Polynomial Time Algorithms for Learning *k*-Reversible Languages and Pattern Languages with Correction Queries[⋆]

Cristina Tîrnăucă[1] and Timo Knuutila[2]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tàrraco 1, Tarragona 43005, Spain
[2] Department of Information Technology, University of Turku
Joukahaisenkatu 3-5 B, FI-20014 Turku, Finland
`cristina.bibire@estudiants.urv.es,timo.knuutila@it.utu.fi`

Abstract. We investigate two of the language classes intensively studied by the algorithmic learning theory community in the context of learning with correction queries. More precisely, we show that any pattern language can be inferred in polynomial time in length of the pattern by asking just a linear number of correction queries, and that k-reversible languages are efficiently learnable within this setting. Note that although the class of all pattern languages is learnable with membership queries, this cannot be done in polynomial time. Moreover, the class of k-reversible languages is not learnable at all using membership queries only.

Key words: Correction queries, k-reversible languages, pattern languages, polynomial algorithms.

1 Introduction

Without any doubt, there is no formal model that can capture all aspects of human learning. Nevertheless, the overall aim of researchers working in algorithmic learning theory has been to gain a better understanding of what learning really is. Actually, the field itself has been introduced as an attempt to construct a precise model for the notion of "being able to speak a language" [9].

Among the most celebrated models (Gold's model of *learning from examples* [9], Angluin's *query learning* model [4], Valiant's *PAC learning* model [18]), the best one for describing the child-adult interaction within the process of child acquiring his native language is the one proposed in [4]. There, *the learner* receives information about a target concept by asking queries of a specific kind (depending on the chosen query model type) which will be truthfully answered

[⋆] The preparation of this paper was done while the first author was visiting the Department of Mathematics of Turku University, and was supported in part by the European Science Foundation (ESF) for the activity entitled 'Automata: from Mathematics to Applications', and by the FPU Fellowship AP2004-6968 from the Spanish Ministry of Education and Science.

by *the teacher.* After finitely many queries, the learner is required to return its hypothesis, and this should be the correct one.

The first query learning algorithm, called L^*, was able to identify any minimal complete deterministic finite automaton (DFA) in polynomial time, using membership queries (MQs) and equivalence queries (EQs) [4]. Meanwhile, other types of queries have been introduced: subset, superset, disjointness and exhaustive queries [5], structured MQs [15], *etc.*, and also various target concepts have been investigated: non-deterministic finite automata [19], context-free grammars [15], two-tape automata [20], regular tree languages [8, 17], *etc.*

Still, none of the above mentioned queries reflects one important aspect of children language acquisition, namely that although children are not explicitly provided negative information, they are corrected when they make mistakes. Following this idea, L. Becerra-Bonache, A.H. Dediu and C. Tîrnăucă introduced in [7] a new type of query, the so-called *correction query* (CQ), and showed that DFAs are learnable in polynomial time using CQs and EQs. Continuing the investigation on CQs, C. Tîrnăucă and S. Kobayashi found necessary and sufficient conditions for an indexable class of recursive languages to be learnable with CQs only [16]. Also, they showed some relations existing between this model and other well-known (query and Gold-style) learning models.

In contrast with the approach in [16], where the learnability was studied regardless time complexity, we focus in this paper on algorithms for identifying language classes in polynomial time. Thus, the rest of the paper is structured as follows. Preliminary notions and results are presented in Section 2. In Section 3 we give a polynomial time algorithm for learning the class of k-reversible languages with CQs. Section 4 contains an algorithm for learning the class of pattern languages, along with discussions about correctness, termination and time analysis. In Section 5 we present some results on the learnability with MQs of the classes investigated in the previous sections. We conclude with remarks and future work ideas (Section 6).

2 Preliminaries

Familiarity with standard recursion and language theoretic notions is assumed (good introductory books are [10, 12], for example).

Let Σ be a finite alphabet of symbols. By Σ^* we denote the set of all finite strings of symbols from Σ. A *language* is any set of strings over Σ. The length of a string w is denoted by $|w|$, and the concatenation of two strings u and v by uv or $u \cdot v$. The empty string (i.e., the unique string of length 0) is denoted by λ. If $w = uv$ for some $u, v \in \Sigma^*$, then u is a *prefix* of w and v is a *suffix* of w.

A set S is said to be *prefix-closed* if for all strings u in S and all prefixes v of u, the string v is also in S. The notion of *suffix-closed* set is defined similarly.

By $\Sigma^{\leq k}$ we denote the set $\{w \in \Sigma^* \mid |w| \leq k\}$, by $\mathit{Pref}(L)$ the set $\{u \mid \exists v \in \Sigma^* \text{ such that } uv \in L\}$ of all prefixes of a language $L \subseteq \Sigma^*$, and by $\mathit{Tail}_L(u) = \{v \mid uv \in L\}$ the left-quotient of L and u. Thus, $\mathit{Tail}_L(u) \neq \emptyset$ if and only if $u \in \mathit{Pref}(L)$.

A *deterministic finite automaton* is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a 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, called *transition function*, that maps $Q \times \Sigma$ to Q. This function can be extended to strings by writing $\delta(q, \lambda) = q$, and $\delta(q, u \cdot a) = \delta(\delta(q, u), a)$ for all $q \in Q$, $u \in \Sigma^*$ and $a \in \Sigma$. A string $u \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\delta(q_0, u) \in F$. The set of strings accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$ and called a *regular language*. The number of states of an automaton $\mathcal{A}$ is also called the *size* of $\mathcal{A}$. A DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is *complete* if for all q in Q and a in Σ, $\delta(q, a)$ is defined, i.e., δ is a total function. For any regular language L, there exists a minimum state DFA $\mathcal{A}_L$ such that $L(\mathcal{A}_L) = L$ (see [10], pp. 65-71).

A state q is called *reachable* if there exists $u \in \Sigma^*$ such that $\delta(q_0, u) = q$ and *co-reachable* if there exists $u \in \Sigma^*$ such that $\delta(q, u) \in F$. A reachable state that is not co-reachable is a *sink* state. Note that in a minimum DFA there is at most one sink state, and all states are reachable.

Given a language $L \subseteq \Sigma^*$, one can define the following relation on strings: $u_1 \equiv_L u_2$ if and only if for all u in Σ^*, $u_1 \cdot u \in L \Leftrightarrow u_2 \cdot u \in L$. It is easy to show that $\equiv_L$ is an equivalence relation, and thus it divides the set of all finite strings in Σ^* into one or more equivalence classes. We denote by $[u]_L$ (or simply $[u]$, when there is no confusion) the equivalence class of the string u (i.e., $\{u' \mid u' \equiv_L u\}$), and by $\Sigma^*/_{\equiv_L}$ the set of all equivalence classes induced by $\equiv_L$ on Σ^*.

The Myhill-Nerode Theorem states that the number of equivalence classes of $\equiv_L$ (also called the *index* of L) is equal to the number of states of $\mathcal{A}_L$. As a direct consequence, a language L is regular if and only its index is finite.

Assume that Σ is a totally ordered set, and let $\prec_{lex}$ be the lexicographical order on Σ^*. Then, the *lex-length order* $\prec$ on Σ^* is defined by: $u \prec v$ if either $|u| < |v|$, or else $|u| = |v|$ and $u \prec_{lex} v$. In other words, strings are compared first according to length and then lexicographically.

If $f : A \to B$ is a function, by $f(X)$ we denote the set $\{f(x) \mid x \in X\}$. Moreover, we say that f and g are equal if they have the same domain A, and $f(x) = g(x)$ for all $x \in A$.

2.1 Query Learning

Let $\mathcal{C}$ be a class of recursive languages over Σ^*. We say that $\mathcal{C}$ is an *indexable class* if there is an effective enumeration $(L_i)_{i \geq 1}$ of all and only the languages in $\mathcal{C}$ such that membership is uniformly decidable, i.e., there is a computable function that, for any $w \in \Sigma^*$ and $i \geq 1$, returns 1 if $w \in L_i$, and 0 otherwise. Such an enumeration will subsequently be called an *indexing* of $\mathcal{C}$. In the sequel we might say that $\mathcal{C} = (L_i)_{i \geq 1}$ is an indexable class and understand that $\mathcal{C}$ is an indexable class and $(L_i)_{i \geq 1}$ is an indexing of $\mathcal{C}$.

In the query learning model a learner has access to an oracle that truthfully answers queries of a specified kind. A *query learner* M is an algorithmic device that, depending on the reply on the previous queries, either computes a new query, or returns a hypothesis and halts.

More formally, let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class, M a query learner, and let $L \in \mathcal{C}$. We say that M learns L using some type of queries if it eventually halts and its only hypothesis, say i, correctly describes L, i.e., $L_i = L$. So, M returns its unique and correct guess i after only finitely many queries. Moreover, M learns $\mathcal{C}$ using some type of queries if it learns every $L \in \mathcal{C}$ using queries of the specified type. In the sequel we consider:

– *Membership queries.* The input is a string w, and the answer is 'yes' or 'no', depending on whether or not w belongs to the target language L.
– *Correction queries.* The input is a string w, and the answer is the smallest string (in lex-length order) of the set $\mathit{Tail}_L(w)$ if $w \in \mathit{Pref}(L)$, and the special symbol $\theta \notin \Sigma$ otherwise. We denote the correction of a string w with respect to the language L by $C_L(w)$.

The collection of all indexable classes $\mathcal{C}$ for which there is a query learner M such that M learns $\mathcal{C}$ using MQs (CQs) is denoted by *MemQ* (*CorQ*, respectively).

3 Learning *k*-Reversible Languages with CQs

Angluin introduces the class of k-reversible languages (henceforth denoted by *k-Rev*) in [3], and shows that it is inferable from positive data in the limit. Later on, she proves that there is no polynomial algorithm that exactly identifies DFAs for 0-reversible languages using only equivalence queries [6].

We study the learnability of the class *k-Rev* in the context of learning with CQs, and show that there is a polynomial time algorithm which identifies any k-reversible language after asking a finite number of CQs.

Although the original definition of k-reversible languages uses the notion of k-reversible automata, we will give here only a purely language-theoretic characterization.

Theorem 1 (Angluin, [3]). *Let L be a regular language. Then L is in k-Rev if and only if whenever u_1vw, u_2vw are in L and $|v| = k$, $\mathit{Tail}_L(u_1v) = \mathit{Tail}_L(u_2v)$.*

Let Σ be an alphabet, and $L \subseteq \Sigma^*$ be the target k-reversible language. For any string u in Σ^*, we define the function $row_k(u) : \Sigma^{\leq k} \to \Sigma^* \cup \{\theta\}$ by $row_k(u)(v) = C_L(uv)$. We show that each equivalence class in $\Sigma^*/_{\equiv_L}$ is uniquely identified by the values of function row_k on $\Sigma^{\leq k}$.

Proposition 1. *Let L be a k-reversible language. Then, for all $u_1, u_2 \in \Sigma^*$, $u_1 \equiv_L u_2$ if and only if $row_k(u_1) = row_k(u_2)$.*

Proof. Let us first notice that for all regular languages L and for any $k \in \mathbb{N}$, $u_1 \equiv_L u_2 \Rightarrow row_k(u_1) = row_k(u_2)$ (by the definition of function row_k), so we just have to show that $row_k(u_1) = row_k(u_2) \Rightarrow u_1 \equiv_L u_2$.

Indeed, suppose there exist $u_1, u_2 \in \Sigma^*$ such that $row_k(u_1) = row_k(u_2)$ and $u_1 \not\equiv_L u_2$. Hence, there must exist w such that either

– $u_1w \in L$ and $u_2w \notin L$, or
– $u_1w \notin L$ and $u_2w \in L$.

Let us assume the former case (the other one is similar).

1) If $|w| \leq k$, then $w \in \Sigma^{\leq k}$, and since $row_k(u_1) = row_k(u_2)$ we get in particular $row_k(u_1)(w) = row_k(u_2)(w)$, that is $C_L(u_1w) = C_L(u_2w)$. But $u_1w \in L$ implies $C_L(u_1w) = \lambda$, and so $C_L(u_2w) = \lambda$ which is in contradiction with $u_2w \notin L$.
2) If $|w| > k$, then there must exist $v, w' \in \Sigma^*$ such that $w = vw'$ and $|v| = k$. Moreover, by assumption $u_1vw' \in L$ and $u_2vw' \notin L$, so $u_1v \not\equiv_L u_2v$. On the other hand since $row_k(u_1) = row_k(u_2)$ and $v \in \Sigma^{\leq k}$, we have $row_k(u_1)(v) = row_k(u_2)(v)$, that is $C_L(u_1v) = C_L(u_2v) = v'$. Because $u_1v \cdot w' \in L$, $Tail_L(u_1v) \neq \emptyset$ and hence $C_L(u_1v) \in \Sigma^*$. Since $L \in k\text{-}Rev$, $u_1vv' \in L, u_2vv' \in L$ and $|v| = k$, we get $Tail_L(u_1v) = Tail_L(u_2v)$ (*cf.* Theorem 1) which is in contradiction with $u_1v \not\equiv_L u_2v$.

□

This result tells us that if $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ is the minimal complete automaton for the k-reversible language L, then the values of function $row_k(u)$ on $\Sigma^{\leq k}$ uniquely identify the state $\delta(q_0, u)$. We use this property to show that k-reversible languages are learnable in polynomial time with CQs.

3.1 The Algorithm

The algorithm follows the lines of L^*. We have an *observation table* denoted by (S, E, C) in which lines are indexed by the elements of a prefix-closed set S, columns are indexed by the elements of a suffix-closed set E, and the element of the table situated at the intersection of line u with column v is $C_L(uv)$.

We start with $S = \{\lambda\}$ and $E = \Sigma^{\leq k}$, and then increase the size of S by adding elements with distinct row values. An important difference between our algorithm and L^* is that in our case the set E is never modified during the run of the algorithm (in L^*, E contains only one element in the beginning, and it is gradually enlarged when needed).

We say that the observation table (S, E, C) is *closed* if for all $u \in S$ and $a \in \Sigma$, there exists $u' \in S$ such that $row_k(u') = row_k(ua)$. Moreover, (S, E, C) is *consistent* if for all $u_1, u_2 \in S$, $row_k(u_1) \neq row_k(u_2)$. It is clear that if the table (S, E, C) is consistent and S has exactly n elements, where n is the index of L, then the strings in S are in bijection with the elements of $\Sigma^*/_{\equiv_L}$.

For any closed and consistent table (S, E, C), we construct the automaton $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$ as follows. $Q := \{row_k(u) \mid u \in S\}$, $q_0 := row_k(\lambda)$, $F := \{row_k(u) \mid u \in S \text{ and } C_L(u) = \lambda\}$, and $\delta(row_k(u), a) := row_k(ua)$ for all $u \in S$ and $a \in \Sigma$.

To see that this is a well-defined automaton, note that since S is a non-empty prefix-closed set, it must contain λ, so q_0 is defined. Because S is consistent, there are no two elements u_1, u_2 in S such that $row_k(u_1) = row_k(u_2)$. Thus, F is well defined. Since the observation table (S, E, C) is closed, for each $u \in S$ and $a \in \Sigma$, there exists u' in S such that $row_k(ua) = row_k(u')$, and because it is consistent, this u' is unique. So δ is well defined.

Remark 1. The following statements are true.

1) $row_k(u)$ is a sink state if and only if $C_L(u) = \theta$;
2) $\delta(q_0, u) = row_k(u)$ for all u in $S \cup S\Sigma$.

We present a polynomial time algorithm that learns any k-reversible language L after asking a finite number of CQs.

Algorithm 1 An algorithm for learning the class *k-Rev* with CQs

```
1: $S := \{\lambda\}$, $E := \Sigma^{\leq k}$
2: $closed :=$ TRUE
3: update the table by asking CQs for all strings in $\{uv \mid u \in S \cup S\Sigma, v \in E\}$
4: repeat
5:   if $\exists u \in S$ and $a \in \Sigma$ such that $row_k(ua) \notin row_k(S)$ then
6:     add $ua$ to $S$
7:     update the table by asking CQs for all strings in $\{uaa'v \mid a' \in \Sigma, v \in E\}$
8:     $closed :=$ FALSE
9:   end if
10: until $closed$
11: output $\mathcal{A}(S, E, C)$ and halt.
```

Note that since the algorithm adds to S only elements with distinct row values, the table (S, E, C) is always consistent. We will see that as long as $|S| < n$, it is not closed.

Lemma 1. *If $|S| < n$, then (S, E, C) is not closed.*

Proof. Let us assume that there exists $m < n$ such that $|S| = m$ and the table (S, E, C) is closed. Let $\mathcal{A}_L = (Q', \Sigma, \delta', q_0', F')$ be the minimal complete automaton accepting L, and $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$.

We define the function $\varphi : Q \to Q'$ by $\varphi(row_k(u)) := \delta'(q_0', u)$. Note that φ is well-defined because there are no two strings u_1, u_2 in S such that $row_k(u_1) = row_k(u_2)$. Moreover, it is injective since $\varphi(row_k(u_1)) = \varphi(row_k(u_2))$ implies $\delta'(q_0', u_1) = \delta'(q_0', u_2)$ which is equivalent to $[u_1] = [u_2]$, and *cf.* Proposition 1, to $row_k(u_1) = row_k(u_2)$. We show that φ is a morphism of automata from $\mathcal{A}(S, E, C)$ to $\mathcal{A}_L$, that is: $\varphi(q_0) = q_0'$, $\varphi(F) \subseteq F'$, and $\varphi(\delta(row_k(u), a)) = \delta'(\varphi(row_k(u)), a)$ for all $u \in S$ and $a \in \Sigma$.

Clearly, $\varphi(q_0) = \varphi(row_k(\lambda)) = \delta'(q_0', \lambda) = q_0'$. Let us now take $row_k(u)$ in F, that is, $u \in S$ and $C_L(u) = \lambda$. Since $\varphi(row_k(u)) = \delta'(q_0', u)$ and $u \in L$, it follows that $\varphi(row_k(u)) \in F'$. Finally, $\varphi(\delta(row_k(u), a)) = \varphi(row_k(ua))$ $= \varphi(row_k(v))$ for some v in S such that $row_k(ua) = row_k(v)$ (the table is closed), and $\delta'(\varphi(row_k(u)), a) = \delta'(\delta'(q_0', u), a) = \delta'(q_0', ua)$. It is enough to see that $\varphi(row_k(v)) = \delta'(q_0', v) = \delta'(q_0', ua)$ (because by Proposition 1, $row_k(v) =$

$row_k(ua)$ implies $[v] = [ua]$, and $\mathcal{A}_L$ is the minimal automaton accepting L) to conclude the proof.

We have constructed an injective morphism from $\mathcal{A}(S, E, C)$ to $\mathcal{A}_L$ such that $|Q| = m < n = |Q|$. Since both $\mathcal{A}(S, E, C)$ and $\mathcal{A}_L$ are complete automata, this leads to a contradiction. □

We show that Algorithm 1 cannot be used for the whole class of regular languages.

Lemma 2. *Algorithm 1 does not work in general for arbitrary regular languages.*

Proof. Indeed, let us assume that Algorithm 1 can identify any regular language. Let us fix $k \geq 0$, and consider the language $L_k = \{ab^k a, ab^k b, b^{k+1}a\}$ which is finite, and hence regular. The minimal complete DFA of L_k is represented in Figure 1.

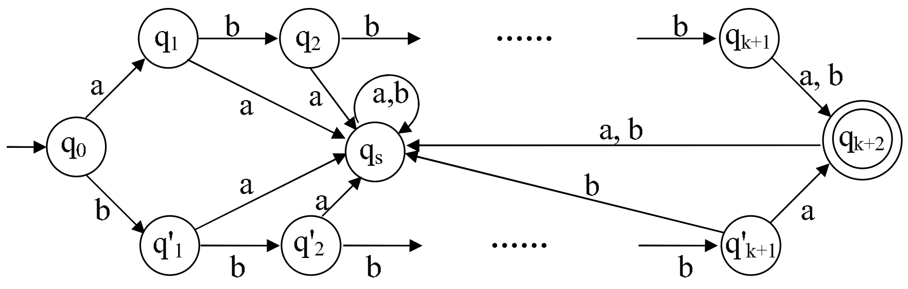

Fig. 1. The automaton $\mathcal{A}_{L_k}$

Since the strings $a \cdot b^k \cdot a$ and $b \cdot b^k \cdot a$ are both in L_k, and $Tail_{L_k}(a \cdot b^k) = \{a, b\} \neq \{a\} = Tail_{L_k}(b \cdot b^k)$, the language L_k is not k-reversible (Theorem 1).

When running the algorithm on L_k, the set S is initialized with the value $\{\lambda\}$. Then, since both $row_k(a)$ and $row_k(b)$ are different from $row_k(\lambda)$, one of the two elements is added to S. Note that for all u in $\Sigma^{\leq k}$, $row_k(a)(u) = row_k(b)(u)$ because:

- if $u = b^i$ with $0 \leq i \leq k$, then $C_{L_k}(au) = b^{k-i}a = C_{L_k}(bu)$, and
- if $u = \Sigma^{\leq k} \backslash \{b^i \mid 0 \leq i \leq k\}$, then $C_{L_k}(au) = \theta = C_{L_k}(bu)$.

Hence, $row_k(a) = row_k(b)$. But this implies that in the automaton output by the algorithm, the strings a and b represent the same state, a contradiction. □

In the following sections we show that the algorithm runs in polynomial time, and terminates with the minimal automaton for the target language as its output.

3.2 Correctness and Termination

We have seen that as long as $|S| < n$, the table is not closed, so there will always be an u in S and a symbol a in Σ such that $row_k(ua) \notin row_k(S)$. Since the cardinality of the set S is initially 1, and increases by 1 with each "repeat-until" loop (lines 4–10), it will eventually be n, and hence the algorithm is guaranteed to terminate.

We claim that when $|S| = n$, the observation table (S, E, C) is closed and consistent, and $\mathcal{A}(S, E, C)$ is isomorphic to $\mathcal{A}_L$. Indeed if $|S| = n$, then the set $\{row_k(u) \mid u \in S\}$ has cardinality n, since the elements of S have distinct row values. Thus for all $u \in S$ and $a \in \Sigma$, $row_k(ua) \in row_k(S)$ (otherwise $[ua]$ would be the $(n + 1)^{\text{th}}$ equivalence class of $\Sigma^*/_{\equiv_L}$), and hence the table is closed.

To see that $\mathcal{A}(S, E, C)$ and $\mathcal{A}_L$ are isomorphic, let us take $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$, $\mathcal{A}_L = (Q', \Sigma, \delta', q'_0, F')$, and the function $\varphi : Q \to Q'$ defined by $\varphi(row_k(u)) := \delta'(q'_0, u)$ for all $u \in S$. As in the proof of Lemma 1, it can be shown that φ is a well-defined and injective automata morphism. Since the two automata have the same number of states, φ is also surjective, and hence bijective. Let us now show that $\varphi(F) = F'$. Indeed, take $q \in F'$. Because φ is bijective, there exists u in S such that $\varphi(row_k(u)) = q$. It follows immediately that $\delta'(q'_0, u) \in F'$, and hence $u \in L$. Thus, $C_L(u) = \lambda$ and $row_k(u) \in F$. Clearly, $\varphi(row_k(u)) = q \in \varphi(F)$. So, $F' \subseteq \varphi(F)$, and since $\varphi(F) \subseteq F'$, $\varphi(F) = F'$ which concludes the proof.

3.3 Time Analysis and Query Complexity

Let us now discuss the time complexity of the algorithm. While the cardinality of S is smaller than n, the algorithm searches for a string u in S and a symbol a in Σ such that $row_k(ua)$ is distinct from all $row_k(v)$ with $v \in S$. This can be done using at most $|S|^2 \cdot |\Sigma| \cdot |E|$ operations: there are $|S|$ possibilities for choosing u (and the same number for v), $|\Sigma|$ for choosing a, and $|E|$ operations to compare $row_k(ua)$ with $row_k(v)$. If we take $|\Sigma| = l$, the total running time of the "repeat-until" loop can be bounded by $(1^2 + 2^2 + \ldots + (n-1)^2) \cdot l \cdot (1 + l + l^2 + \ldots + l^k)$. Note that by "operations" we mean string comparisons, since they are generally acknowledged as being the most costly tasks.

On the other hand, to construct $\mathcal{A}(S, E, C)$ we need n comparisons for determining the final states, and at most $n^2 \cdot |\Sigma| \cdot |E|$ operations for constructing the transition function. This means that the total running time of the algorithm is bounded by $n + l \cdot \frac{l^{k+1}-1}{l-1} \cdot \frac{n(n+1)(2n+1)}{6}$, that is $O(n^3 l^k)$.

As for the number of queries asked by the algorithm, it can be bounded by $|S \cup S\Sigma| \cdot |E|$ (i.e., by the size of the final observation table), so the query complexity of the algorithm is $O(nl^k)$.

4 Pattern Languages

Initially introduced by Angluin [1] to show that there are non-trivial classes of languages learnable from text in the limit, the class of pattern languages has been intensively studied in the context of language learning ever since. Polynomial time algorithms have been given for learning pattern languages using one or more examples and queries [13], or just superset queries [5], or for learning k-variables pattern languages from examples [11], *etc.*

We assume a finite alphabet Σ such that $|\Sigma| \geq 2$, and a countable, infinite set of *variables* $X = \{x, y, z, x_1, y_1, z_1, \dots, \}$. A *pattern* π is any non-empty string over $\Sigma \cup X$. The *pattern language* $L(\pi)$ consists of all the words obtained by replacing the variables in π with arbitrary strings in Σ^+. Let us denote by $\mathcal{P}$ the set of all pattern languages over a fixed alphabet Σ.

We say that the pattern π is in *normal form* if the variables occurring in π are precisely $x_1, \dots, x_k$, and for every j with $1 \leq j < k$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1}.

Next we show that there exists an algorithm which learns $\mathcal{P}$ using a finite number of CQs.

4.1 The Algorithm

Suppose that the target language is a pattern language $L(\pi)$, where π is in normal form. Then the following algorithm outputs the pattern π after asking a finite number of CQs.

Algorithm 2 An algorithm for learning the class $\mathcal{P}$ with CQs

```
1: $w := C_L(\lambda), n := |w|, var := 0$
2: for $i := 1$ to $n$ do
3:    $\pi[i] := null$
4: end for
5: for $i := 1$ to $n$ do
6:    if $(\pi[i] = null)$ then
7:       choose $a \in \Sigma \backslash \{w[i]\}$ arbitrarily
8:       $v := C_L(w[1 \dots i-1]a), m := |v|$
9:       if $(|v| = |w[i+1, \dots, n]|)$ then
10:         $var := var + 1, \pi[i] := x_{var}$
11:         for all $j \in \{1, \dots, m\}$ for which $v[j] \neq w[i+j]$ do
12:            $\pi[i+j] := x_{var}$
13:         end for
14:      else
15:         $\pi[i] := w[i]$
16:      end if
17:   end if
18: end for
19: output $\pi$
```

4.2 Correctness and Termination

The correctness of the algorithm is based on the following observation. If w is the smallest string (in lex-length order) in $L(\pi)$ and $n = |w|$, then for all i in $\{1, \dots, n\}$, we have:

- if $\pi[i]$ is a variable x such that i is the position of the leftmost occurrence of x in π, then $|C_L(w[1, \dots, i-1]a)| = |w[i+1, \dots, n]|$ for any symbol $a \in \Sigma$; moreover, we can detect the other occurrences of the variable x in π by just checking the positions where the strings $C_L(w[1, \dots, i-1]a)$ and $w[i+1, \dots, n]$ do not coincide, where a is any symbol in $\Sigma \setminus \{w[i]\}$;
- if $\pi[i] = a$ for some a in Σ, then for all $b \in \Sigma \setminus \{a\}$, $C_L(w[1, \dots, i-1]b)$ is either θ, or longer than $w[i+1, \dots, n]$.

Obviously, the algorithm terminates in finite steps.

4.3 Time Analysis and Query Complexity

For each symbol in the pattern, the algorithm makes at most $n+1$ comparisons, where n is the length of the pattern. This implies that the total running time of the algorithm is bounded by $n(n+1)$, that is $O(n^2)$.

It is easy to see that the query complexity is linear in the length of the pattern since the algorithm does not ask more than $n+1$ CQs.

5 Learning with CQs versus Learning with MQs

The notion of CQ appeared as an extension of the well-known and intensively studied MQ. The inspiration for introducing them comes from a real life setting (which is the case for MQs also): when children make mistakes, the adults do not reply by a simple 'yes' or 'no' (the agreement is actually implicit), but they also provide them with a corrected word. Clearly, CQs can be thought as some more informative MQs. So, it is only natural to compare the two learning settings (learning with CQs *vs.* learning with MQs), and to analyze their expressive power.

The first step in this direction has already been done: C. Tîrnăucă and S. Kobayashi showed in [16] that learning with CQs is strictly more powerful than learning with MQs, when we neglect the time complexity.

In this section we make a step further towards understanding the differences and similarities between these two learning models by taking into consideration the efficiency of the learning algorithms, that is, the time complexity. For this, we need some further terminology.

Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. We say that $\mathcal{C}$ is *polynomially learnable with MQs* (or *with CQs*) if there exists a polynomial time algorithm which learns $\mathcal{C}$ using MQs (CQs, respectively). We denote the collection of all indexable classes $\mathcal{C}$ which are polynomially learnable with MQs by *PolMemQ* (*PolCorQ* is defined similarly).

Recall that if the correction for a given string u is λ, then the string is in the language, and the oracle's answer would be 'yes'; in all other cases, the string is not in the language, and the answer would be 'no'. Since the answer to any CQ gives us also the answer to the corresponding MQ, it follows immediately that the class *PolMemQ* is included in *PolCorQ*. We show that the inclusion is strict using pattern languages as the separating case.

Theorem 2. *The class $\mathcal{P}$ is in PolCorQ$\backslash$PolMemQ.*

Proof. It is clear that $\mathcal{P}$ is in *PolCorQ* since Algorithm 2 is a polynomial time algorithm which identifies any pattern language using COs (see Section 4).

Assume now that $\mathcal{P}$ is in *PolMemQ*, and consider the class of singletons $\mathcal{S}$ of fixed length n over the alphabet Σ. Because every language $L = \{w\}$ in $\mathcal{S}$ can be written as a pattern language ($L = L(w)$, where w is a pattern without any variables), $\mathcal{S}$ is also in *PolMemQ*. But Angluin shows that, if l is the cardinality of the alphabet, then any algorithm which learns $\mathcal{S}$ using MQs needs to ask at least $l^n - 1$ MQs [2], which leads to a contradiction. □

Note that although $\mathcal{P}$ is not polynomially learnable with MQs, it is in *MemQ* (see [14], page 266). However, there are classes of languages in *PolCorQ* which cannot be learned at all (polynomially or not) using MQs, as we will see in the sequel.

Theorem 3. *The class k-Rev is in PolCorQ$\backslash$MemQ.*

Proof. Since Algorithm 1 learns any k-reversible language using CQs in polynomial time (see Section 3), it follows immediately that *k-Rev* is in *PolCorQ*.

To show that *k-Rev* is not in *MemQ*, we use Mukouchi's characterization of the class *MemQ* in terms of pairs of definite finite tell-tales. A pair $\langle T, F\rangle$ is said to be a *pair of definite finite tell-tales of* L_i if:

(1) T_i is a finite subset of L_i, F_i is a finite subset of $\Sigma^*\backslash L_i$, and
(2) for all $j \geq 1$, if L_j is *consistent with the pair* $\langle T, F\rangle$ (that is, $T \subseteq L_j$ and $F \subseteq \Sigma^*\backslash L_j$), then $L_j = L_i$.

Mukouchi proves in [14] that an indexable class $\mathcal{C} = (L_i)_{i\geq 1}$ belongs to *MemQ* if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any index i.

So, let us assume that *k-Rev* is in *MemQ*. Consider the alphabet Σ such that $\{a, b\} \subseteq \Sigma$, and the language $L = \{a\}$. Clearly, L is in *k-Rev* for all $k \geq 0$ and hence a pair of definite finite tell-tales $\langle T, F\rangle$ is computable for L. This means that $T \subseteq L$ and F is a finite set included in $\Sigma^*\backslash\{a\}$. Let us take $m = \max\{|w| \mid w \in F\}$ and the language $L' = \{a, ba^m b\}$. It is clear that L' is in *k-Rev* for all $k \geq 0$, and that it is consistent with $\langle T, F\rangle$. Moreover, $L' \neq L$ which leads to a contradiction. □

On the other hand, very simple classes of languages cannot be learned in polynomial time using CQs. For example, if we take $\bar{\mathcal{S}}$ to be $\bar{\mathcal{S}} = (L_w)_{w\in\Sigma^*}$, where $L_w = \Sigma^*\backslash\{w\}$, then any algorithm would require at least $1+l+l^2+\ldots+l^n$ CQs in order to learn L_w, where $n = |w|$ and $l = |\Sigma|$.

6 Concluding Remarks

We have investigated the learnability of some well-known language classes in the query learning setting. Figure 2 illustrates a synthesis of the results obtained.

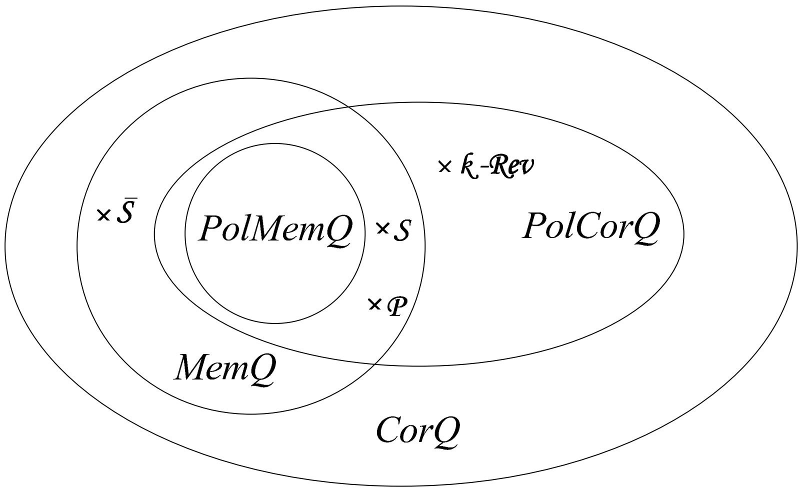

Fig. 2. CQ learning vs MQ learning

The class of pattern languages was known to be learnable with MQs. We gave a polynomial time algorithm for learning $\mathcal{P}$ using CQs, and showed that they cannot be efficiently learned with MQs. Moreover, we proved that k-reversible languages are efficiently learnable with CQs, and not learnable (at all) with MQs.

For the future, we would like to see what happens with the learnability results obtained so far when we change the correcting string. A possible direction could be to choose as correction the closest string in the edit distance.

Acknowledgments

Many thanks to Magnus Steinby for fruitful discussions and valuable advices.

References

1. Angluin, D.: Finding patterns common to a set of strings (extended abstract). In: Proc. 11th Annual ACM Symposium on Theory of Computing (STOC '79), New York, NY, USA, ACM Press (1979) 130–141
2. Angluin, D.: A note on the number of queries needed to identify regular languages. Information and Control **51**(1) (1981) 76–87

3. Angluin, D.: Inference of reversible languages. Journal of the ACM **29**(3) (1982) 741–765
4. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
5. Angluin, D.: Queries and concept learning. Machine Learning **2**(4) (1988) 319–342
6. Angluin, D.: Negative results for equivalence queries. Machine Learning **5**(2) (1990) 121–150
7. Beccera-Bonache, L., Dediu, A.H., Tîrnăucă, C.: Learning DFA from correction and equivalence queries. In: Proc. 8th International Colloquium on Grammatical Inference (ICGI '06). Volume 4201 of Lecture Notes in Artificial Intelligence, Berlin, Heidelberg, Springer-Verlag (2006) 281–292
8. Drewes, F., Högberg, J.: Query learning of regular tree languages: How to avoid dead states. Theory of Computing Systems **40**(2) (2007) 163–185
9. Gold, E.M.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)
11. Kearns, M., Pitt, L.: A polynomial-time algorithm for learning k-variable pattern languages from examples. In: Proc. 2nd Annual Workshop on Computational Learning Theory (COLT '89), San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1989) 57–71
12. Martín-Vide, C., Mitrana, V., Păun, G., eds.: Formal Languages and Applications. Studies in Fuzzyness and Soft Computing 148. Berlin, Heidelberg, Springer-Verlag(2004)
13. Marron, A., Ko, K.I.: Identification of pattern languages from examples and queries. Information and Computation **74**(2) (1987) 91–112
14. Mukouchi, Y.: Characterization of finite identification. In: Proc. 3rd International Workshop on Analogical and Inductive Inference (AII '92). Volume 642 of Lecture Notes in Artificial Intelligence, London, UK, Springer-Verlag (1992) 260–267
15. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theoretical Computer Science **76** (1990) 223–242
16. Tîrnăucă, C., Kobayashi, S.: A characterization of the language classes learnable with correction queries. In: Proc. 4rd International Conference on Theory and Applications of Models of Computation (TAMC '07). Volume 4484 of Lecture Notes in Computer Science, Berlin, Heidelberg, Springer-Verlag (2007) 398–407
17. Tîrnăucă, C.I., Tîrnăucă, C.: Learning regular tree languages from correction and equivalence queries. Journal of Automata, Languages and Combinatorics, Special Issue WATA 2006 (2007). To appear.
18. Valiant, L.G.: A theory of the learnable. Communications of the ACM **27**(11) (1984) 1134–1142
19. Yokomori, T.: Learning non-deterministic finite automata from queries and counterexamples. Machine Intelligence **13** (1994) 169–189
20. Yokomori, T.: Learning two-tape automata from queries and counterexamples. Mathematical Systems Theory **29**(3) (1996) 259–270