LEARNING DFA FROM CORRECTIONS[1]

Leonor Becerra-Bonache, Cristina Bibire, Adrian Horia Dediu
Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tarraco 1, 43005, Tarragona, Spain
e-mail: {leonor.becerra,cristina.bibire,adrianhoria.dediu}@estudiants.urv.es

ABSTRACT

This paper is focused on learning deterministic finite automata (DFA) within the framework of query learning. We present an efficient algorithm for learning DFA, based on Angluin's algorithm to identify DFA from membership and equivalence queries. We improve Angluin's results using a new type of query called correction query. Therefore, our learning algorithm has access to a teacher able of answering two type of queries: correction and equivalence queries. It can learn any DFA from an adequate teacher in polynomial time. We prove that it is possible to learn DFA from corrections and that the number of queries are reduced considerably.

Keywords: computational learning theory, query learning, learning DFA, membership query, equivalence query, correction query

1. Introduction

In the last four decades three important formal models have been developed within Computational Learning Theory: Gold's model of *identification in the limit* [5], the *query learning model* of Angluin [1, 2], and the *PAC learning model* of Valiant [12]. All these models have been vividly investigated in the field of Grammatical Inference.

The problem of identifying DFA from examples have been studied quite extensively (see, e.g., [3, 9]). There are several interesting results based on different techniques: identification of DFA from queries in polynomial-time, identification of subclasses of DFA from positive data, identification from erroneous examples, etc. One of the main positive results in the computational learning framework is that DFA can be learned from membership and equivalence queries.

Learning from queries was introduced by Angluin [1]. In query learning, there is a teacher (oracle) that knows the language and has to answer correctly specific kind of queries asked by the learner. The learnability of DFA has been successfully studied in the context of query learning. In [1], Angluin gave an algorithm for learning DFA from membership and equivalence queries. She was the first who proved learnability of DFA via queries. Later, Rivest and Schapire in 1993 ([10]), Hellerstein et al. in 1995 ([6]) or Balcázar et al. in 1996 ([4]) developed more efficient versions of the same algorithm trying to increase the parallelism level, to reduce the number of equivalence queries, etc.

In all the mentioned versions of the Angluin's algorithm, when the learner asks about a word in the language, the teacher's answer is very simple, *yes* or *no*. We consider that this hypothesis is oversimplified for a normal learning process. Our goal is to model a more natural way of

[1]This work was possible thanks to a FPU Fellowship from the Spanish Ministry of Education and Science; Special thanks to professor Victor Mitrana for valuable advices and a careful review; also many thanks to anonymous reviewers for their remarks and suggestions.

answering. When we consider how a child learns a language with the help of an adult, we can see that several aspects of the process of children's language acquisition could be represented in the query learning model. For instance, in that stage in which children overgeneralize, the adults correct them, children apply the corrections to their previous knowledge of the language in order not to repeat again the same errors. Our idea is to reflect this process in the query learning model.

We introduce an extension of membership queries called correction queries. Our algorithm is similar to Angluin's algorithm, the main difference consist of the kind of answers the learner receives. Due to the increased complexity of teacher's answers, we obtained a faster learning process; the increased speed is based on the reduced number of queries (correction and equivalence queries) between the learner and the teacher until the discovering of the language.

The paper is organized as follows. Formal preliminaries and several basic remarks are presented in Section 2. In Section 3 we describe the observation table as the main data structure of the algorithm, we give a proof for the correctness of our algorithm and we present the algorithm along with the time analysis. Section 4 contains a running example and comparative results with Angluin's algorithm whereas in Section 5 we present several concluding remarks.

2. Preliminaries

In this paper we follow standard definitions and notations in formal language theory. We find supplementary information for this domain in [7, 8, 11].

Let Σ be a finite set of symbols called the alphabet and let Σ^* be the set of strings over Σ. A *language* L over Σ is a subset of Σ^*. The elements of L are called *words* or *strings*. Let α, β, γ be strings in Σ^* and $|\alpha|$ be the length of the string α. λ is a special string called the *empty* string and has length 0. Given a string $\alpha = \beta\gamma$, β is the *prefix* of α and γ is the *suffix* of α.

A *deterministic finite automata* (DFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the (finite) set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and δ is a partial function that maps $Q \times \Sigma$ to Q which can be extended to words by doing $\delta(q, \lambda) = q$ and $\delta(q, \alpha a) = \delta(\delta(q, \alpha), a)$, $\forall q \in Q$, $\forall \alpha \in \Sigma^*$, $\forall a \in \Sigma$. A word α is accepted by A if $\delta(q_0, \alpha) \in F$. The set of words accepted by A is denoted by $L(A)$ and is called a *regular language*.

We say that a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is *complete* if for all q in Q and a in Σ, $\delta(q, a)$ is defined (that is δ is a total function). For any DFA A, there exists DFA A' (also called the canonical DFA), such that $L(A) = L(A')$ and the number of states of A' is minim. Without loss of generality, we assume that the target DFA which is to be learned is a canonical DFA.

A state q is called a *live state* if there exist strings α and β such that $\delta(q_0, \alpha) = q$ and $\delta(q, \beta) \in F$. The set of all the live states is called the *liveSet(A)*. A state that is not in the *liveSet* is called a *dead state*. The set of all dead states is called the *deadSet(A)*. Note that for a canonical A DFA the $deadSet(A)$ has at most one element.

For a string $\alpha \in \Sigma^*$, we denote the left quotient of L by α by $Tail_L(\alpha) = \{\beta | \alpha\beta \in L\} = \{\beta | \delta(q_0, \alpha) = q, \delta(q, \beta) \in F\}$, where $A = (Q, \Sigma, q_o, \delta, F)$ is an automaton accepting L.

In a standard query learning algorithm, the learner interacts with a *teacher* called *minimally adequate teacher*, who knows the *target language* (a regular language L over a known alphabet) and is assumed to answer correctly. The goal of the algorithm is to come up with a DFA accepting L. The teacher has to answer two types of queries:

- *Membership queries* (MQ). The learner asks if a string α is in L, and the teacher answers YES or NO.
- *Equivalence queries* (EQ). The learner produces a DFA A and asks whether $L(A)$ is equal with the target language L; the teacher answers YES if the learner automaton is isomorphic

with the target automaton, NO otherwise. If the answer is NO a string α in the symmetric difference of the $L(A)$ and L is returned. This returned string is called *counterexample*.

See [1, 2] for detailed explanations of the model.

We are going to introduce a new type of query called *correction query* (CQ). It is an extension of the MQ; the difference consists in the type of answer that we receive from the teacher. Instead of a *yes/no* answer, a string called the *correctingString* is returned to the learner.

The *correctingString* of α with respect to L is the minimum word (in the lexicographic order, denoted by $\preceq$) of the set $Tail_L(\alpha)$. In the case that $Tail_L(\alpha) = \emptyset$ we set the *correctingString* of α w.r.t. L to φ, where φ is a symbol which does not belong to the alphabet Σ. With these considerations, for the sake of simplicity in notations, we use C instead of *correctingString*. Hence, C is a function from Σ^* to $\Sigma^* \cup \varphi$. Note that $C(\alpha) = \lambda$ if and only if $\alpha \in L$.

Remark 1 If α, β, γ are strings in Σ^* such that $C(\alpha) = \beta \cdot \gamma$ then $C(\alpha \cdot \beta) = \gamma$.

Remark 2 For any $\alpha, \beta \in \Sigma^*$, if $Tail_L(\alpha) = \emptyset$ then $Tail_L(\alpha \cdot \beta) = \emptyset$.

Remark 3 The following results hold:

1. For any $\alpha \in \Sigma^*$ such that $C(\alpha) \neq \varphi$ we have $C(\alpha \cdot C(\alpha)) = \lambda$.
2. Let $\alpha, \beta \in \Sigma^*$. If $C(\alpha) = \varphi$, then $C(\alpha\beta) = \varphi$ but the converse does not hold.

3. Learning with Correction Algorithm (LCA)

We describe the learning algorithm LCA and we show that it efficiently learns an initially unknown regular set from any adequate teacher. Let L be the unknown regular set and let Σ be the alphabet of L.

3.1. Observation Tables

This information is organized into an *observation table* consisting of three parts: a nonempty finite prefix-closed set S of strings, a nonempty finite suffix-closed set E of strings, and the restriction of the mapping C to the set $((S \cup S\Sigma) \cdot E)$. The observation table will be denoted (S, E, C).

An observation table can be visualized as a two-dimensional array with rows labeled by elements of $S \cup S\Sigma$ and columns labeled by elements of E with the entry for row s and column e equal to $C(s \cdot e)$. If s is an element of $(S \cup S\Sigma)$ then $row(s)$ denotes the finite function from E to $\Sigma^* \cup \{\varphi\}$ defined by $row(s)(e) = C(s \cdot e)$. By $rows(S)$ we understand the set $\{row(s) \mid s \in S\}$.

The algorithm LCA uses the observation table to build a deterministic finite automaton. Rows labeled by the elements of S are the candidates for states of the automaton being constructed, and columns labeled by the elements of E correspond to distinguishing experiments for these states. Rows labeled by elements of $S\Sigma$ are used to construct the transition function.

Closed, consistent observation tables. An observation table is called *closed* if for every t in $(S\Sigma - S)$ there exists an s in S such that $row(t) = row(s)$. An observation table is called *consistent* if for any s_1, s_2 in S such that $row(s_1) = row(s_2)$, we have $row(s_1 \cdot a) = row(s_2 \cdot a), \forall a \in \Sigma$.

If (S, E, C) is a closed, consistent observation table, we define a corresponding automaton $A(S, E, C) = (Q, \Sigma, \delta, q_0, F)$, where Q, q_0, F and δ are defined as follows:

$Q = \{row(s) \mid s \in S\}$

$q_0 = row(\lambda)$

$F = \{row(s) \mid s \in S \text{ and } C(s) = \lambda\}$

$\delta(row(s), a) = row(s \cdot a)$

It can be easily shown that *deadSet(A)* $= \{row(s) \mid s \in S \text{ and } C(s) = \varphi\}$ (from Remark 3 we know that $C(s) = \varphi \Rightarrow C(s \cdot a) = \varphi$, $\forall a \in \Sigma$).

To see that this is a well defined automaton, note that since S is a nonempty prefix-closed set, it must contain λ, so q_0 is defined. Also, since E is a nonempty suffix-closed set, it must contain λ. Thus, if s_1 and s_2 are elements of S such that $row(s_1)$=$row(s_2)$, then $C(s_1) = C(s_1 \cdot \lambda) = row(s_1)(\lambda)$ and $C(s_2) = C(s_2 \cdot \lambda) = row(s_2)(\lambda)$ are defined and equal to each other, hence F is well defined. To see that δ is well defined, suppose s_1 and s_2 are elements of S such that $row(s_1) = row(s_2)$. Then since the observation table $A(S, E, C)$ is consistent, for each a in A, $row(s_1 \cdot a) = row(s_2 \cdot a)$, and since it is closed, this common value is equal to $row(s)$ for some s in S.

The important fact about this automaton is the following.

Theorem 1 *If (S, E, C) is a closed and consistent observation table, then the automaton $A(S, E, C)$ is consistent with the finite function C. Any other automaton consistent with C but inequivalent to $A(S, E, C)$ must have more states.*

The theorem is proved by a sequence of straightforward lemmas.

Lemma 2 *Assume that (S, E, C) is a closed and consistent observation table. For the automaton $A(S, E, C)$ and for every s in $S \cup S\Sigma$, $\delta(q_0, s) = row(s)$.*

This lemma can be proved by induction on the length of s. The reader is referred to [1] for details.

Lemma 3 *Assume that (S, E, C) is a closed and consistent observation table. Then the automaton $A = A(S, E, C)$ is consistent with the function C. That is, for every s in $S \cup S\Sigma$ and e in E, the following statements hold:*

1. $C(s \cdot e) = \varphi \Leftrightarrow \delta(q_0, s \cdot e) \in deadSet(A)$,
2. $C(s \cdot e) = t \Leftrightarrow (\delta(q_0, s \cdot e \cdot t) \in F$ *and* $\forall t' \in \Sigma^* \ (\delta(q_0, s \cdot e \cdot t') \in F \Rightarrow t \preceq t'))$.

Proof. We will use in the proof the following remark

Remark 4 For each s in $S \cup S\Sigma$, there exists s' in S such that $\delta(q_0, s \cdot e) = \delta(q_0, s')$ and $C(s \cdot e) = C(s')$ for all $e \in E$.

1. Let s be in $S \cup S\Sigma$ and e in E. Let s' in S be such that $\delta(q_0, s \cdot e) = \delta(q_0, s')$ and $C(s \cdot e) = C(s')$. Then, $C(s \cdot e) = \varphi \Leftrightarrow C(s') = \varphi \Leftrightarrow row(s') \in deadSet(A) \Leftrightarrow \delta(q_0, s') \in deadSet(A) \Leftrightarrow \delta(q_0, s \cdot e) \in deadSet(A)$.

2. Let s be in $S \cup S\Sigma$, e in E and $C(s \cdot e) = t$. We distinguish two cases: $t = \lambda$ and $t \in \Sigma^+$.

Case I) In the case $t = \lambda$, we have to prove that $C(s \cdot e) = \lambda$ if and only if $\delta(q_0, s \cdot e) \in F$ (it is clear that $\forall t' \in \Sigma^* \ \delta(q_0, s \cdot e \cdot t') \in F \Rightarrow \lambda \preceq t'$ since λ is smaller than any other string).

Lets take s' in S such that $\delta(q_0, s \cdot e) = \delta(q_0, s')$ and $C(s \cdot e) = C(s')$. $C(s \cdot e) = \lambda \Leftrightarrow C(s') = \lambda \Leftrightarrow row(s') \in F \Leftrightarrow \delta(q_0, s') \in F \Leftrightarrow \delta(q_0, s \cdot e) \in F$.

Case II) Lets take $t = a_1 \cdot a_2 \cdots a_n$ with $a_i \in \Sigma$, $\forall i \in \{1, 2, ...n\}$ and $n \geq 1$.

From Remark 4, we can find the strings $\{s_0, s_1, ..., s_n\}$ as in the sequel:

s in $S \cup S\Sigma$, e in $E \Rightarrow \exists \ s_0 \in S$ such that $\delta(q_0, s_0) = \delta(q_0, s \cdot e)$ and $C(s_0) = C(s \cdot e)$.

$s_0 \cdot a_1$ in $S \cup S\Sigma$, λ in $E \Rightarrow \exists \ s_1 \in S$ such that $\delta(q_0, s_1) = \delta(q_0, s_0 \cdot a_1)$ and $C(s_1) = C(s_0 \cdot a_1)$.

$s_1 \cdot a_2$ in $S \cup S\Sigma$, λ in $E \Rightarrow \exists \ s_2 \in S$ such that $\delta(q_0, s_2) = \delta(q_0, s_1 \cdot a_2)$ and $C(s_2) = C(s_1 \cdot a_2)$.

. . .

$s_{n-1} \cdot a_n$ in $S \cup S\Sigma$, λ in $E \Rightarrow \exists \ s_n \in S$ such that $\delta(q_0, s_n) = \delta(q_0, s_{n-1} \cdot a_n)$ and $C(s_n) = C(s_{n-1} \cdot a_n)$.

$\delta(q_0, s\cdot e\cdot t) = \delta(q_0, s\cdot e\cdot a_1\cdot a_2\cdots a_n) = \delta(\delta(q_0, s\cdot e), a_1\cdot a_2\cdots a_n) = \delta(\delta(q_0, s_0), a_1\cdot a_2\cdots a_n) = \delta(q_0, s_0\cdot a_1\cdot a_2\cdots a_n) = \delta(\delta(q_0, s_0\cdot a_1), a_2\cdots a_n) = \delta(\delta(q_0, s_1), a_2\cdots a_n)) = \ldots = \delta(\delta(q_0, s_{n-1}), a_n) = \delta(q_0, s_{n-1}\cdot a_n) = \delta(q_0, s_n)$.

We start by showing that $C(s\cdot e) = t$ implies $\delta(q_0, s\cdot e\cdot t) \in F$.

$C(s\cdot e) = t \Rightarrow C(s_0) = a_1\cdot a_2\cdots a_n \Rightarrow C(s_0\cdot a_1) = a_2\cdots a_n \Rightarrow C(s_1) = a_2\cdots a_n \Rightarrow C(s_1\cdot a_2) = a_3\cdots a_n \Rightarrow C(s_2) = a_3\cdots a_n \Rightarrow \ldots \Rightarrow C(s_{n-1}\cdot a_n) = \lambda \Rightarrow C(s_n) = \lambda \Rightarrow row(s_n) \in F \Rightarrow \delta(q_0, s_n) \in F \Rightarrow \delta(q_0, s\cdot e\cdot t) \in F$.

Next we will show that if we take $t = a_1\cdot a_2\cdots a_n$ to be the smallest string in lexicographic order such that $\delta(q_0, s\cdot e\cdot t) \in F$ then $C(s\cdot e) = t$. We know that the set $\{t \mid \delta(q_0, s\cdot e\cdot t) \in F\}$ is not empty because we proved it contains the string $C(s\cdot e)$. Because $\delta(q_0, s\cdot e\cdot t) = \delta(q_0, s_n)$ and $\delta(q_0, s\cdot e\cdot t) \in F$ it follows $\delta(q_0, s_n) \in F \Leftrightarrow row(s_n) \in F \Leftrightarrow C(s_n) = \lambda$.

But $C(s_n) = C(s_{n-1}\cdot a_n) \Rightarrow C(s_{n-1}\cdot a_n) = \lambda \Rightarrow a_n \in Tail_L(s_{n-1})$. We will show that this implies $C(s_{n-1}) = a_n$. Suppose that there exists a string $x_n \prec a_n$ such that $C(s_{n-1}) = x_n$. Then we found a string t' strictly smaller than t such that $\delta(q_0, s\cdot e\cdot t') \in F$, which contradicts the choice we have made for t. We can take t' to be $a_1\cdot a_2\cdots a_{n-1}\cdot x$. Therefore $\delta(q_0, s\cdot e\cdot t') = \delta(q_0, s\cdot e\cdot a_1\cdots a_{n-1}\cdot x) = \delta(\delta(q_0, s\cdot e\cdot a_1\cdots a_{n-1}), x) = \delta(\delta(q_0, s_{n-1}), x) = \delta(q_0, s_{n-1}\cdot x) \in F$ and because $C(s_{n-1}) = x$, it follows that $\delta(q_0, s_{n-1}\cdot x) \in F$.

But $C(s_{n-2}\cdot a_{n-1}) = C(s_{n-1}) = a_n$. For a similar reason as above this implies $C(s_{n-2}) = a_{n-1}\cdot a_n$ (otherwise $t'' = a_1\cdots a_{n-2}\cdot x$, where $x = C(s_{n-2})$, $x \prec a_{n-1}\cdot a_n$ is a string strictly smaller than t such that $\delta(q_0, s\cdot e\cdot t'')$ is in F).

Reasoning in the same manner we obtain that $C(s_0) = a_1\cdot a_2\cdots a_n$ which implies $C(s\cdot e) = t$.

This concludes the proof of the lemma. □

Lemma 4 *Assume that (S, E, C) is a closed, consistent observation table. Suppose the automaton $A(S, E, C)$ has n states. If $A' = (Q', \Sigma, \delta', q'_0, F')$ is any automaton consistent with C that has n or fewer states, then A' is isomorphic with $A(S, E, C)$.*

Proof. We define the relation $\phi \subseteq Q \times Q'$ as follows. For all $s \in S$, $row(s)\ \phi\ q' \Leftrightarrow q' = \delta'(q'_0, s)$.

Lets take $s_1, s_2 \in S$ such that there exists q', $row(s_1)\ \phi\ q'$ and $row(s_2)\ \phi\ q'$. We will show that this implies $row(s_1) = row(s_2)$. $row(s_1)\ \phi\ q' \Leftrightarrow q' = \delta'(q'_0, s_1)$, $row(s_2)\ \phi\ q' \Leftrightarrow q' = \delta'(q'_0, s_2)$, so $\delta'(q'_0, s_1) = \delta'(q'_0, s_2)$. Suppose by contrary that $row(s_1) \neq row(s_2) \Rightarrow \exists e \in E$ such that $row(s_1)(e) \neq row(s_2)(e) \Leftrightarrow C(s_1\cdot e) \neq C(s_2\cdot e))$.

We distinguish two cases: $C(s_1\cdot e) = \varphi$, $C(s_2\cdot e) \neq \varphi$ and $C(s_1\cdot e), C(s_2\cdot e) \neq \varphi$.

Case I) $C(s_1\cdot e) = \varphi$, $C(s_2\cdot e) = t \neq \varphi$. Because A' is consistent with C we have $\delta'(q'_0, s_1\cdot e) \in deadSet(A')$, $\delta'(q'_0, s_2\cdot e\cdot t) \in F'$.

$\delta'(q'_0, s_1) = \delta'(q'_0, s_2) \Rightarrow \delta'(\delta'(q'_0, s_1), e) = \delta'(\delta'(q'_0, s_2), e) \Rightarrow \delta'(q'_0, s_1\cdot e) = \delta'(q'_0, s_2\cdot e) \Rightarrow \delta'(q'_0, s_2\cdot e) \in deadSet(A') \Rightarrow \delta'(q'_0, s_2\cdot e\cdot t) \in deadSet(A')$, which contradicts $\delta'(q'_0, s_2\cdot e\cdot t) \in F'$.

Case II) $C(s_1\cdot e) = t_1$, $C(s_2\cdot e) = t_2$, $t_1 \neq t_2$ and $t_1, t_2 \neq \varphi$.

Because A' is consistent with C we have $\delta'(q'_0, s_1\cdot e\cdot t_1) \in F'$, $\delta'(q'_0, s_2\cdot e\cdot t_2) \in F'$ and t_1, t_2 are the smallest strings with this property.

$\delta'(q'_0, s_1) = \delta'(q'_0, s_2) \Rightarrow \delta'(\delta'(q'_0, s_1), e\cdot t_1) = \delta'(\delta'(q'_0, s_2), e\cdot t_1) \Rightarrow \delta'(q'_0, s_1\cdot e\cdot t_1) = \delta'(q'_0, s_2\cdot e\cdot t_1) \Rightarrow \delta'(q'_0, s_2\cdot e\cdot t_1) \in F \Rightarrow t_2 \preceq t_1$. In a similar way it can be shown that $t_1 \preceq t_2$, from which we draw the conclusion that $t_1 = t_2$, which leads to a contradiction.

We have shown that the relation ϕ is an injection. This implies that $|Q| \leq |\phi(Q)|$. But from our hypothesis we know that $|Q'| \leq |Q|$. So $|Q| \leq |\phi(Q)| \leq |Q'| \leq |Q|$ implies $|Q| = |\phi(Q)| = |Q'|$, which makes our relation ϕ a function.

Because the function ϕ is injective and has the domain and range finite and of the same cardinality, it follows immediately that ϕ is surjective and hence bijective.

We will show that ϕ is an automata isomorphism, that is:

1. $\phi(q_0) = q'_0$. $\phi(q_0) = \phi(row(\lambda)) = \delta'(q'_0, \lambda) = q'_0$.

2. $\phi(F) = F'$. $q' \in \phi(F) \Leftrightarrow \exists s \in S$ such that $row(s) \in F$ and $\phi(row(s)) = q' \Leftrightarrow \exists s \in S$ such that $C(s) = \lambda$ and $\delta'(q'_0, s) = q' \Leftrightarrow \exists s \in S$, $\delta'(q'_0, s) \in F'$, $\delta'(q'_0, s) = q' \Leftrightarrow q' \in F'$.
   In the proof we used the fact that $C(s) = \lambda \Leftrightarrow \delta'(q'_0, s)$ which can be easily deduced from the preceding lemma.
3. $\phi(\delta(row(s), a)) = \delta'(\phi(row(s)), a), \forall s \in S$, $\forall a \in \Sigma$. $\phi(\delta(row(s), a)) = \phi(row(s \cdot a)) = \phi(row(s')) = \delta'(q'_0, s')$, where $s' \in S$ such that $row(s') = row(s \cdot a)$.
   $\delta'(\phi(row(s)), a) = \delta'(\delta'(q'_0, s), a) = \delta'(q'_0, s \cdot a)$.
   Since $\delta'(q'_0, s')$ and $\delta'(q'_0, s \cdot a)$ have identical row values, namely $row(s')$ and $row(s \cdot a)$, they must be the same state of A', so that we conclude that $\phi(\delta(row(s), a)) = \delta'(\phi(row(s)), a), \forall s \in S$, $\forall a \in \Sigma$.

This concludes the proof of Lemma 4. □

Now, the proof of Theorem 1 follows, since Lemma 3 shows that $A(S, E, C)$ is consistent with C, and Lemma 4 shows that any other automaton consistent with C is either isomorphic to $A(S, E, C)$ or contains at least one more state. Thus, $A(S, E, C)$ is the unique smallest automaton consistent with C.

3.2. The Learner LCA

Briefly we recall that the learning algorithm consists actually in a "learner" algorithm communicating with a "teacher" algorithm. The teacher knows a *target language* and the learner wants to discover it. In this description, the teacher is represented as a passive entity, while the learner calls the teacher's methods and properties in order to discover the target language.

The learner algorithm uses as its main data structure the observation table that we described in the previous subsection. Initially $S = E = \lambda$. To determine C, LCA asks CQs for λ and each a in Σ. This initial observation table may or may not be closed and consistent.

The main loop of LCA tests the current observation table (S, E, C) in order to see if it is closed and consistent. If (S, E, C) is not closed, then LCA adds a new string to S and updates the table asking CQs for missing elements. If (S, E, C) is not consistent, then LCA adds a new string to E and updates the table using CQs for missing elements.

When the learner's automaton is closed and consistent the learner asks an EQ. The teacher's answers can be "yes" (in which case the algorithm terminates with the output $A(S, E, C)$) or "no"(in which case a counterexample is provided, all its prefixes are added to S and the table is updated using CQs).

For the proof of the *Termination of LCA* the reader is referred to [1].

Correctness of LCA. If the teacher answers always correctly then LCA terminates. Recall that the teacher's last answer to an EQ is *yes*, the learner's automaton is isomorphic with the target automaton.

Time analysis of LCA. The total running time of LCA can be bounded by a polynomial in n and m (n is the number of states in the minimum automaton accepting L; m is the maximum length of any counterexample string presented by the teacher). For the details of the proof, the reader is referred to [1].

As a learner's optimization, we can mention the creation of a special table $knownValues$. In this table the learner can add the current teacher's answer t indexed by the student's query s and in the same time the answers for all the strings resulted from the concatenation of s with each suffix of t. Another optimization may be performed for the dead state; when for a given query the teacher answers with φ then for all the suffixes of that query the answers are already known and equal to φ.

If we compare LCA with L^* algorithm we can observe that in Angluin's algorithm the number of elements in E is upper bounded by n. Due to the fact that in Angluin's observation table

the answers are coded in binary, to get n different states we need at least $log_2 n$ elements in E. In the LCA algorithm this limitation does not exist so potentially $|E|$ might be even 1 and this lower bound has implications in the general number of asked questions. Let us denote by ℓ the average length for teacher's answers to Correction Queries. The LCA algorithm behaves like having an average lookahead of length ℓ. Supplementary we may consider that the number of already known answers from the previous questions is increased with a factor of ℓ and as a general result we get $CQ \leq MQ/\ell$. In Table 1 we summarize the comparative properties for the L^* and LCA algorithms (recall that n is the number of states, m is the maximal length of an EQ answer, $k = |\Sigma|$, and ℓ is the average length of a CQ).

Table 1: Comparative properties for the L^* and LCA algorithms

Property	L^*	LCA
$\|E\|$	$log_2 n \leq \|E\| \leq n$	$1 \leq \|E\| \leq n$
Size of EQ answers	m	m
MQ/CQ type	$\{yes/no\}$	$\{\lambda/corrStr\}$, $\|corrStr\| \simeq \ell$
General number of MQ/CQ	$\leq (k+1)(n+m(n-1))n$	$\leq MQ/\ell$

The algorithm LCA is described in Figure 1.

```
Procedure Learning with Correction Algorithm
 1)  Initialize S and E with λ;
 2)  Ask correction queries for λ and each a ∈ Σ;
 3)  Construct the initial observation table (S, E, C);
 4)  Repeat
 5)     Repeat
 6)        if Not tableClosed(S, E, C) then
 7)           find s in (SΣ − S) such that row(s) ∉ rows(S);
 8)           add s to S; //actually remove first s from SΣ
 9)           extend SΣ accordingly and C to (S ∪ SΣ)E using correction queries;
10)        if Not tableConsistent(S, E, C) then
11)           find s1,s2 ∈ S, a ∈ Σ and e ∈ E such that
12)              row(s1) = row(s2), and C(s1 · a · e) ≠ C(s2 · a · e);
13)           add a · e to E;
14)           extend C to (S ∪ SΣ)E using correction queries;
15)     until tableClosed and tableConsistent;
16)     construct Learner's conjecture -> learnerAuto, learnerFinalStates;
17)     foundAutomaton:= teacher.askEquiv(learnerAuto, learnerFinalStates);
18)     If Not foundAutomaton then
19)        strCounterEx:=teacher.counterExample;
20)        for each s ∈ Pref(strCounterEx)
21)          if  s ∉ S then
22)           add s to S;
23)           extend SΣ accordingly and C to (S ∪ SΣ)E using correction queries;
24)  until foundAutomaton;
```

Figure 1: Procedure Learning from Corrections

Remark 5 The teacher's *compute corrections* algorithm is still polynomial, performed only once at the beginning, for each state computes the tail starting from the final states.

4. Running Example and Comparative Results

In all our examples we consider that the initial state is always counted as being the state q_0. In order to simplify the automata description we use only the state number as labels for states.

We also introduce the *linear transition table* that is a normal transition table with all the lines written on the same row. From a linear transition table we can restore a normal transition table if we know the cardinality of the alphabet. The final states are deduced from the observation table as being the states having λ on the first column of E (the one corresponding to the experiment λ).

We explain how our algorithm runs by tracing the evolution of the observation table for a language over the alphabet $\Sigma = \{0, 1\}$, $L_1 = (0 + 110)^+$. We can see a minimal automaton associated with the mentioned language in Figure 2. We observe that the linear transition table for this automaton is $(1, 2, 1, 2, 3, 4, 3, 3, 1, 3)$ and the set of final states is $F = \{1\}$. In the theoretical part of the article we use φ as the response of the teacher for the empty set to avoid confusions. For the running example we find more suggestive to use directly $\emptyset$ instead.

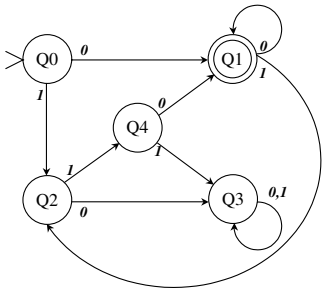

Figure 2: Minimal automaton associated to the language $L_1 = (0 + 110)^+$

Initially the learner starts with the following observation table described as Table 2.

We can observe that the information for the string "0" and the experiment "λ" is known from the corresponding query for $row(\lambda)$: because the correction for the string "λ" is the string "0", after an input string "0" we are in a final state, so we need the string "λ" to reach a final state. We can also see that the table is not closed because $row(0)$ does not belong to $rows(S)$. We add the string "0" to S, "$0 \cdot 0$" and "$0 \cdot 1$" to $S\Sigma - S$, and the corresponding rows to the observation table. The algorithm proceeds in a similar manner with $row(1)$ and after two "not closed" steps we get the Table 3.

Now we see that the current observation table is not closed since $row(10)$ does not belong to $rows(S)$. Again the algorithm adds the string "10" to S, "$10 \cdot 0$" and "$10 \cdot 1$" to $S\Sigma - S$, and the corresponding rows to the observation table. We can see all these operations in Table 4.

As an optimization, the learner infers the teacher answers for $row(100)$ and $row(101)$ as being transitions from a dead state.

In this moment, we can see that the observation table is closed and consistent and it follows an EQ. We added the state information for each row and the automaton that the learner discovered until this moment has: the linear transition table $(1, 2, 1, 2, 3, 0, 3, 3)$, the final states set $F = \{1\}$ and the representation given in Figure 3.

Table 2: $S = \{\lambda\}$, $E = \{\lambda\}$

T_1		λ
1	λ	$\{0\}$
2	0	$\{\lambda\}^{(\lambda,\lambda)}$
3	1	$\{10\}$

Table 3: $S = \{\lambda, 0, 1\}, E = \{\lambda\}$

T_2		λ
1	λ	$\{0\}$
2	0	$\{\lambda\}^{(\lambda,\lambda)}$
3	1	$\{10\}$
4	00	$\{\lambda\}$
5	01	$\{10\}$
6	10	$\emptyset$
7	11	$\{0\}^{(1,\lambda)}$

Table 4: $S = \{\lambda, 0, 1, 10\}, E = \{\lambda\}$

T_3		λ	State
1	λ	$\{0\}$	q_0
2	0	$\{\lambda\}^{(\lambda,\lambda)}$	$q_1 \in F$
3	1	$\{10\}$	q_2
4	10	$\emptyset$	q_3
5	00	$\{\lambda\}$	$q_1 \in F$
6	01	$\{10\}$	q_2
7	11	$\{0\}^{(1,\lambda)}$	q_0
8	100	$\emptyset^{(10,\lambda)}$	q_3
9	101	$\emptyset^{(10,\lambda)}$	q_3

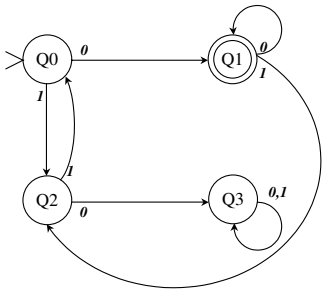

Figure 3: Observation Table 4 and the associated automaton

The teacher's answer to the EQ is negative and the learner gets as a counterexample the string 11110. Adding the counterexample and all its prefixes to S we get the Table 5.

This table is not consistent, since $row(\lambda)$ equals $row(11)$ but $C(\lambda \cdot 1 \cdot \lambda) \neq C(11 \cdot 1 \cdot \lambda)$. In this moment we have to add $1 \cdot \lambda$ to E (Table 6).

Table 5:
$S = \{\lambda, 0, 1, 10, 11, 111, 1111, 11110\}$,
$E = \{\lambda\}$

T_4		λ
1	λ	$\{0\}$
2	0	$\{\lambda\}^{(\lambda,\lambda)}$
3	1	$\{10\}$
4	10	$\emptyset$
5	11	$\{0\}^{(1,\lambda)}$
6	111	$\emptyset$
7	1111	$\emptyset^{(111,\lambda)}$
8	11110	$\emptyset^{(111,\lambda)}$
9	00	$\{\lambda\}$
10	01	$\{10\}$
11	100	$\emptyset^{(10,\lambda)}$
12	101	$\emptyset^{(10,\lambda)}$
13	110	$\{\lambda\}^{(1,\lambda)}$
14	1110	$\emptyset^{(111,\lambda)}$
15	11111	$\emptyset^{(111,\lambda)}$
16	111100	$\emptyset^{(111,\lambda)}$
17	111101	$\emptyset^{(111,\lambda)}$

Table 6:
$S = \{\lambda, 0, 1, 10, 11, 111, 1111, 11110\}$,
$E = \{\lambda, 1\}$

T_5	λ	1	State
λ	$\{0\}$	$\{10\}^{(1,\lambda)}$	q_0
0	$\{\lambda\}^{(\lambda,\lambda)}$	$\{10\}^{(01,\lambda)}$	$q_1 \in F$
1	$\{10\}$	$\{0\}^{(1,\lambda)}$	q_2
10	$\emptyset$	$\emptyset^{(10,\lambda)}$	q_3
11	$\{0\}^{(1,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_4
111	$\emptyset$	$\emptyset^{(111,\lambda)}$	q_3
1111	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3
11110	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3
00	$\{\lambda\}$	$\{10\}$	$q_1 \in F$
01	$\{10\}$	$\{0\}^{(01,\lambda)}$	q_2
100	$\emptyset^{(10,\lambda)}$	$\emptyset^{(10,\lambda)}$	q_3
101	$\emptyset^{(10,\lambda)}$	$\emptyset^{(10,\lambda)}$	q_3
110	$\{\lambda\}^{(1,\lambda)}$	$\{10\}$	$q_1 \in F$
1110	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3
11111	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3
111100	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3
111101	$\emptyset^{(111,\lambda)}$	$\emptyset^{(111,\lambda)}$	q_3

The automaton of the learner now corresponds to the teacher's automaton, so the answer to the EQ is positive. We notice that during the whole algorithm's execution, the learner asked only 2 EQs (the last one was successful) and 8 CQs.

We compare the results obtained by our algorithm with the classical algorithm proposed by Angluin [1]. We used several test languages and we summarize the obtained results in Table 7.

Table 7: Comparative results for different languages using Angluin's algorithm and LCA algorithm

Language description				Angluin		CQ	
Id	Alphabet	Linear transition table	Final states	EQs	MQs	EQs	CQs
L1	$\{0, 1\}$	$(1, 2, 1, 2, 3, 4, 3, 3, 1, 3)$	$\{1\}$	3	44	2	8
L2	$\{0, 1\}$	$(1, 2, 0, 3, 3, 0, 2, 1)$	$\{0\}$	2	19	1	6
L3	$\{a, b\}$	$(1, 2, 3, 4, 4, 4, 1, 4, 4, 4)$	$\{2, 3\}$	2	23	2	10
L4	$\{0, 1, a, b\}$	$(1, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2,$ $0, 0, 4, 2, 2, 2, 2, 5, 0, 0, 3, 3)$	$\{3, 5\}$	4	108	2	48
L5	$\{0, 1\}$	$(1, 2, 3, 4, 5, 6, 7, 8,$ $9, 10, 11, 12, 13, 14, 15, 0)$	$\{1, 2, 4, 8\}$	3	24	3	8
L6	$\{0, 1\}$	$(1, 2, 3, 2, 2, 2, 4, 2, 5, 2, 1, 2)$	$\{5\}$	3	65	1	7

5. Concluding Remarks

We propose a new paradigm for the computational learning theory, namely learning from corrections. Our algorithm based on Angluin's L^* learning algorithm for regular languages uses an observation table with *correcting strings* instead of 0s and 1s. Due to this approach, we use a smaller number of queries and in this way the learning time is reduced. In our running examples, the number of EQs are always less or equal than in Angluin's ones and the number of CQs is significantly smaller than the number of MQs.

One of the reasons of this reduction is that an answer to a CQ contains embedded much more information. Another advantage of our approach is that we can differentiate better between states.

Among the improvements previously discussed, we would like to mention here the adequacy of CQs in a real learning process. They reflect in a more accurate manner the process of children's language acquisition. We are aware that this kind of formalism is for an ideal teacher who knows everything and always gives the correct answers and for practical applications our working hypothesis should be adjusted.

For the future we will try to improve our algorithm using additional information such as multiple correct answers and to reduce the number of EQ to 0. In fact these EQ are quite un-natural for real learning environments. We will also try to extend this result to Context Free and Mildly Context Sensitive Languages.

References

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

[2] D. Angluin, Queries and concept learning. *Machine Learning* **2** (1988), 319–342.

[3] D. Angluin, C. H. Smith, Inductive inference: Theory and methods. *Comput. Surveys* **15** (1983), 237–269.

[4] J. L. Balcázar, J. Díaz, R. Gavaldá, O. Watanabe, Algorithms for learning finite automata from queries: A unified view. *Chapter in Advances in Algorithms, Languages, and Complexity*. D.-Z. Du and K.-I. Ko (eds.), Kluwer Academic Publishers, 1997, 73-91.

[5] E. M. Gold, Identification in the limit. *Information and Control* **10** (1967), 447–474.

[6] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, D. Wilkins, How many queries are needed to learn?. In: *Proc. 27th Annual ACM Symposium on the Theory of Computing*. ACM Press, 1995, 190–199.

[7] J. Hopcroft, R. Motwani, J. Ullman, *Introduction to automata theory, languages, and computation*. Addison Wesley, 2001.

[8] C. Martín-Vide, V. Mitrana, Gh. Paun (eds.), *Formal languages and applications*. Springer-Verlag, Berlin, 2004.

[9] D. Pitt, Inductive inference, DFAs, and computational complexity. In: *Proc. AII-89 Workshop on Analogical and Inductive Inference*. Lecture Notes in Computer Science **397**, Springer-Verlag, Berlin, 1989, 18–44.

[10] R. L. Rivest, R. E. Schapire, Inference of finite automata using homing sequences *Information and Computation*. **103(2)**:299347, Apr. 1993.

[11] G. Rozenberg, A. Salomaa (eds.), *Handbook of formal languages*. Vol **1-3**, Springer-Verlag, Berlin, 1997.

[12] L. G. Valiant, A theory of the learnable. *Communication of the ACM* **27** (1984), 1134–1142.

Annex, Proofs of Remarks

Remark 1 If α, β, γ are strings in Σ^* such that $C(\alpha) = \beta \cdot \gamma$ then $C(\alpha \cdot \beta) = \gamma$.

Proof. Lets take $\alpha, \beta, \gamma \in \Sigma^*$ such that $C(\alpha) = \beta \cdot \gamma \Rightarrow \alpha \cdot \beta \cdot \gamma \in L \Rightarrow \gamma \in Tail_L(\alpha \cdot \beta)$. Assume that $\gamma \neq C(\alpha \cdot \beta) \Rightarrow \exists\ \gamma' \in \Sigma^*$ such that $\gamma' \prec \gamma$ and $C(\alpha \cdot \beta) = \gamma' \Rightarrow \alpha \cdot \beta \cdot \gamma' \in L \Rightarrow \beta \cdot \gamma' \in Tail_L(\alpha)$. But $\beta \cdot \gamma = C(\alpha) \Rightarrow \beta \cdot \gamma \preceq \beta \cdot \gamma' \Rightarrow \gamma \preceq \gamma'$, which is a contradiction. Hence, $C(\alpha \cdot \beta) = \gamma$ □

Remark 2 For any $\alpha, \beta \in \Sigma^*$, if $Tail_L(\alpha) = \emptyset$ then $Tail_L(\alpha \cdot \beta) = \emptyset$.

Proof. Lets take $\alpha, \beta \in \Sigma^*$ such that $Tail_L(\alpha) = \emptyset$. Suppose $Tail_L(\alpha \cdot \beta) \neq \emptyset$ and let γ be an element of $Tail_L(\alpha \cdot \beta)$. $\gamma \in Tail_L(\alpha \cdot \beta) \Rightarrow (\alpha \cdot \beta) \cdot \gamma \in L \Rightarrow \beta \cdot \gamma \in Tail_L(\alpha) \Rightarrow Tail_L(\alpha) \neq \emptyset$, contradiction with our hypothesis. □

Remark 3 The following results hold:

1. For any $\alpha \in \Sigma^*$ such that $C(\alpha) \neq \varphi$ we have $C(\alpha \cdot C(\alpha)) = \lambda$.
2. Let $\alpha, \beta \in \Sigma^*$. If $C(\alpha) = \varphi$, then $C(\alpha\beta) = \varphi$ but the converse does not hold.

Proof. Let α be an arbitrary string in Σ^*.

1. Let $\beta = C(\alpha)$, $\beta \in \Sigma^*$. $C(\alpha) = \beta \Rightarrow \beta \in Tail_L(\alpha) \Rightarrow \alpha \cdot \beta \in L \Rightarrow (\alpha \cdot \beta) \cdot \lambda \in L \Rightarrow \lambda \in Tail_L(\alpha \cdot \beta) \Rightarrow C(\alpha \cdot \beta) = \lambda$.
2. Suppose $C(\alpha) = \varphi$ and take any $\beta \in \Sigma^*$. We have $C(\alpha) = \varphi \Leftrightarrow Tail_L(\alpha) = \emptyset \Rightarrow Tail_L(\alpha \cdot \beta) = \emptyset \Leftrightarrow C(\alpha \cdot \beta) = \varphi$.
   It is easy to see that for $L = (ab)^*$, $C(aba \cdot a) = \varphi$ but $C(aba) \neq \varphi$.

□

Remark 4 For each s in $S \cup S\Sigma$, there exists s' in S such that $\delta(q_0, s \cdot e) = \delta(q_0, s')$ and $C(s \cdot e) = C(s')$ for all $e \in E$.

Proof. The proof is by induction on the length of e.

- $e = \lambda$. (S, E, C) is a closed table, $s \in S \cup S\Sigma \Rightarrow \exists s' \in S$ such that $row(s') = row(s) \Rightarrow (C(s') = C(s)$ and $\delta(q_0, s') = \delta(q_0, s))$.
- Suppose the result holds for all e in E of length at most k, and let e be an element of E of length $k+1$. Then, since E is suffix-closed, $e = a \cdot e'$ for some a in Σ and e' in E. We take s' in S such that $row(s)$=$row(s') \Rightarrow C(s \cdot e) = C(s' \cdot e)$.
  $\delta(q_0, s \cdot e) = \delta(\delta(q_0, s), a \cdot e')$
  $= \delta(row(s), a \cdot e')$, by the previous lemma,
  $= \delta(row(s'), a \cdot e')$, since $row(s) = row(s')$,
  $= \delta(\delta(row(s'), a), e')$,
  $= \delta(row(s' \cdot a), e')$, by the definition of δ,
  $= \delta(\delta(q_0, s' \cdot a), e')$, by the previous lemma,
  $= \delta(q_0, s' \cdot a \cdot e')$.
  By the induction hypothesis on e', there exist s'' in S such that $\delta(q_0, (s' \cdot a) \cdot e') = \delta(q_0, s'')$ (which implies $\delta(q_0, s \cdot e) = \delta(q_0, s'')$) and $C(s' \cdot a \cdot e') = C(s \cdot e) = C(s'')$.

□