$, i.e., if it correctly classifies all words from $\mathcal{T}$.

Definition 11 *$\mathcal{R}_\mathcal{T}$ is* consistent with *$\mathcal{T}$ if, for all words $w \in (U \cup U\Sigma)V$, we have $T(w) = +$ iff $w \in L(\mathcal{R}_\mathcal{T})$.*

The next lemma is a stronger version of Lemma 1, if we additionally have that $\mathcal{R}_\mathcal{T}$ is consistent with $\mathcal{T}$.

Lemma 4 *If $\mathcal{R}_\mathcal{T} = (Q, Q_0, F, \delta)$ is consistent with $\mathcal{T}$, then, for all $u \in U$ with $row(u) \in Q$, we have $row(u) \in \delta(Q_0, u)$.*

Proof: Suppose $row(u) \notin \delta(Q_0, u)$. With Lemma 1, we have $\forall r \in \delta(Q_0, u).r \sqsubseteq row(u)$. Then, Lemma 3 implies $\forall r \in \delta(Q_0, u).L_r \subseteq L_{row(u)}$. As $row(u) \in Q$ and $row(u) \notin \delta(Q_0, u)$, there is $v \in V$ such that $row(u)(v) = +$ and, for all $r \in \delta(Q_0, u)$, $r(v) = -$. This, with Lemma 2, implies $\forall r \in \delta(Q_0, u).v \notin L_r$. But then, $uv \notin L(\mathcal{R}_\mathcal{T})$, a contradiction to the fact that $\mathcal{R}_\mathcal{T}$ is consistent with $\mathcal{T}$. □

Theorem 1 *Let $\mathcal{T}$ be RFSA-closed and RFSA-consistent and let $\mathcal{R}_\mathcal{T}$ be consistent with $\mathcal{T}$. Then, $\mathcal{R}_\mathcal{T}$ is a canonical RFSA.*

Proof: Let $\mathcal{T} = (T, U, V)$ and assume $\mathcal{R}_\mathcal{T} = (Q, q_0, F, \delta)$. Set $L = L(\mathcal{R}_\mathcal{T})$. We first prove that $\mathcal{R}_\mathcal{T}$ is an RFSA. Let $u \in U$ with $row(u) \in Q$. Let us show $L_{row(u)} = u^{-1}L$. By Lemma 4, we have $row(u) \in \delta(Q_0, u)$, which implies $L_{row(u)} \subseteq u^{-1}L$. By Lemma 1, $\forall r \in \delta(Q_0, u).r \sqsubseteq row(u)$. Thus, with Lemma 3, $\forall r \in \delta(Q_0, u).L_r \subseteq L_{row(u)}$. This gives $u^{-1}L \subseteq L_{row(u)}$. With $L_{row(u)} \subseteq u^{-1}L$, we have $L_{row(u)} = u^{-1}L$. As, by Lemma 3, the relation $\sqsubseteq$ over rows corresponds to the subset relation over languages, $L_{row(u)}$ is prime and the transition function δ is saturated. □

The Algorithm We now describe NL*, which takes a regular language $L \subseteq \Sigma^*$ as input. Its pseudo code is given in Algorithm 1. After initializing the table $\mathcal{T}$, the it is repeatedly checked for RFSA-closedness and RFSA-consistency. If the algorithm detects a prime row $row(ua)$ that is not contained in $Primes_{\text{upp}}(\mathcal{T})$ (a violation of the RFSA-closedness condition from Def. 8), then ua is added to U. This involves additional membership queries. On the other hand, whenever the algorithm perceives an RFSA-consistency violation (Def. 9), then a suffix av can be determined that makes two existing rows distinct or incomparable. In this case, a column is added to V invoking supplemental queries. This procedure is repeated until $\mathcal{T}$ is RFSA-closed and RFSA-consistent. If both properties are fulfilled, a conjecture $\mathcal{R}_\mathcal{T}$ can be derived from $\mathcal{T}$ (cf. Def. 10), and either a counterexample u from the symmetric difference of L and $L(\mathcal{R}_\mathcal{T})$ is provided and $Suff(u)$ added to V invoking NL*, or the learning procedure terminates successfully. Note that the algorithm ensures that V is always suffix-closed and U prefix-closed.

Remark 2 *The counterexamples are handled as described in Remark 1, since treating them as in* L* *leads to a non-terminating algorithm [Bollig* et al.*, 2008]. Our treatment of counterexamples ensures that each row can appear at most once in the upper part of the table.*

Proving termination of L* is quite straightforward. In our setting, however, the termination proof is intricate as, after an equivalence query or a violation of RFSA-consistency, the number of states of the hypothesized automaton does not necessarily increase [Bollig *et al.*, 2008].

The following theorem constitutes our main contribution.

Theorem 2 *Let n be the number of states of the minimal DFA $\mathcal{A}^*$ for a given regular language $L \subseteq \Sigma^*$. Let m be the length of the biggest counterexample returned by the equivalence test (or 1 if the equivalence test always succeeds). Then,*

Algorithm 1 NL* (Σ; regular language $L \subseteq \Sigma^*$)

initialize $\mathcal{T} := (T, U, V)$ for $U = V = \{\epsilon\}$ by memb. queries
REPEAT
  WHILE $\mathcal{T}$ is not (RFSA-closed and RFSA-consistent) **DO**
    IF $\mathcal{T}$ is not RFSA-closed **THEN**
      find $u \in U$ and $a \in \Sigma$ with
        $row(ua) \in Primes(\mathcal{T}) \setminus Primes_{\text{upp}}(\mathcal{T})$
      extend $\mathcal{T}$ to $(T', U \cup \{ua\}, V)$ by memb. queries
    IF $\mathcal{T}$ is not RFSA-consistent **THEN**
      find $u, u' \in U, a \in \Sigma$, and $v \in V$ with
        $T(uav) = -$, $T(u'av) = +$, and $row(u') \sqsubseteq row(u)$
      extend $\mathcal{T}$ to $(T', U, V \cup \{av\})$ by memb. queries
  from $\mathcal{T}$ construct hypothesized NFA $\mathcal{R}_\mathcal{T}$ *// cf. Def. 10*
  IF $(L = L(\mathcal{R}_\mathcal{T}))$ **THEN** equivalence test succeeds
  ELSE
    get counterexample $w \in (L \setminus L(\mathcal{R}_\mathcal{T})) \cup (L(\mathcal{R}_\mathcal{T}) \setminus L)$
    extend $\mathcal{T}$ to $(T', U, V \cup Suff(w))$ by memb. queries
UNTIL equivalence test succeeds
RETURN $\mathcal{R}_\mathcal{T}$

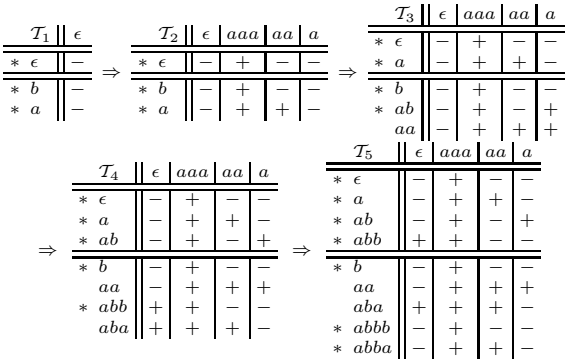
NL* *returns, after at most* $O(n^2)$ *equivalence queries and* $O(m|\Sigma|n^3)$ *membership queries, the canonical RFSA* $\mathcal{R}(L)$.

Proof: First of all, if the algorithm terminates, then it outputs the canonical RFSA for L due to Theorem 1, because passing the equivalence test implies that the constructed automaton must be consistent with the table. To show that the algorithm terminates after at most $O(n^2)$ equivalence queries, we create a measure M that associates a tuple of positive natural numbers to a table $\mathcal{T}$. We let $M(\mathcal{T}) = (l_{up}, l, p, i)$, where $l_{up} = |Rows_{\text{upp}}(\mathcal{T})|$, $l = |Rows(\mathcal{T})|$, $p = |Primes(\mathcal{T})|$, and $i = |\{(r, r') \mid r, r' \in Rows(\mathcal{T}) \text{ and } r \sqsubset r'\}|$. An analysis of the evolution of M during an execution of NL* reveals that, after each extension of the table, either (1) l_{up} is increased or (2) l is increased by $k > 0$ and, simultaneously, i is increased by at most $kl + k(k-1)/2$ or (3) l stays the same and i decreases or p increases. However, l_{up}, l, and p cannot increase beyond n. Hence, the algorithm must (a) always reach an equivalence query and (b) terminate after at most $O(n^2)$ equivalence queries. Concerning the number of membership queries, we notice that their maximal number corresponds to the size of the table which has at most $n + n|\Sigma|$ (n rows in the upper part + their successors) rows and $O(mn^2)$ columns since at each extension at most m suffixes are added to V. □

The theoretical complexity of NL* wrt. equivalence (resp. membership) queries is higher compared to L* where at most n equivalence (resp. roughly $|\Sigma|mn^2$ membership) queries are needed. But we observe that, in practice, *fewer* equivalence and membership queries are needed (cf. Section 4).

NL* by means of an Example Suppose $\Sigma = \{a, b\}$ and let $L_n \subseteq \Sigma^*$ be given by the regular expression $\Sigma^* a \Sigma^n$. I.e., L_n contains the words having an a at the $(n+1)$-last position. Then, L_n is accepted by a minimal DFA $\mathcal{A}_n^*$ with 2^{n+1} states. However, it is easy to see that the canonical RFSA $\mathcal{R}(L_n)$ has $n + 2$ states (see Fig. 1 for $n = 2$). In other words, $\mathcal{R}(L_n)$ is exponentially more succinct than $\mathcal{A}_n^*$.

Next, we show how $\mathcal{R}(L_2)$ is inferred by NL*:



Rows with a preceding $*$ are prime. The table $\mathcal{T}_1$ is RFSA-closed and RFSA-consistent but does not represent the target language because aaa is not accepted. We add aaa and its suffixes to V, perform membership queries, and obtain table $\mathcal{T}_2$, which is not RFSA-closed. We add a to U and continue. Resolving two more closedness violations, we obtain table $\mathcal{T}_5$, which is RFSA-closed and RFSA-consistent. Its automaton $\mathcal{R}_{\mathcal{T}_5}$ given in Fig. 1, is the canonical RFSA for L_2. Notice that table $\mathcal{T}_5$ is not closed in Angluin's sense so that L* would continue adding strings to the upper part of the table.

4 Experiments

To evaluate the performance of our learning algorithm NL*, we compare it with Angluin's L* and its modification wrt. to Remark 1, called L$^*_{\text{col}}$. As NL* is similar in spirit to L$^*_{\text{col}}$, a comparison with this algorithm seems fairer. All algorithms have been implemented in Java and tested on a wide range of examples. Following [Denis *et al.*, 2004], we randomly generate large sets of regular expressions over different sizes of alphabets. A detailed description of this as well as a full description of the outcome can be found in [Bollig *et al.*, 2008]. As in [Denis *et al.*, 2004], we present a characteristic selection of the results for an alphabet of size two.

Results We generated a set of 3180 regular expressions, resulting in minimal DFA of sizes between 1 and 200 states. These DFA were given to the learning algorithms, i.e., membership and equivalence queries were answered according to these automata. To evaluate the algorithms' performance, we measured, for each algorithm and input regular expression, the *number of states of the final automaton* (RSFA or DFA) and the *number of membership (resp. equivalence) queries* that are needed to infer it. As Fig. 2 (top) shows, the automata learned by NL* are considerably smaller than those that are returned by L* and L$^*_{\text{col}}$, confirming the results of [Denis *et al.*, 2004]. More importantly, in practice, the actual sizes of RFSA compared to DFA seem to follow an exponential gap.

In Fig. 2 (middle), the number of membership queries is depicted. As in the first case, NL* behaves much better than the other learning algorithms. While the difference between

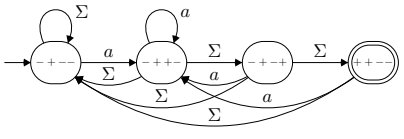


Figure 1: Canonical RFSA of L_2

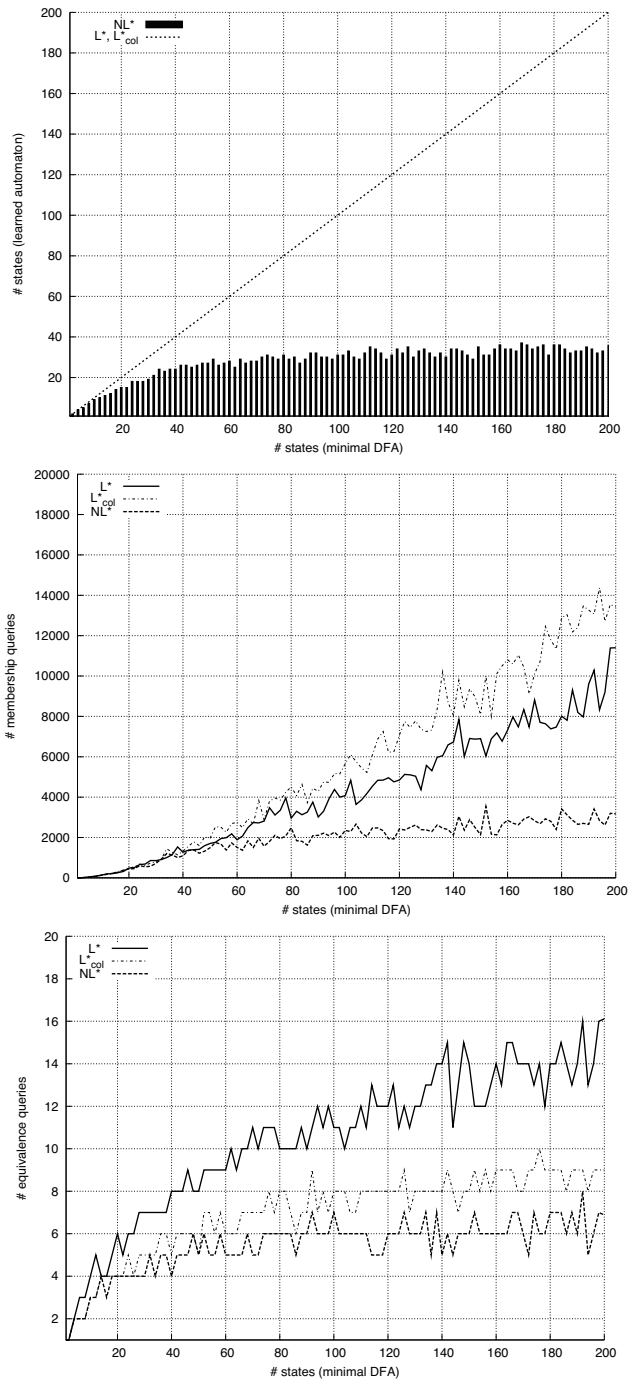


Figure 2: Experimental results

the curves is rather small for automata with less than 40 states, it increases significantly for larger automata. The same is the case for the number of equivalence queries depicted in Fig. 2 (bottom). This is in contrast with the theoretical result we obtained in Theorem 2. The experiments we performed point out the clear predominance of NL^* over L^* and L^*_{col} as long as the user is not dependent on a deterministic model.

5 Future Work

There is room for further improvement by adapting the various recent variants of L^* (see [Leucker, 2007] for references). In the future, we plan to show that using our NL^* algorithm, the limits of learning-based verification techniques can be pushed ahead considerably, as most often non-deterministic automata should be sufficient for verification tasks.

References

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

[Biermann and Feldman, 1972] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behaviour. *IEEE Transactions on Computers*, 21:592–597, 1972.

[Bollig *et al.*, 2008] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. Research Rep. LSV-08-28, LSV, ENS Cachan, 2008.

[de la Higuera, 2005] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[Denis *et al.*, 2002] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51(4):339–368, 2002.

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

[De Wulf *et al.*, 2006] M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV'06*, 2006.

[García *et al.*, 2008] P. García, M. Vazquez de Parga, G. Alvarez, and J. Ruiz. Learning regular languages using non-deterministic finite automata. In *CIAA*, LNCS 5148, 2008.

[Lang, 1992] K. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *COLT*, 1992.

[Leucker, 2007] M. Leucker. Learning meets verification. In *FMCO'07*, LNCS 4709, 2007.

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

[Nerode, 1958] A. Nerode. Linear Automata Transformation. *American Math. Society*, 9:541–544, 1958.

[Oncina and Garcia, 1992] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*. 1992.

[Trakhtenbrot and Barzdin, 1973] B.A. Trakhtenbrot and J.M. Barzdin. *Finite automata: behaviour and synthesis*. North-Holland, 1973.

[Yokomori, 1994] T. Yokomori. Learning non-deterministic finite automata from queries and counterexamples. *Machine Learning*, 13:169–189, 1994.