Learning DFA from Correction and Equivalence Queries*

Leonor Becerra-Bonache[1], Adrian Horia Dediu[1,2], and Cristina Tîrnăucă[1]

[1] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
[2] Faculty of Engineering in Foreign Languages
University "Politehnica" of Bucharest
Splaiul Unirii 313, 060042 Bucharest, Romania
{leonor.becerra, adrianhoria.dediu, cristina.bibire}@estudiants.urv.es
http://www.grlmc.com

Abstract. In active learning, membership queries and equivalence queries have established themselves as the standard combination to be used. However, they are quite "unnatural" for real learning environments (membership queries are oversimplified and equivalence queries do not have a correspondence in a real life setting). Based on several linguistic arguments that support the presence of corrections in children's language acquisition, we propose another kind of query called correction query. We provide an algorithm that learns DFA using correction and equivalence queries in polynomial time. Despite the fact that the worst case complexity of our algorithm is not better than Angluin's algorithm, we show through a large number of experiments that the average number of queries is considerably reduced by using correction queries.

Keywords: Active learning, learning DFA, membership query, equivalence query, correction query.

1 Introduction

A general theory regarding human learning mechanisms that underlie natural language acquisition is still missing. Several questions arise from the beginning, among others: how are children able to learn languages so fluently and effortlessly, without explicit instruction? To what kind of data are they exposed to?

There is no doubt that children learn a language in part by hearing sentences of that language. However, there is an aspect of the child's linguistic environment which has been subject of a long debate and which is still an important research topic for both linguists and formal language theoreticians. While it is accepted that positive data are available to the child, the availability of another kind of data has been widely argued.

* This work was possible thanks to the FPU Fellowships AP2001-1880, AP2004-6968 from the Spanish Ministry of Education and Science and to the grant 2002CAJAL-BURV4, provided by the University Rovira i Virgili.

Y. Sakakibara et al. (Eds.): ICGI 2006, LNAI 4201, pp. 281–292, 2006.

Taking into account that children do not receive only passive information and they interact with their environment, we consider that active learning might be useful to model several aspects of children's language acquisition.

In active learning, the learner is allowed to ask queries to the teacher. Membership queries (MQs) and equivalence queries (EQs) have established themselves as the standard combination to be used. They were introduced by Angluin in [1]. She proved that DFA can be inferred in polynomial time from this type of queries (this algorithm is known as L^*). However, they are quite "unnatural" for real learning environments. For instance, when the learner asks about a word in the language, the teacher's answer yes/no is oversimplified.

As there is a growing evidence that corrections are available to children [2,3], we believe that they can play a complementary role in language learning, although the main source of information received during the learning process is positive data. Therefore, we propose another kind of query called correction query (CQ).

What should a good correction be like? It could be defined by some function that given any string associates some string in the language. This function might be deterministic or stochastic. In order to simplify the problem, we are going to consider in this paper a deterministic correction, which consists of the smallest string (in lex-length order) that attached to the end of the requested string will give us a string in the language.

Considering the simplicity of DFA and their adequacy for various applications of natural language processing (although regular languages have limited expressiveness), we consider that a starting point could be to apply corrections to learn DFA. We design an algorithm called *Learning from Corrections Algorithm* (LCA), which is able to infer a minimal complete DFA using CQs and EQs.

Although in the worst case our algorithm works as Angluin's algorithm, we show that in general LCA performs better. Therefore, we can see not only that it is possible to learn DFA from corrections, but also that the number of queries between the learner and the teacher until the discovering of the language is reduced considerably. Moreover, there are some classes of languages for which the number of EQs is reduced to only one (in this case, the conjectured DFA is equivalent to the target one) and therefore, we can consider that EQs are not necessary at all for these classes.

This 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 Section 5 presents some comparative results with Angluin's algorithm, both theoretical and experimental. In Section 6 we present several concluding remarks.

2 Preliminaries

In this paper we follow standard definitions and notations in formal language theory. Supplementary information for this domain can be found in [4,5].

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. This function can be extended to words by writing $\delta(q, \lambda) = q$ and $\delta(q, s \cdot a) = \delta(\delta(q, s), a)$, $\forall q \in Q, \forall s \in \Sigma^*, \forall a \in \Sigma$. A string s is accepted by A if $\delta(q_0, s) \in F$. The set of strings 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 a minimum state DFA A', such that $L(A) = L(A')$. Without loss of generality, we assume that the target DFA which is to be learned is a minimal complete 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 minimal DFA A, *deadSet*(A) has at most one element.

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

In a standard query learning algorithm, the learner interacts with a *teacher* that 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: MQs - the learner asks if a string α is in L, and the teacher answers "yes" or "no"; EQs - the learner makes a conjecture of the DFA; the teacher answers "yes" if the learner automaton is isomorphic with the target automaton and "no" otherwise; if the answer is "no" a string α in the symmetric difference of $L(A)$ and L is returned (the *counter example*). See [1,6] for detailed explanations of the model.

We are going to propose another type of query called 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 lex-length order, denoted by $\preceq$) of the set L_α. In the case that $L_\alpha = \emptyset$ 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 $L_\alpha = \emptyset$ then $L_{\alpha \cdot \beta} = \emptyset$.

Remark 3. For any $\alpha \in \Sigma^*$, the following results hold:

1. If $C(\alpha) \neq \varphi$ then $C(\alpha \cdot C(\alpha)) = \lambda$.
2. If $C(\alpha) = \varphi$ then $\forall \beta \in \Sigma^*$, $C(\alpha\beta) = \varphi$.

3 Learning from Corrections Algorithm (*LCA*)

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

3.1 Observation Tables

The information we have at each step of the algorithm is organized into an *observation table* consisting of: 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 labelled by elements of $S \cup S\Sigma$ and columns labelled 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 DFA. Rows labelled by the elements of S are the candidates for states of the automaton being constructed, and columns labelled by the elements of E correspond to distinguishing experiments for these states. Rows labelled 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 s in $(S\Sigma - S)$ there exists an s' in S such that $row(s) = row(s')$. 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:
$Q = \{row(s) \mid s \in S\}$
$q_0 = row(\lambda)$
$F = \{row(s) \mid s \in S \text{ and } C(s) = \lambda\}$
$\delta(row(s), a) = row(s \cdot a)$

One can see that this automaton is well defined and that *deadSet(A)* $= \{row(s) \mid s \in S \text{ and } C(s) = \varphi\}$ (from Remark 3 we know that $C(s) = \varphi \Rightarrow C(s \cdot a) = \varphi, \forall a \in \Sigma$).

Definition 1. *Assume that (S, E, C) is a closed and consistent observation table. We say that the automaton $A = (Q, \Sigma, \delta, q_0, F)$ is consistent with the function C if 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* $\forall t' \in \Sigma^* \ (\delta(q_0, s \cdot e \cdot t') \in F \Rightarrow t \preceq t'))$.

The important fact about the automaton $A(S, E, C)$ 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 follows from the following sequence of straightforward lemmas.

Lemma 1. *Assume that (S, E, C) is a closed and consistent observation table. For the automaton $A(S, E, C)$ and for every s in $S \cup S\Sigma$, $\delta(q_0, s) = row(s)$.*

Lemma 2. *If (S, E, C) is a closed and consistent observation table and the automaton $A(S, E, C)$ is $(Q, \Sigma, \delta, q_0, F)$, then for each s in $S \cup S\Sigma$ and all $e \in E$, there exists s' in S such that $\delta(q_0, s \cdot e) = \delta(q_0, s')$ and $C(s \cdot e) = C(s')$.*

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.*

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 $A(S, E, C)$.*

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*

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).

Correctness of LCA. If the teacher answers always correctly then if LCA ever terminates its output is clearly the target one. Recall that the teacher's last answer to an EQ before halting is *yes*.

Algorithm 1. Learning from Corrections algorithm

Initialize S and E with λ
Ask correction queries for λ and each $a \in \Sigma$
Construct the initial observation table (S, E, C)
repeat
  while (S, E, C) is not closed or not consistent **do**
    if (S, E, C) is not closed **then**
      find s in S and a in Σ such that $row(s \cdot a) \notin rows(S)$
      add $s \cdot a$ to S
      extend C to $(S \cup S\Sigma)E$ using CQs
    end if
    if (S, E, C) is not consistent **then**
      find s_1,$s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ and $C(s_1 \cdot a \cdot e) \neq C(s_2 \cdot a \cdot e)$
      add $a \cdot e$ to E
      extend C to $(S \cup S\Sigma)E$ using CQs
    end if
  end while
  Construct the conjecture $A(S, E, C)$
  if the teacher replies with a counter example s **then**
    add s and all its prefixes to S;
    extend C to $(S \cup S\Sigma)E$ using CQs;
  end if
until the teacher replies *yes* to the conjecture
Halt and output $A(S, E, C)$

Termination of LCA. To see that LCA terminates, notice that the injectivity of function ϕ defined on Lemma 4 implies that for any closed and consistent observation table (S, E, C), if n denotes the number of different values of $row(s)$ for s in S then any automaton consistent with C must have at least n states. The proof follows the lines of Angluin's paper [1].

Time analysis of LCA. The time complexity of LCA is polynomial in n and m (n is the number of states in the minimum automaton accepting L and 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].

4 Running Example

In order to simplify the automata description we introduced the *linear transition table* that is a normal transition table with all the lines written on the same row.

We explain how our algorithm runs by tracing the evolution of the observation table for a language over the alphabet $\Sigma = \{0, 1\}$, $L = (0 + 110)^+$. We can see a minimal automaton associated with the mentioned language in Figure 1.

We observe that the linear transition table for this automaton is (q_1, q_2, q_1, q_2, q_3, q_4, q_3, q_3, q_1, q_3) and the set of final states is $F = \{q_1\}$.

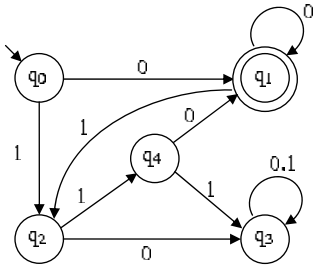

Fig. 1. Minimal automaton associated with the language $L = (0 + 110)^+$

Initially the learner starts with $S = \{\lambda\}, E = \{\lambda\}$ and the observation table described as Table 1.

Table 1.

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

We can observe that the information for the string 0 and the experiment λ is known from the corresponding query for λ, since $C(\lambda) = 0$ implies $C(0) = \lambda$.

The table is not closed because $row(0)$ and $row(1)$ do not belong to $rows(S)$. We add the strings 0 and 1 to S and we extend the table using CQs. The observation table is still not closed since $row(10)$ does not belong to $rows(S)$. The algorithm adds the string 10 to S. We can notice that the corrections for the strings 100 and 101 are already known, since $C(10) = \varphi$ implies $C(100) = C(101) = \varphi$. The current observation table is represented in Table 2.

Table 2.

T_2	λ	State
λ	0	q_0
0	$\lambda^{(\lambda,\lambda)}$	$q_1 \in F$
1	10	q_2
10	φ	q_3
00	λ	$q_1 \in F$
01	10	q_2
11	$0^{(1,\lambda)}$	q_0
100	$\varphi^{(10,\lambda)}$	q_3
101	$\varphi^{(10,\lambda)}$	q_3

In this moment, we can see that the observation table is closed and consistent and it follows an EQ. The conjectured automaton has the linear transition

table $(q_1, q_2, q_1, q_2, q_3, q_0, q_3, q_3)$ and the set of final states $F = \{q_1\}$ and the representation given in Figure 2.

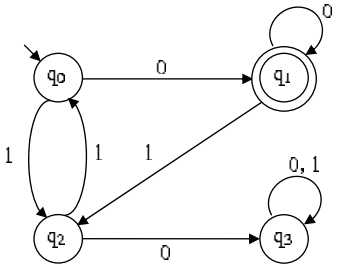

Fig. 2. The automaton associated with Table 2

This is not the target automaton and hence the teacher answers with a counter example. Suppose that the counter example returned is the string 11110. The algorithm adds this string and all its prefixes to S and updates the table. The current observation table is not consistent, since $row(\lambda)$ equals $row(11)$ but $C(\lambda \cdot 1 \cdot \lambda) \neq C(11 \cdot 1 \cdot \lambda)$. The algorithm adds the string 1 to E.

As we can see, the current observation table (Table 3) is closed and consistent and the conjectured automaton is isomorphic with the target one, so the teacher's answer to the EQ is positive. We notice that during the whole algorithm's execution, the learner asked only two EQs (the last one was successful) and eight CQs.

Table 3.

T_3	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	φ	$\varphi^{(10,\lambda)}$	q_3
11	$0^{(1,\lambda)}$	$\varphi^{(111,\lambda)}$	q_4
111	φ	$\varphi^{(111,\lambda)}$	q_3
1111	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3
11110	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3
00	λ	10	$q_1 \in F$
01	10	$0^{(01,\lambda)}$	q_2
100	$\varphi^{(10,\lambda)}$	$\varphi^{(10,\lambda)}$	q_3
101	$\varphi^{(10,\lambda)}$	$\varphi^{(10,\lambda)}$	q_3
110	$\lambda^{(1,\lambda)}$	10	$q_1 \in F$
1110	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3
11111	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3
111100	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3
111101	$\varphi^{(111,\lambda)}$	$\varphi^{(111,\lambda)}$	q_3

5 Comparative Results

In this section we present some theoretical and practical results. We prove that there exist classes of languages for which the number of queries needed to learn is lower for our algorithm. Using running tests we also show that in practice our algorithm uses less queries.

5.1 Theoretical Results

We believe that in most of the cases LCA performs not worst that L^* and that there are several subclasses of regular languages for which our algorithm needs smaller number of queries. In the sequel we present one of such subclasses.

Without loss of generality, the teacher is supposed to return the shortest counterexample. Let L be the target language and m the size of the minimal complete DFA accepting L.

Definition 2. *By $MQ_L(m)$ ($CQ_L(m)$) we denote the number of different words submitted by the learner L^* (LCA) to the teacher in order to identify L.*

Theorem 2. *There exists an infinite class of languages which require a polynomial number of MQs but a linear number of CQs in order to be identified.*

Let us consider $\mathcal{S}_\Sigma$ the class of singletons over Σ, that is the languages which contain only one string. The theorem follows from the following two lemmas.

Lemma 5. *For any fixed alphabet Σ of length k and any language L in $\mathcal{S}_\Sigma$ the number of MQs needed by L^* in order to identify L is:*

$$MQ_L(m) = 2(k-1)m^2 - (4k-7)m + 2k - 5\ . \tag{1}$$

Lemma 6. *For any fixed alphabet Σ of length k and any language L in $\mathcal{S}_\Sigma$ the number of CQs needed by LCA in order to identify L is:*

$$CQ_L(m) = (k-1)(m-1) + 2\ . \tag{2}$$

and the number of EQs is only one.

Corollary 1. *For the one letter alphabet, the class $\mathcal{S}_\Sigma$ needs a linear number of MQs and a constant number of CQs. More precisely, given the language L, L^* asks a total number of $3m - 3$ MQs (where m is the size of its minimal DFA), meanwhile LCA asks only 2 CQs.*

5.2 Practical Results

Due to the coding of the states and to the embedded information within the teacher's answers, our practical results reflect the improvement brought by our algorithm.

We tested L^* and LCA on an randomly generated set of 209 DFA, all of them on a two letter alphabet. To be able to visualize the comparison between the efficiency of the two algorithms, we used the average value of the number of queries for automata with the same number of states. For that purpose, we needed a test set with equally distributed number of states (more precisely, we randomly generated 11 automata having 2, 3,... up to 20 states).

In order to generate the DFA test set, we used the public package from http://www.research.att.com/sw/tools/fsm/ having the documentation available in [7].

To generate random automata, we considered a maximum number of states of 20 for our examples. Without loss of generality, the initial state is always the state 0. Then we generate complete transition tables having the destination states generated randomly between 0 and the maximum number of states. For the number of final states, we generated a random number between 0 and the maximum number of states -1. Then, we generated again randomly the final states (without repeating twice the same state).

After minimizing, we checked the newly found DFA for non-equivalence with the already generated automata. In case of equivalence, we generated new automata. Finally, we checked for completeness and then we loaded the automata in our programs.

We generated two graphics for the results obtained with the two letters alphabet. The first one, represented in Figure 3, contains the average values for the number of EQs asked by L^* and LCA respectively, the average being computed for automata having the same number of states.

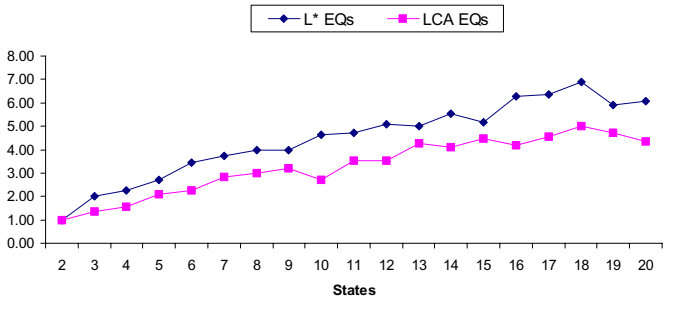

Fig. 3. *EQs* average values for automata with the same number of states;

The second graphic, represented in Figure 4, contains the average values obtained for the number of MQs and CQs, respectively, for automata with the same number of states.

We also run tests on different size alphabets. The results obtained strengthen our belief that the improvements of our algorithm are not limited to some small class of languages.

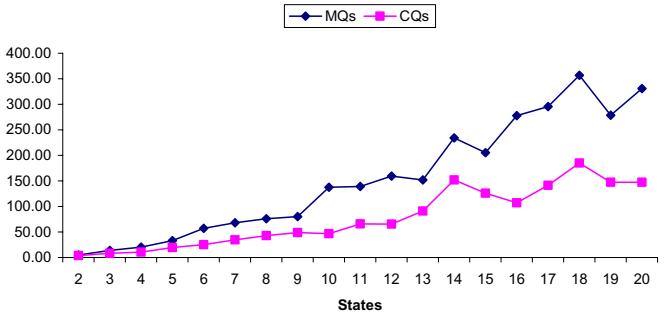

Fig. 4. *MQs* and *CQs*, average values for automata with the same number of states

6 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. In our running examples, generally, the number of EQs are less or equal than in Angluin's ones and the number of CQs is significantly smaller than the number of MQs.

The empirical results show that in most of the cases the number of queries used by LCA is smaller. We believe that this is related to the injectivity property: a language is injective if any two non equivalent strings (using the standard Myhill-Nerode equivalence) have different corrections. One can see that the rare cases in which our algorithm performs worse, compared to L^*, is when this injectivity property is far from being satisfied. On the other hand, when this property is fulfilled, EQs are no longer needed.

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.

Since there is no correspondence of an EQ in a real situation (a child will never ask to the adult if her grammar is the correct one), as a future work we would like to extend our results on singleton languages to a bigger class and hence to enlarge the class of languages learnable from only CQs. Moreover, we will try to find subsets of regular languages for which LCA performs always better than L^* and to identify a necessary and sufficient condition for a class of languages to be faster learnable using our algorithm. We will also like to extend this result to Context Free and Mildly Context Sensitive Languages, which are considered more appropriate to model some aspects of natural language acquisition (probably another type of correction would be needed).

Acknowledgements

Special thanks to professors Victor Mitrana, Colin de la Higuera and Dana Angluin for valuable advices and a careful review. Also many thanks to anonymous reviewers for their remarks and suggestions.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
2. Becerra-Bonache, L.: On the Learnability of Mildly Context-Sensitive Languages using Positive Data and Correction Queries. Doctoral thesis, Rovira i Virgili University (2006)
3. Becerra-Bonache, L., Yokomori, T.: Learning mild context-sensitiveness: Toward understanding children's language learning. In: Proceedings of the 7th International Colloquium on Grammatical Inference. (2004) 53–64
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (2001)
5. Yu, S.: Finite automata. In Martin-Vide, C., Mitrana, V., Paun, G., eds.: Formal Languages and Applications. Studies in Fuzzyness and Soft Computing 148. Springer, Berlin (2004) 55–85 ISBN 3-540-20907-7.
6. Angluin, D.: Queries and concept learning. Machine Learning **2**(4) (1988) 319–342
7. Mohri, M., Pereira, F.C.N., Riley, M.: A rational design for a weighted finite-state transducer library. In: WIA '97: Revised Papers from the Second International Workshop on Implementing Automata, London, UK, Springer-Verlag (1998) 144–158