Universitat Rovira i Virgili
Facultat de Lletres
Departament de Filologies Romàniques

Language Learning with Correction Queries

PhD Dissertation

Cristina Tîrnăucă

Supervised by
Colin de la Higuera and Victor Mitrana

Tarragona, Spain, 2009

SUPERVISORS :

Professor Colin DE LA HIGUERA
Laboratoire Hubert Curien
Jean Monnet University
18 Rue du Professeur Benoît Lauras
42000 Saint-Etienne
FRANCE

Professor Victor MITRANA
Faculty of Mathematics and Computer Science
University of Bucharest
Str. Academiei 14
70109 Bucharest
ROMANIA

TUTOR :

Dr. Gemma BEL ENGUIX
Grup de Recerca en Lingüística Matemàtica
Universitat Rovira i Virgili
Pl. Imperial Tàrraco 1
43005 Tarragona
SPAIN

Contents

Preface

I would not be here today if it was not for my dear friend and colleague Mihai Ionescu, who persuaded me to continue my studies and to enroll in a PhD program in Tarragona. Also, I would like to address my gratitude to my mother, first of all, for raising me the way she did, and secondly, for her altruistic advice to pursue an academic career, even if this meant that I would be so far away from her.

I am deeply grateful to my supervisors, Colin de la Higuera and Victor Mitrana. To Colin, for introducing me to the field, guiding me along the way, and being my source for a consistent reality check. And to Victor, for his constant help, from proof-reading my first paper to the present time.

I have benefited greatly from the collaboration with my co-authors. Many thanks to Leonor Becerra Bonache, who came up with the idea of corrections - the backbone of my thesis, and to Adrian Horia Dediu, who co-opted me into writing what became my very first paper. I would also like to thank Cătălin Tîrnăucă, who got me interested in the world of tree languages. I enjoyed working with Timo Knuutila, who is both an excellent programmer and a talented theoretician. I owe much to Satoshi Kobayashi, with whom I did a substantial part of the work presented here, and who has closely followed and guided my research.

Over the last four years I have been very fortunate to meet fantastic people, excellent researchers and good friends. Many of them had their own contribution to this dissertation by sharing ideas, providing feedback, suggesting hints for proofs, or proof-reading and improving the contents of the present manuscript. Due to space limitations, I will only list (in alphabetical order) some of them: Amittai Axelrod, Zoltan Ésik, Szilárd Fazekas, Sanjay Jain, Efim Kinber, Andreas Maletti, Robert Mercaş, Alexander Okhotin, Jason Riesa, Magnus Steinby, Osamu Watanabe, Thomas Zeugmann.

Thanks are due also to the members of the GRLMC research group and to the professors of our PhD school. I have acquired a very good background in formal language theory, and I have learned a great deal from our weekly

seminars and other fruitful discussions. I would especially like to thank my colleagues for providing such a warm and friendly environment (the meetings from Plaça de la Font will not be forgotten). I will always be grateful to the head of our research group, Carlos Martín Vide, who was a permanent source of down-to-earth advice and constant support. Special thanks to our assistant and my good friend, Lilica Voicu, who helped me through all bureaucratic matters.

Additionally, none of this would have been possible without the financial support of the FPU fellowship AP2004-6968 from the Spanish Ministry of Education and Science, which also facilitated my stays in the University of Electro-Communication, Tokyo, Japan and in the Hubert Curien Laboratory, Saint-Étienne, France. I would also like to acknowledge the financial support of the European Science Foundation (ESF), under the AutoMathA scheme, for sponsoring my visit to the Department of Mathematics of Turku University, Finland. Further, a great part of the actual writing of the thesis was done during my stay at the Information Sciences Institute (ISI) of the University of Southern California (USC), Los Angeles, USA. I would like to thank Kevin Knight and his group for their hospitality.

Special thanks to all my family members, especially to my beloved husband, for his patience and kindness, for supporting me and encouraging me all the way, and most importantly, for being willing to relocate so I could follow my dream. To my sister and her family, to my niece Marika and my nephew Alessandro, whom I hope will be able to read this one day. To my grandmother, who missed me so much, and yet never asked me to come back.

I would like to dedicate this work to the memory of my father, who always told me, since I was a little child, that English and Computer Science represent the future, and I had better know both.

Abstract

In the field of grammatical inference, the goal of any learning algorithm is to identify (or to output a good enough approximation of) a target concept from a given class by having access to a specific type of information. The main learning settings are Gold's model of learning in the limit [Gol67], Angluin's query learning model [Ang87c] and Valiant's model of probably approximately correct (PAC) learning [Val84]. This dissertation is primarily concerned with the second approach.

We thoroughly investigate a recently introduced, linguistic motivated, type of query called Correction Query (CQ). We consider three possible definitions, and for each of them we give necessary and sufficient conditions for a language class to be learnable with these types of queries. Furthermore, we compare the model of learning with CQs with other well-known Gold-style and query learning models when no efficiency constraints are imposed. Results are also obtained for the restricted version of the model of learning with CQs in polynomial time.

Additionally, we discuss the learnability of deterministic finite automata (DFAs) with correction and equivalence queries. We design several learning algorithms and we present a comparison between our algorithms and the standard L^* algorithm for learning DFAs with membership and equivalence queries. These results are furthermore extended from string languages to tree languages.

Chapter 1

Introduction

No formal model can currently express all aspects of human learning. Nevertheless, the overall aim of researchers working in grammatical inference has been to gain a better understanding of what learning really is, as E.M. Gold points out in his pioneering paper [Gol67].

> *I wish to construct a precise model for the intuitive notion "able to speak a language" [...] My justification for studying identifiability in the limit is this: A person does not know when he is speaking a language correctly; there is always the possibility that he will find that his grammar contains an error. But we can guarantee that a child will eventually learn a natural language, even if it will not know when it is correct.* **E.M. Gold [Gol67]**

Gold views learning as an infinite process. At each time the learner receives a unit of information and has to make a guess as to the identity of the unknown language on the basis of the information received so far. In Gold's model, the learner has access to either a growing sequence of positive examples, i.e., strings in the target language (*learning from text*), or both positive and negative information (*learning from informant*). A class of languages will be considered *learnable* with respect to the specified method of presenting the information if there is an algorithm that the learner can use to make his guesses, having the following property: given any language of the class, there is some finite time after which the guesses will all be the same and they will be correct. Similarly, children progressively acquire their native language, but no one can accurately indicate a precise moment for when it happened.

The *Probably Approximately Correct* (PAC) learning model proposed by L.G. Valiant [Val84] reflects another feature of the human language acquisition process. In the PAC model, the learner has to output, with high probability, a language that is close enough to the target one (see [KV94] for further details). This formalism captures our intuition that, although many of us might never master to perfection a language, we should be able to get a good enough approximation eventually.

Children's ability to ask questions within the process of acquiring their native language has also been modeled by formal language theoreticians.

> *...a technique that humans use to learn (asking questions) is also advantageous to machines that are trying to learn. Inquisitiveness is the driving force behind all of science.* **W.I. Gasarch and C.H. Smith [GS92]**

In the query learning model, the *learner* receives information about a target concept by asking specific queries to be truthfully answered by the *teacher* (or the *oracle*). The learner is required to return its hypothesis after finitely many queries, and this should be the correct one.

There are also hybrid models combining, for example, positive information with queries (see, e.g., [Val84, Sak95, TTWT04]), a representation of our intuition that children are not only widely exposed to words and sentences in the language, but they can also enrich their knowledge by asking questions.

So, researchers have always been looking for formal models that can describe as accurately as possible the way humans learn,

> *On the other hand, when we consider how a child learns a language communicating with a teacher, it becomes clear that these models* [Gold's models of learning in the limit from text and informant] *reflect two extremes of this process: positive data only is certainly less than what a child actually gets in the learning process, while informant (the characteristic function of the language) is much more than what a learner can expect (see, for example, [BH70, DPS86, HPTS84]).* **S. Jain and E. Kinber [JK08]**

and the ability of asking queries is playing an important role.

The first query learning algorithm, called L^*, identifies any deterministic finite automaton (DFA) in polynomial time by asking two types of queries: *membership queries* (MQs) and *equivalence queries* (EQs) [Ang87c]. Meanwhile, other types of queries have been introduced: subset, superset, disjointness and exhaustive queries [Ang88], structured membership queries [Sak90], etc. None of these queries reflect an important aspect of human language acquisition, namely that although children are not explicitly provided negative examples (i.e., words that are not in the language or ungrammatical sentences), they are corrected when they make mistakes.

...the child receives negative instances by being corrected in a way we do not recognize. **E.M. Gold [Gol67]**

Therefore, we study a recently introduced type of query, called CORRECTION QUERY, which copes with this particularity of child's language acquisition.

Defining a formal concept that corrects all possible mistakes one can make in either written or spoken natural language is a difficult task. The following examples illustrate the variety of error types one can encounter:

- Grammatical errors:
  - subject-verb agreement ("The majority of people *are* Rh-positive"),
  - verb tense agreement ("Did you *sent* me the postcard?"),
  - noun-number agreement ("All birds are very good at building their *nest*")
- Word Choice: bring/take ("*Take* the supplies to my house so we can work on the project")
- Pronunciation: escape, pronounced *excape*;
- Punctuation: ("Hi Michael" instead of "Hi, Michael")
- Spelling: *wierd* instead of weird

However, research in grammatical inference is focused on formal grammars rather than on grammars of natural languages. Therefore, we concentrate in this dissertation on defining and analyzing correction queries (CQs) that may be used (alone or in combination with other types of queries) in formal language theory.

Correction queries were first mentioned in [BBY04] as a new learning model for families of mildly context-sensitive languages.

Correction queries are an extension of membership queries [...] In the case of correction queries, if the answer is "no", then a corrected string is returned. However, in this paper, we will study the learnability in the limit from only positive data [...] In our future research schema, by correction queries we intend to define as an oracle that takes a string w as input and produces as output a corrected string w_c (if w is close to an element w_c of the target L) and "no" (otherwise), where w being close to w_c is defined by a certain measure (such as "one-letter" difference or Hamming distance).

L. Becerra Bonache and T. Yokomori [BBY04]

The idea of extending MQs by providing feedback when the queried string is not in the target language appears also in [JK07]. Jain and Kinber motivate the use of a *nearest positive example* by an observation discussed, in particular, in [RP99].

> *...while learning a language, in addition to overt explicit negative evidence (when a parent points out that a certain statement by a child is grammatically incorrect), a child often receives also covert explicit evidence in form of corrected or rephrased utterances.*
> **S. Jain and E. Kinber [JK07]**

The first formal definition of CQs appears in a paper by L. Becerra Bonache, A.H. Dediu and myself [BeBiDe05], in which the given algorithm - a straightforward modification of Angluin's L* - allows the learner to identify any minimal complete DFA from CQs and EQs in polynomial time. In order to distinguish these queries from all the other types subsequently introduced, we will refer to them as *prefix correction queries* (PCQs) throughout this thesis. A learner, in response to a PCQ, gets the smallest (in the lex-length order) extension of the queried datum such that their concatenation belongs to the target language.

Note that in the case of DFAs, returning the answer to a PCQ can be easily done in polynomial time. However, when the target concept ranges over arbitrary recursive languages, the answer to a PCQ might be very long or not even computable (given a recursive language L and a string w, one cannot decide, in general, if w is a prefix of a string in L). A possible solution to avoid very long (or infinite) searches is to restrict the search space to only short enough suffixes. Therefore, we introduce the notion of *length bounded correction query* (LBCQ). Given a fixed number l, answering an l-bounded correction query consists in returning all strings having the length smaller than or equal to l that, concatenated with the queried datum, form a string in the target language [Tîr08a, Tîr08b].

The third main type of CQs is based on the edit distance, and has been independently introduced by E. Kinber in [Kin08] and by L. Becerra Bonache, C. de la Higuera, J.C. Janodet and F. Tantini in [BBdlHJT07]. According to [BBdlHJT07], the *edit distance correction* of a given string is, by definition, one string in the target language at minimum distance from the queried datum. If many such strings exist, the teacher randomly returns one of them. A slightly modified type of edit distance correction query (EDCQ) is introduced by Kinber in [Kin08]. The author there imposes a supplementary condition: the teacher has to return only strings that have not previously appeared during the run of the algorithm. Clearly, this restriction makes sense only with respect to an algorithm, so we will consider this type of EDCQ only in the chapter dedicated to polynomial time learning algorithms.

Several questions arise: if we are given a class of languages, what kind of CQs should we use? What is the power of CQs? Can we compare them with other query learning models? What about other Gold-style learning models? Which of them provides the learner with more information? Can we build efficient algorithms that learn well-known classes of languages with CQs? If we combine them with EQs, can we outperform MQs? Is there a straightforward extension of CQs that deals with trees? The main thrust of this thesis is to address these questions.

The first topic we discuss is their power. More precisely, we compare the model of learning with CQs with other established learning models. While Gold-style learning and query learning seem to be two very different models, there are strong correlations between them. In [LZ04b, LZ04c], S. Lange and S. Zilles provide characterizations of different models of Gold-style learning (learning in the limit, conservative inference, and behaviorally correct learning) in terms of query learning for indexed families of recursive languages. As an example, they show that the collection of all language classes learnable with MQs coincides with the collection of those that are finitely identifiable from informant, and that learning with EQs is equally powerful as learning in the limit from informant. The study is furthermore extended to cover different hypotheses spaces: uniformly recursive, uniformly recursively enumerable, and uniformly K-recursively enumerable families (for definitions and details, see [LZ04a, LZ05, JLZ05, JLZ07]). In this dissertation, we compare the models of learning with CQs with both Gold-style and query learning models.

Which techniques should one use in order to compare our model with others? Obviously, some sort of characterization is needed. D. Angluin [Ang80b] is the first one to observe the connection between the learnability of an indexed family from text and the existence of a finite set (called *tell-tale*) that uniquely *identifies* each language (that is, no other language in the class that contains the tell-tale is properly included in the given language). She shows that an indexed family is identifiable in the limit from text if and only if there exists a procedure that enumerates, for each language in the class, a finite tell-tale. Twelve years later, the class of languages finitely identifiable from text (informant) is independently described by Y. Mukouchi [Muk92a] and S. Lange and T. Zeugmann [LZ92] in terms of definite finite tell-tales (pairs of definite finite tell-tales, respectively). They show that an indexed family is finitely learnable from text (informant) if and only if one can compute a (pair of) definite finite tell-tales for each language. We do something similar for the model of learning with CQs, by providing necessary and sufficient conditions in terms of what we call *triples of definite finite tell-tales*.

So far, we have been talking about the power of CQs in the general framework, without the requirement for the algorithms to be polynomial. What happens when we do impose efficiency constraints? Are the relations between CQs and MQs preserved? In order to answer these questions, we focus on finding language classes that would separate the two learning models. With that in mind, we give polynomial time algorithms for two well-known language classes: the class of pattern languages (that is, non-erasing pattern languages) and the class of k-reversible languages, both introduced by Angluin (in [Ang79] and [Ang82], respectively) and shown to be learnable from text in the limit.

Pattern languages have been widely investigated (see, e.g., [Sal94],[Sal95] or [SA95, Mit99] for an overview). Polynomial time algorithms have been given for learning from examples subclasses of regular patterns [CJR$^+$03, CJR$^+$06] and k-variable patterns [Lan90, KP89], or for learning from (one or more) examples and queries [MK87, Mar88, LW90, LW91, MS97]. More recent research concerns the learnability of erasing pattern languages in the query learning model [LZ03, NL05]. However, note that deciding membership for the pattern languages is NP-complete (and hence, answering PCQs cannot be done in deterministic polynomial time either). Therefore, any learning algorithm testing membership will be infeasible in practice (because of the impossibility to implement such an oracle) under the usual assumption that P $\neq$ NP.

Not so much has been done for the k-reversible languages. They were known to be learnable from text in the limit since their introduction. Other results concerning this class include the algorithm given by S. Kobayashi and T. Yokomori [KY97] for approximately learning regular languages having k-reversible languages as hypotheses space (the algorithm outputs the smallest k-reversible language that includes the target language), and the general algorithm for function distinguishable languages [Fer00] given by H. Fernau, algorithm that can be applied, as a special case, to the class of k-reversible languages. Regarding the learnability of this class in the query learning model, it is clear that the combination of MQs and EQs allows a learner to identify any k-reversible language from a minimally adequate teacher (the whole class of regular languages is learnable this way). Notice that neither MQs[Ang81], nor EQs [Ang90] alone are enough to allow polynomial time learning in this setting.

Once we determine what can be done with CQs alone, efficiently or not, we move on to the identification of deterministic finite automata. The emphasis on DFA learning is due to the fact that algorithms regarding the inference problem for DFAs can be nicely adapted for larger classes of grammars, for instance even linear grammars [SG94, Tak88, Tak94, Tak95], subsequential transducers [OGV93], context-free grammars [OGV93, Sak92], when the data is presented as unlabeled trees, or regular tree languages [DH03, DH07].

DFA learning has an extensive history, both in passive learning and in active learning. Intractability results by Gold [Gol78] and Angluin [Ang78] show that finding the smallest automaton consistent with a set of accepted and rejected strings is NP-complete. Nevertheless, a lot of work has been done on learning DFAs from examples within specific heuristics, starting with the early algorithms by B. Trakhtenbrot and Y. Barzdin [TB73] and Gold [Gol78]. Many other algorithms have been developed, most of them having the convergence based on characteristic sets: RPNI (Regular Positive and Negative Inference) by J. Oncina and P. García [OG91, OG92], Traxbar by K. Lang [Lan92], EDSM (Evidence Driven State Merging), Windowed EDSM and Blue-Fringe EDSM by K. Lang, B. Pearlmutter and R. Price [LPP98], SAGE (Self-Adaptive Greedy Estimate) by H. Juillé [JP98], etc.

The picture becomes brighter when we turn to active learning. Angluin [Ang87c], elaborating on an algorithm by Gold [Gol72], proves that if a learning algorithm can ask both MQs and EQs, then finite automata are learnable in polynomial time. Later, R.L. Rivest and R.E. Schapire [RS87, RS93], L. Hellerstein et al. [HRPW95, HPRW96] or J.L. Balcázar et al. [BDGW94] develop more efficient versions of the same algorithm trying to increase the parallelism level, to reduce the number of equivalence queries, etc. D. Angluin and M. Kriķis [AK94] show how to efficiently learn DFAs from MQs and EQs when the answers to the MQs are wrong on some subsets of the queries which may be arbitrary but of bounded size. M.J. Kearns and U.V. Vazirani's version [KV94] provides ample additional intuitions that are rather hard to grasp in previous works. The paper by J.L. Balcázar, J. Díaz and R. Gavaldà [BDGW97] presents some of the previous algorithms in a unified view. In our dissertation, we thoroughly investigate DFA learning with CQs and EQs, and we propose some subclasses of regular languages that can be learned efficiently with CQs alone.

The basic paradigm of asking questions has been applied to other formalisms such as: DNF formulas [Ang87b], nondeterministic finite automata [Yok94], two-tape automata [Yok96], context-free grammars [Ang87a, Sak90], deterministic one-counter automata [BR87], deterministic bottom up tree automata [Sak87a, DH07], deterministic skeletal automata [Sak87b], Prolog programs [Sha83], probabilistic automata [Tze89]. Valiant also considered the issue briefly [Val84]. For a nice summary of some of these results, see [Ang88] and [Ang89].

Among the above mentioned language classes, the one that attracted a lot of attention was the learnability of context-free grammars (CFGs). This problem deserves to be investigated for both practical and theoretical reasons. Regarding practical applications, the learnability of a CFG that produces a set of patterns [Fu74] constitutes an important issue for people working in pattern recognition. Also, the ability to infer (in fact to approximate since the problem of natural languages' context-freeness is subject to a long debate) CFGs for natural languages would enable a speech recognizer to adapt its internal grammar according to the particularities of an individual speaker [Hor69]. The problem seems to be interesting also from a theoretical point of view because, although context-free languages (CFLs) are well-understood, there are various and serious restrictions in the process of learning them. A very good survey of the problem of learning CFLs can be found in [Lee96].

The first attempt to extend the query learning method to CFGs belongs to Angluin [Ang87a]. She designs the algorithm L^{cf} and shows that, within specific heuristics, CFGs are learnable. Among them, we mention two important ones: the grammar should be in Chomsky normal form and the set of nonterminal symbols, along with the start symbol, are assumed to be known by the learner. Sakakibara extends the algorithm L^* to skeletal regular tree automata, which are a variation of finite automata that take skeletons as input [Sak90]. Intuitively, skeletons (introduced in [LJ78]) are derivation trees of strings in a grammar in which internal nodes are unlabeled. Such a tree reveals only the syntactic structure associated with a string with respect to a CFG but not the concrete rules generating it. F. Drewes and J. Högberg [DH03, DH07] improve Sakakibara's algorithm, with a generalization of L^* to regular tree languages (RTLs) that avoids dead states both in the resulting automaton and the learning phase (which also leads to a considerable improvement with respect to efficiency).

The interest in learning RTLs [FK84, Sak90, Sak92, COCR98] is justified by the fact that they provide a well-known generalization of regular string languages to which nearly all the classical results carry over (see, e.g., [GS84]). Moreover, their yields are exactly the context-free string languages. Therefore, in this dissertation we drop the restriction to skeletal trees and we investigate the learnability of RTLs with structural correction queries (SCQs) and EQs.

The thesis is organized as follows. Chapter 2 provides notions and notations that are used throughout this monograph. In particular, we give definitions of the models of learning in the limit, finite learning and query learning, and we familiarize the reader with the standard L^* algorithm for learning DFAs with MQs and EQs.

In Chapter 3 we focus on the learning power of each of the three types of CQs introduced so far when no efficiency constraints are imposed. In the case of PCQs, we first provide the reader with necessary and sufficient conditions for an indexed family of nonempty recursive languages to be inferable with PCQs alone, and then we use these conditions to compare the model with other standard learning models: (conservative) learning in the limit from text, finite learning from text or informant, query learning with MQs or EQs. For the other two types of CQs, the proofs are done the other way around. We first show that learning with LBCQs and learning with EDCQs is basically the same as learning with MQs. Then, we use the fact that learning with MQs is nothing more than finite learning from informant [LZ04b] and Mukouchi's characterization of the later model [Muk92a] to characterize our models in terms of finite sets (pairs of definite finite tell-tales, to be precise). An intuitive picture displaying all these models and the relations between them is presented in the last section of the chapter.

Our results show that PCQs are more powerful than MQs, that learning with PCQs is strictly weaker than learning in the limit from text, and that learning with PCQs and conservative learning from text in the limit are incomparable. Actually, any indexable class of recursive languages that is learnable with PCQs but not conservatively inferable from text in the limit must contain at least one language with a non-recursive prefix. Therefore, if an indexable class of recursive languages that is learnable with PCQs has only recursive prefixes, then it is also conservatively learnable from text in the limit.

The results obtained for PCQs are based on the characterization of the model in terms of finite tell-tales. Let us recall the basic difference between a finite and a definite finite tell-tale. For the former one, each language in the class should be the **smallest** one consistent with its tell-tale, whereas for the later one, each language in the class should be the **only** one consistent with its tell-tale. Furthermore, if a definite finite tell-tale is a set containing strings in the language, a pair of definite finite tell-tales has two sets: one with positive and one with negative examples. It turns out that positive and negative examples are not enough to characterize the model of learning with PCQs, so a third dimension is required (the third finite set will contain strings that are not a prefix of any other string in the language). We introduce the notion of *triples of definite finite tell-tales*, and we show that having a way to compute such a triple for all languages in the class is a sufficient condition for the inferrability of the given class with PCQs.

In Chapter 4 we focus on polynomial time algorithms to further understand the model of learning with CQs. Again, the chapter is divided into four sections, each of the first three dealing with one type of CQ, and the last one presenting the global picture.

In the first section of Chapter 4 we give polynomial time algorithms for learning the class of pattern languages, and the class of k-reversible languages, respectively, with PCQs. Moreover, we show that pattern languages are not efficiently learnable with MQs, and that there is no algorithm that identifies k-reversible languages with a finite number of MQs(even if we allow exponentially many MQs). All these results facilitate a comparison between PCQs and MQs when efficiency constraints are imposed.

In the second and third section of Chapter 4 we show that learning with LBCQs is not more powerful than learning with MQs even when we restrict to polynomial time algorithms, and that the EDCQs and MQs do no longer share the same learning power when we talk about efficient algorithms. Moreover, we could prove that there are language classes polynomial time learnable with PCQs for which there is no EDCQ efficient algorithm. Even if intuitively, there should be situations where EDCQs are more powerful than PCQs in the restricted case of polynomial time learnability, we could not find such a class, so we leave this question open.

In Chapter 5 we switch to the problem of learning DFAs, one of the earliest in grammatical inference, being also the purpose for introducing the model of learning from queries (regular languages were known not to be learnable in the identification in the limit model proposed by Gold). We investigate DFA learnability with CQs and EQs - note that they are not learnable with CQs alone since no superfinite class is learnable from text in the limit [Gol67], and the model of learning with CQs is weaker. Moreover, we show how is the efficiency of the algorithm L* affected when replacing MQs with PCQs (and with LBCQs, respectively). Furthermore, we investigate two subclasses for which EQs are not needed: injective languages and singleton languages. The reader may notice that we have completely ignored the third type of CQ, namely the EDCQ. It is because EDCQs are not helping in this case. A detailed discussion is presented in Section 5.3.

In Chapter 6 we extend the notion of CQs to trees, thus defining Structural Correction Queries (SCQs), and we provide an algorithm that learns any deterministic bottom-up tree recognizer with SCQs and EQs in polynomial time (in the size of the finite state machine and the size of the biggest counterexample returned by the teacher). We also extend the results from the string case, showing that there are subclasses of RTLs learnable with SCQs alone. The last result has relevance in practice since one can imagine the teacher as a human expert who might not have an automaton-like representation for the language but is still able to return the correction based only on his domain-specific knowledge.

The order we choose for comparing trees (contexts) to obtain the minimal one is a Knuth-Bendix order based on the weights associated to the symbols of the alphabet. The minimal context may correspond to the most probable choice in a real-life setting, like speech recognition or even machine translation if we imagine an extension of our algorithm to transducers.

Results in this monograph have been previously published in [BeBiDe05, BeDeTî06, TîKo07, TîKn07a, TîKn07b, TîTî07, Tîr08a, TîKo09, MiTî08].

Chapter 2

Preliminaries

We follow standard definitions and notations in formal language theory. The reader is referred to [HU79, HU90, MVMP04] for further information about this domain.

If ϕ is a function from X to Y then $\phi(X') = \{\phi(x) \mid x \in X'\}$ for any $X' \subseteq X$. By $|X|$ we denote the *cardinality* of the finite set X and by $\mathcal{P}(X)$ its *power set* (i.e., the set of all subsets of X). The function ϕ is *injective* if for all $x_1, x_2 \in X, f(x_1) = f(x_2)$ implies $x_1 = x_2$, and *surjective* if for all $y \in Y$ there exists $x \in X$ such that $\phi(x) = y$. Then ϕ is *bijective* if and only if it is injective and surjective.

A partition of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets. More precisely, a set ρ of nonempty sets is a partition of X if

- the union of the elements of ρ is equal to X, and
- the intersection of any two elements of ρ is empty.

The elements of ρ are sometimes called *blocks* or *parts* of the partition.

Given two partitions ρ_1 and ρ_2 of a given set X, we say that ρ_1 is *finer* than ρ_2, (or that ρ_2 is *coarser* than ρ_1), if ρ_1 splits the set X into smaller blocks than ρ_2 does, i.e., every element of ρ_1 is a subset of some element of ρ_2.

2.1 Strings, String Languages and Automata

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 *strings* or *words*. Let u, v, w be strings in Σ^* and $|w|$ be the length w. λ is a special word called the *empty* string and has length 0. We denote by uv

or $u \cdot v$ the concatenation of the strings u and v. If $w = uv$ for some u, v in Σ^*, then u is a *prefix* of w and v is a *suffix* of w. A set L is called *prefix-closed* (*suffix-closed*) if for any w in L, all prefixes (suffixes) of w are also in L.

By $\mathit{Pref}(L)$ we denote the set $\{u \in \Sigma^* \mid \exists v \in \Sigma^* \text{ such that } uv \in L\}$ and by $\mathit{Tail}_L(u)$ the set $\{v \mid uv \in L\}$. Then, by $\Sigma^{\leq k}$, Σ^k and $\Sigma^{\geq k}$ we denote the sets $\{u \in \Sigma^* \mid |u| \leq k\}$, $\{u \in \Sigma^* \mid |u| = k\}$ and $\{u \in \Sigma^* \mid |u| \geq k\}$, respectively.

Assume that Σ is a totally ordered set and let $\prec_l$ 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 $u \prec_l v$. In other words, strings are compared first according to length and then lexicographically.

2.1.1 Distances on Strings

There are several ways to determine how similar two strings are (a fairly complete overview on string metrics can be found in [Cha]). One of the string distances most used in practice (e.g., by spell checkers) is the *edit distance* (also called Levenshtein distance [Lev66]). Informally, the edit distance between the strings w and w', denoted $d(w, w')$ in the sequel, is the minimum number of *edit operations* needed to transform w into w'. The edit operations are either (1) *deletion*: $w = uav$ and $w' = uv$, or (2) *insertion*: $w = uv$ and $w' = uav$, or (3) *substitution*: $w = uav$ and $w' = ubv$, where $u, v \in \Sigma^*, a, b \in \Sigma$ and $a \neq b$. The above mentioned distance can be considered a generalization of the Hamming distance [Ham50], which is used for strings of the same length and only considers substitution edits. There are also further generalizations of the Levenshtein distance that consider, for example, exchanging two characters as an operation (called *transposition*), like in the Damerau-Levenshtein distance algorithm [Dam64].

Example 1. Let us consider the strings *bbab* and *baa*. There are several ways of transforming *bbab* into *baa* by using the three above mentioned operations. For example, $\underline{b}bab \rightarrow ba\underline{b} \rightarrow baa$ or $bba\underline{b} \rightarrow b\underline{b}a \rightarrow baa$, but there is no way of obtaining *baa* from *bbab* with less than 2 operations. Therefore, the distance $d(bbab, baa)$ is 2.

Notice that for longer strings it is not as easy as in the example to find, by hand, the shortest number of operations needed. Nevertheless, the edit distance between two strings w and w' can be computed in $\mathcal{O}(|w| \cdot |w'|)$ time by dynamic programming [WF74].

2.1.2 Relations on Strings

Given a language L, one can define the following *equivalence relation* (i.e., a relation which is reflexive, symmetric and transitive) on strings: $u \equiv_L v$ if and only if $(\forall w \in \Sigma^*, u \cdot w \in L \Leftrightarrow v \cdot w \in L)$. This equivalence relation divides the set of all strings into one or more *equivalence classes*. For any language L over Σ and any w in Σ^* we denote by $[w]_L$ the equivalence class of w with respect to the language L, or simply $[w]$ when L is understood from the context. The number of equivalence classes induced by $\equiv_L$ is called the *index* of L.

2.1.3 Finite Automata

A *deterministic finite automaton* (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ 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 strings by doing $\delta(q, \lambda) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$ whenever the right-hand side is defined. The number of states in Q gives the *size* of $\mathcal{A}$. A string w is accepted by $\mathcal{A}$ if $\delta(q_0, w) \in F$. The set of strings accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$ and is called a *regular language*.

We say that 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 complete DFA hereinafter denoted $\mathcal{A}_L$ such that $\mathcal{L}(\mathcal{A}_L) = L$. The Myhill-Nerode Theorem [Myh57, Ner58] states that the size of $\mathcal{A}_L$ equals the index of L. An immediate consequence of this theorem is that a language L is regular if and only if L has finite index.

A *nondeterministic finite automaton* (NFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q, Σ, q_0 and F are defined exactly the same way as for a DFA and the transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ takes values in $\mathcal{P}(Q)$ instead of in Q.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA. A state q is called *reachable* if there exists a string u in Σ^* such that $\delta(q_0, u) = q$ and *co-reachable* if there exists a string v such that $\delta(q, v) \in F$. A state which is reachable but not co-reachable is also called a *sink state*. Note that for a minimal DFA $\mathcal{A}$, there is at most one sink state and all the states are reachable.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ be two DFAs and ϕ be a function from Q to Q'. Then ϕ is an *automata morphism* if $\phi(q_0) = q_0'$, $\phi(F) \subseteq F'$ and for all $q \in Q$, $a \in \Sigma$, $\phi(\delta(q, a)) = \delta'(\phi(q), a)$. Moreover, if ϕ is a bijective function and $\phi(F) = F'$ then ϕ is an *automata isomorphism*. The two automata $\mathcal{A}$ and $\mathcal{A}'$ are said to be *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and *isomorphic* if there exists an automata isomorphism $\phi : Q \to Q'$.

The *reverse* of an automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \{q_f\})$ is the automaton $\mathcal{A}^r = (Q, \Sigma, \delta^r, q_f, \{q_0\})$ where $\delta^r(q, a) = \{q' \mid q \in \delta(q', a)\}$, for all $a \in \Sigma$ and $q \in Q$.

Pictorially, we obtain $\mathcal{A}^r$ from $\mathcal{A}$ by interchanging the initial and final states and reversing each of the transition arrows. Then, an automaton $\mathcal{A}$ is said to be *0-reversible* if and only if both $\mathcal{A}$ and $\mathcal{A}^r$ are deterministic. Note that if $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is a 0-reversible automaton which accepts both u_1v and u_2v, then $\delta(q_0, u_1) = \delta(q_0, u_2)$.

Let k be a fixed nonnegative integer and $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ an automaton. The string u is said to be a *k-follower* (or *k-leader*) of the state q in $\mathcal{A}$ if $|u| = k$ and $\delta(q, u) \neq \emptyset$ ($\delta^r(q, u^r) \neq \emptyset$ respectively). Note that every state has exactly one 0-follower and one 0-leader, namely, λ. The automaton $\mathcal{A}$ is defined to be *deterministic with lookahead k* if for any pair of distinct states q_1 and q_2 such that $q_1, q_2 \in \delta(q_3, a)$ for some $q_3 \in Q$ and $a \in \Sigma$, there is no string that is a k-follower of both q_1 and q_2. This guarantees that any nondeterministic choice in the operation of $\mathcal{A}$ can be resolved by looking ahead k symbols past the current one. An automaton $\mathcal{A}$ is said to be *k-reversible* if $\mathcal{A}$ is deterministic and $\mathcal{A}^r$ is deterministic with lookahead k. Note that this definition coincide with the definition for 0-reversible automaton when $k = 0$.

2.1.4 String Languages

Let us first introduce some particular classes of languages which will be later used in this monograph.

- Following [BBdlHJT07], one can define the language $B_r(w)$ of all strings whose edit distance is at most r from w where $w \in \Sigma^*$ and $r \in \mathbb{R}$. This language will subsequently be called the *ball of center w and radius r*. Formally, $B_r(w) = \{v \in \Sigma^* \mid d(v, w) \leq r\}$. We denote by $\mathcal{B}$ the class of all balls of strings over a fixed alphabet Σ.

  Example 2. If we take $\Sigma = \{a, b\}$, then $B_k(\lambda) = \Sigma^{\leq k}$ for all $k \in \mathbb{N}$, and $B_1(aba) = \{aa, ab, ba, aaa, aba, abb, bba, aaba, abaa, abab, abba, baba\}$.

- Given a nonnegative integer k, a language L is said to be *k-reversible* if there exists a k-reversible automaton $\mathcal{A}$ such that $L = \mathcal{L}(\mathcal{A})$. We denote by *$k$-Rev* the class of all reversible languages. We have:

  Proposition 1 (Angluin [Ang82])**.** *A regular language L is in k-Rev if and only if whenever u_1vw, u_2vw are in L and $|v| = k$, $Tail_L(u_1v) = Tail_L(u_2v)$.*

  As a direct consequence, we get:

  Corollary 1 (Angluin [Ang82])**.** *A regular language L is 0-reversible if and only if whenever u_1w, u_2w are in L, $Tail_L(u_1) = Tail_L(u_2)$ (and hence, $u_1 \equiv_L u_2$).*

Example 3. Let $\Sigma = \{a, b\}$ and $L = \{ab^2a, b^3a, b^4\}$. Because the strings $a \cdot b^2 \cdot a$ and $b \cdot b^2 \cdot a$ are in L, $|b^2| = 2$ and $Tail_L(a \cdot b^2) = \{a\} \neq \{a, b\} = Tail_L(b \cdot b^2)$, the language L is not in *2-Rev* (by Proposition 1). On the other hand, the only strings in L having a common suffix of length at least 3 are $a \cdot b^2a$ and $b \cdot b^2a$. But $Tail_L(a \cdot b^2a) = \{\lambda\} = Tail_L(b \cdot b^2a)$. Hence, L is 3-reversible and not 2-reversible.

- 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 nonempty string over $\Sigma \cup X$. The *pattern language* $L(\pi)$ consists of all the strings obtained by uniformly replacing the variables in π with arbitrary strings in Σ^+. Let us denote by $\mathbb{P}$ the set of all pattern languages over a fixed alphabet Σ. The pattern π is in *normal form* if the variables occurring in π are precisely $x_1, \dots, x_k$, and for every i with $1 \leq i < k$, the leftmost occurrence of x_i in π is left to the leftmost occurrence of x_{i+1}.

  Example 4. Let $\Sigma = \{a, b\}$ and $\pi_1 = x, \pi_2 = xx, \pi_3 = axyb$. Then, $L(\pi_1) = L(x) = \Sigma^+$, $L(\pi_2) = L(xx) = \{ww \mid w \in \Sigma^+\}$, and $L(\pi_3) = L(axyb) =$ the set of all words of length at least 4 starting with a and ending with b.

- Given a fixed alphabet Σ, we denote by $\mathcal{S}$ the class of all singleton languages $L_w = \{w\}$ over Σ, and by $\mathcal{S}_k$ the class $(L_w)_{w \in \Sigma^k}$.

We assume the reader is familiar with basic notions of formal language theory. In the sequel we briefly recall some definitions, closure properties and decidability issues. The Chomsky hierarchy [Cho56] consists of:

- *Type-0 grammars* include all formal grammars. They generate exactly all languages for which there exists a Turing machine recognizing them (also known as *recursively enumerable* (RE) languages). A proper subclass is the class of *recursive languages* that consists of all languages for which there exists an always-halting Turing machine recognizing them.
- *Type-1 grammars* (context-sensitive grammars) generate the class of all context-sensitive languages (CSLs).
- *Type-2 grammars* (context-free grammars) generate the class of context-free languages (CFLs). They are the theoretical basis for the syntax of most programming languages, and are also considered to be the best ones, in the Chomsky hierarchy, for describing the syntax of natural languages.
- *Type-3 grammars* (regular grammars) generate the class of all regular languages.

We denote by *RE*, *CS*, *CF* and *Reg* the family of languages generated by arbitrary, context-sensitive, context-free, and regular grammars, respectively. By *Rec* we denote the family of recursive languages.

A *decision problem* is a question in some formal system with a `Yes`/`No` answer, depending on the values of some input parameters. A decision problem is *(algorithmically/recursively) decidable* if there exists an algorithm such that, given any instance of the problem as input, outputs `Yes` or `No`, provided that the input is true or not, respectively [MVMP04]. Here is a list with the most common decidability issues.

- *Emptiness*: is a given language empty?
- *Finiteness*: is a given language a finite set?
- *Membership*: does $w \in L$ hold for a given string w and a language L?
- *Inclusion*: does $L_1 \subseteq L_2$ hold for two given languages L_1 and L_2?
- *Equivalence*: does $L_1 = L_2$ hold for two given languages L_1 and L_2?

Let us detail the status of these decision problems for some of the above mentioned grammars (U stands for undecidable, and D for decidable).

Table 2.1: Decidability results

	RE	*Rec*	*CF*	*Reg*	$\mathbb{P}$	$\mathcal{B}$
Emptiness	U	U	D	D	D	D
Finiteness	U	U	D	D	D	D
Membership	U	D	D	D	D	D
Inclusion	U	U	U	D	U	D
Equivalence	U	U	U	D	D	D

The next table presents some closure properties of these classes of languages, among which we emphasize that the intersection of two recursive languages is recursive, and that *Rec* is closed under union and intersection with regular sets (i.e., if L_1, L_2 are arbitrary languages in *Rec* and L is a regular language, then $L_1 \cap L_2$, $L_1 \cup L_2$ and $L_1 \cap L$ are also in *Rec*).

Table 2.2: Closure properties

	RE	*Rec*	*CF*	*Reg*
Union	Yes	Yes	Yes	Yes
Intersection	Yes	Yes	No	Yes
Intersection with regular languages	Yes	Yes	Yes	Yes

Finally, the last class of string languages we are going to present in this introduction is the class of LL(1) *linear context-free languages*. An LL(1) parser is a top-down parser for a subset of the context-free grammars. It parses the input from left to right, and constructs a leftmost derivation of the sentence using one token of look-ahead. The class of grammars which are parsable in this way is known as the class of LL(1) grammars. We are not going into further details, since the only property of this class that we use is that LL(1) linear CFLs are all context-free, and hence recursive.

2.2 Trees, Tree Languages and Tree Recognizers

A wealth of information about trees, tree languages and tree recognizers can be found in [CDG$^+$07, GS84]. The trees considered here are finite, their nodes are labeled by symbols, and the branches leaving any given node have a specified order.

A *ranked alphabet* Δ is a finite set of symbols each of them having a given non-negative integer arity. For any $m \geq 0$, the set of m-ary symbols in Δ is denoted by Δ_m. In examples we may write $\Delta = \{f_1/m_1, \ldots, f_k/m_k\}$ to indicate that Δ consists of the symbols $f_1, \ldots, f_k$ with the respective arities $m_1, \ldots, m_k$. In what follows Δ is always a ranked alphabet.

In addition to a ranked alphabet, an ordinary finite alphabet, called a *leaf alphabet* and usually denoted by X, is used for labeling leaves of trees. Leaf alphabets are assumed to be disjoint from the ranked alphabets considered.

The set $\mathbb{T}_\Delta(X)$ of Δ-*terms with variables in* X is the smallest set T such that

- $\Delta_0 \cup X \subseteq T$, and
- $f(t_1, ..., t_m) \in T$ whenever $m > 0$, $f \in \Delta_m$ and $t_1, ..., t_m \in T$.

Such terms are regarded as representations of trees, and we call them ΔX-*trees*. Any $f \in \Delta_0 \cup X$ represents a tree with only one node labeled with f. Similarly, $f(t_1, ..., t_m)$ is interpreted as a tree formed by adjoining the m trees represented by $t_1, ..., t_m$ to a new f labeled root. The trees $t_1, ..., t_m$ are said to be the *direct subtrees* of the tree. Subsets of $\mathbb{T}_\Delta(X)$ are called ΔX-*tree languages*. We will generally speak about *trees* and *tree languages* without specifying the alphabets.

Given a set T of ΔX-trees, we denote by $T\Delta$ the set of all trees $f(t_1, ..., t_m)$ such that $f \in \Delta_m$ for some $m \geq 0$, and $t_1, \ldots, t_m \in T$.

The *height* $hg(t)$ and the set of *subtrees* $sub(t)$ of a ΔX-tree t are defined such that

- $hg(t) = 0$, $sub(t) = \{t\}$ for $t = f \in \Delta_0 \cup X$, and

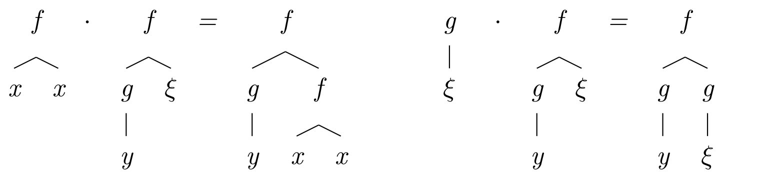
- $hg(t) = \max\{hg(t_1), \ldots, hg(t_m)\} + 1$, $sub(t) = \{t\} \cup sub(t_1) \cup \ldots \cup sub(t_m)$ for $t = f(t_1, \ldots, t_m)$, $m > 0$ and $f \in \Delta_m$.

Let ξ be a special symbol with arity 0 such that $\xi \notin \Delta \cup X$. A $\Delta(X \cup \{\xi\})$-tree in which ξ appears exactly once is called a *ΔX-context*, or just a *context*. The set of all ΔX-contexts is denoted by $\mathbb{C}_\Delta(X)$. If $c, c' \in \mathbb{C}_\Delta(X)$, then $c \cdot c' = c'(c)$ is the ΔX-context, obtained from c' by replacing the ξ in it with c. Similarly, if $t \in \mathbb{T}_\Delta(X)$ and $c \in \mathbb{C}_\Delta(X)$, then $t \cdot c = c(t)$ is the tree obtained when the ξ in c is replaced with t (we also say that the context c is *applied* to t).

Let c be a context in $C_\Delta(X)$. Then $depth(c)$ and $trees(c)$ are defined as follows:

- $depth(c) = 0$, $trees(c) = \emptyset$ for $c = \xi$, and
- $depth(c) = depth(c') + 1$, $trees(c) = trees(c') \cup \{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}$ for $c = f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot c'$ with $m > 0$, $f \in \Delta_m$, $t_1, \ldots, t_{i-1}$, $t_{i+1}, \ldots, t_m \in T_\Delta(X)$, $1 \leq i \leq m$, and $c' \in \mathbb{C}_\Delta(X)$.

Example 5. If $\Delta = \{f/2, g/1\}$ and $X = \{x, y\}$, then $f(x,x)$, $f(g(y), f(x,x))$ are ΔX-trees, and $g(\xi)$, $f(g(y), \xi)$, $f(g(y), g(\xi))$ are ΔX-contexts.



tree · context = tree   context · context = context

We have:

$\mathrm{hg}(f(g(y), f(x,x))) = 2$, $\quad g(\xi) \cdot f(g(y), \xi) = f(g(y), g(\xi))$,
$\mathrm{sub}(f(x,x)) = \{x, f(x,x)\}$, $\quad f(x,x) \cdot f(g(y), \xi) = f(g(y), f(x,x))$.

For a tree $t \in \mathbb{T}_\Delta(X)$, the *frontier derivative language of T with respect to t* is the set $Fron_T(t) = \{c \in \mathbb{C}_\Delta(X) \mid t \cdot c \in T\}$.

A $\mathbb{T}_\Delta(X)$-*substitution*, or simply substitution, if the set of terms is irrelevant or clear from the context - is a function $\phi : X \to \mathbb{T}_\Delta(X)$. The set of all $\mathbb{T}_\Delta(X)$-substitutions will be denoted by $\mathcal{S}ub(\mathbb{T}_\Delta(X))$ or simply $\mathcal{S}ub$. Any $\mathbb{T}_\Delta(X)$-substitution ϕ can be extended to a mapping $\hat{\phi} : \mathbb{T}_\Delta(X) \to \mathbb{T}_\Delta(X)$ as follows:

- $\hat{\phi}(f) = f$ for all $f \in \Delta_0 \cup X$, and
- $\hat{\phi}(f(t_1, \ldots, t_m)) = f(\hat{\phi}(t_1), \ldots, \hat{\phi}(t_m))$ for $m > 0$, f in Δ_m and $t_1, \ldots, t_m$ in $\mathbb{T}_\Delta(X)$.

2.2.1 Finite Tree Recognizers, Regular Tree Languages

A *deterministic bottom-up ΔX-tree recognizer*, or tree recognizer, for short, is a quintuple $\mathcal{R} = (Q, \Delta, X, \delta, F)$, where Δ is a ranked alphabet, X is a leaf alphabet, Q is a finite set of *states* such that $Q \cap \Delta = \emptyset$, $F \subseteq Q$ is a set of *final states*, and the *transition function* δ is a family of maps δ_m $(m \geq 0, \Delta_m \neq \emptyset)$ such that:

- $\delta_0 : \Delta_0 \cup X \to Q$, and
- $\delta_m : Q^m \times \Delta_m \to Q$ for $m > 0$.

Then for $t \in \mathbb{T}_\Delta(X)$, one can inductively define $\delta(t)$ as follows:

- $\delta(t) = \delta_0(t)$ for $t \in \Delta_0 \cup X$, and
- $\delta(t) = \delta_m(\delta(t_1), \ldots, \delta(t_m), f)$ for $m > 0$, $f \in \Delta_m$ and $t = f(t_1, \ldots, t_m)$.

The set $\mathcal{T}(\mathcal{R}) = \{t \in \mathbb{T}_\Delta(X) \mid \delta(t) \in F\}$ is the tree language *recognized* (*accepted*) by $\mathcal{R}$. Such a tree language is a *regular tree language*. Let $\mathcal{R} = (Q, \Delta, X, \delta, F)$ and $\mathcal{R}' = (Q', \Delta, X, \delta', F')$ be deterministic bottom-up ΔX-tree recognizers. One can say that $\mathcal{R}$ and $\mathcal{R}'$ are *equivalent* if they accept the same tree language, and $\mathcal{R}$ is *isomorphic* to $\mathcal{R}'$ if there exists a bijection $\phi : Q \to Q'$ such that:

- $\phi(F) = F'$,
- $\phi(\delta_0(f)) = \delta'_0(f)$ for every $f \in \Delta_0 \cup X$, and
- $\phi(\delta_m(q_1, \ldots, q_m, f)) = \delta'_m(\phi(q_1), \ldots, \phi(q_m), f)$ for every $m > 0$, $f \in \Delta_m$ and every $q_1, \ldots, q_m \in Q$.

Let $\mathcal{R}$ be a tree recognizer. A state q is called *reachable* if there exists a tree $t \in \mathbb{T}_\Delta(X)$ such that $\delta(t) = q$. A reachable state q is *co-reachable* if there exists a context $c \in \mathbb{C}_\Delta(X)$ such that $\delta(t \cdot c) \in F$, where t is any tree in $\mathbb{T}_\Delta(X)$ such that $\delta(t) = q$. A state which is reachable but not co-reachable is said to be a *sink state*.

Lemma 1 (Replacement lemma [Sak90]). *Let $\mathcal{R} = (Q, \Delta, X, \delta, F)$ be a deterministic bottom-up ΔX-tree recognizer. For $t, t' \in \mathbb{T}_\Delta(X)$ and $c \in \mathbb{C}_\Delta(X)$, if $\delta(t) = \delta(t')$, then $\delta(t \cdot c) = \delta(t' \cdot c)$.*

For a given tree language $T \subseteq \mathbb{T}_\Delta(X)$, the equivalence relation $\cong_T$ on $\mathbb{T}_\Delta(X)$ is defined by: for any $t, t' \in \mathbb{T}_\Delta(X)$, $t \cong_T t'$ if for all contexts $c \in \mathbb{C}_\Delta(X)$, $t \cdot c \in T \Leftrightarrow t' \cdot c \in T$. One can notice that $\cong_T$ is a congruence (i.e. an equivalence relation which is preserved by contexts: for $t, t' \in \mathbb{T}_\Delta(X)$, if $t \cong_T t'$, then $t \cdot c \cong_T t' \cdot c$ for all contexts $c \in \mathbb{C}_\Delta(X)$).

It is known that each regular tree language T is accepted by a *minimal* tree recognizer (which we denote by $\mathcal{R}_T$), unique up to isomorphism, that can be constructed from any given tree recognizer $\mathcal{R}$ of T by first deleting all non-reachable states and then merging all pairs of equivalent states. For details, we refer to [GS84], pp. 87 - 94. Note that for a minimal tree recognizer $\mathcal{R}$ there is at most one sink state and all the states are reachable.

Clearly, $Fron_T(t) = \{c \in \mathbb{C}_\Delta(X) \mid \delta(t \cdot c) \in F\}$, where $\mathcal{R} = (Q, \Delta, X, \delta, F)$ is any deterministic bottom-up ΔX-tree recognizer accepting the tree language T. One can speak about the frontier derivative language of a state q as being $Fron_T(t_q)$, where $t_q \in \mathbb{T}_\Delta(X)$ is such that $\delta(t_q) = q$.

Example 6. Let $\Delta = \{f/3, g/2, a/0, b/0\}$, $X = \emptyset$, and consider the tree language

$$T = \{t \in \mathbb{T}_\Delta(\emptyset) \mid t = g(a,b) \cdot [f(a,\xi,b)]^n, n \geq 0\},$$

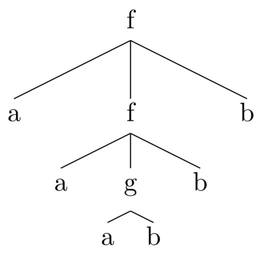
where $[f(a,\xi,b)]^0 = \xi$ and $[f(a,\xi,b)]^n = [f(a,\xi,b)]^{n-1} \cdot f(a,\xi,b)$, $n \geq 1$.

Let us now construct a tree recognizer $\mathcal{R} = (Q, \Delta, \emptyset, \delta, F)$ such that $\mathcal{T}(\mathcal{R}) = T$. For this, we need a set Q of four states: q_a, q_b, q_s and q_f. The transition function δ is defined as follows:

- $\delta_0(a) = q_a$, $\delta_0(b) = q_b$,
- $\delta_2(q_a, q_b, g) = q_f$,
- $\delta_2(q_a, q_a, g) = \delta_2(q_b, q_a, g) = \delta_2(q_b, q_b, g) = q_s$,
- $\delta_3(q_a, q_f, q_b, f) = q_f$, and
- $\delta_3(\cdot, \cdot, \cdot, f) = q_s$ for all remaining cases.

If we take $F = \{q_f\}$, it is clear that $\mathcal{T}(\mathcal{R}) = T$. The (accepting) computation on the input $t = f(a, f(a, g(a,b), b), b)$ is

$$\begin{aligned}\delta(t) &= \delta_3(q_a, \delta_3(q_a, \delta_2(q_a, q_b, g), q_b, f), q_b, f) \\ &= \delta_3(q_a, \delta_3(q_a, q_f, q_b, f), q_b, f) \\ &= \delta_3(q_a, q_f, q_b, f) \\ &= q_f.\end{aligned}$$



Also, it is easy to see that q_s is a sink state and q_f, q_a and q_b are co-reachable.

2.2.2 Orders on Trees

As in the case of strings, one may imagine various ways of defining orders on trees. In this section we briefly present one method of constructing such relations on trees (namely the simplification order) that yields classes of orders (e.g., Knuth-Bendix orders) that can be used in fully automated termination proofs. More details, examples and results can be found in [BN98], pp. 93-133.

Let Δ be a ranked alphabet and X a leaf alphabet. A strict order $<$ on $\mathbb{T}_\Delta(X)$ is called a *rewrite order* if it is:

1. *compatible* with Δ-operations: for all $t, t' \in \mathbb{T}_\Delta(X)$, all $m \geq 0$, and all $f \in \Delta_m$, $t < t'$ implies

$$f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_m) < f(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_m)$$

   for all $i \in \{1, \ldots, m\}$, and all $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m \in \mathbb{T}_\Delta(X)$.

2. *closed under substitutions*: for all $t, t' \in \mathbb{T}_\Delta(X)$ and all substitutions $\sigma \in \mathcal{S}ub(\mathbb{T}_\Delta(X))$, $t < t'$ implies $\sigma(t) < \sigma(t')$.

Note that the first condition implies (and thus is equivalent to) the following condition:

1′. for all $t, t' \in \mathbb{T}_\Delta(X)$ and all $c \in \mathbb{C}_\Delta(X), t < t'$ implies $t \cdot c < t' \cdot c$.

A strict order $<$ on $\mathbb{T}_\Delta(X)$ is called a *simplification order* if it is a rewrite order and satisfies the following *subterm property*: for all trees $t \in \mathbb{T}_\Delta(X)$ and all proper subtrees $t' \in sub(t) \backslash \{t\}$ of t, we have $t' < t$.

There are several methods for constructing specific simplification orders: Knuth-Bendix orders, polynomial simplification orders, recursive path orders, etc. The former one makes a lexicographic comparison, where first the weights of the trees are compared, second their root symbols, and third recursively the collections of direct subtrees. The difference between the Knuth-Bendix order and the lexicographic path order in the case of trees is analogue to the difference between the lex-length order and the lexicographical order for strings, as one can see from Example 7.

Knuth-Bendix Orders

Let Δ be a ranked alphabet, X a leaf alphabet, and $<$ a strict order on Δ. Let $\omega : \Delta \cup X \to \mathbb{R}_0^+$ be a *weight function*, where $\mathbb{R}_0^+$ denotes the set of nonnegative real numbers. A weight function ω is called *admissible* for $<$ if it satisfies the following conditions:

1. There exists $r_0 \in \mathbb{R}_0^+ \backslash \{0\}$ such that $\omega(x) = r_0$ for all variables $x \in X$, and $\omega(f) \geq r_0$ for all $f \in \Delta_0$.

2. If $g \in \Delta_1$ and $\omega(g) = 0$, then g is the greatest element in Δ, i.e. $f \leq g$ for all $f \in \Delta$.

The weight function ω is extended to a function $\omega : \mathbb{T}_\Delta(X) \to \mathbb{R}_0^+$ as follows:

$$\omega(t) = \sum_{x \in X} \omega(x) \cdot |t|_x + \sum_{f \in \Delta} \omega(f) \cdot |t|_f,$$

where $|t|_x$ ($|t|_f$) denotes the number of occurrences of the variable x (symbol f) in t. Thus, $\omega(t)$ simply adds up the weights of all occurrences of symbols in t.

A *Knuth-Bendix order* $<_{kbo}$ on $\mathbb{T}_\Delta(X)$ induced by $<$ and ω is defined as follows. For $t, t' \in \mathbb{T}_\Delta(X)$, we have $t <_{kbo} t'$ if:

1. $\omega(t) < \omega(t')$ and $|t|_x \leq |t'|_x$ for all $x \in X$, or

2. $\omega(t) = \omega(t')$, $|t|_x \leq |t'|_x$ for all $x \in X$, and one of the following properties holds:

   (a) There exist a unary symbol f, a variable x, and a positive integer n such that $t = x$ and $t' = f^n(x)$.

   (b) There exist symbols $f \in \Delta_m$ ($m \geq 0$) and $g \in \Delta_n$ ($n \geq 0$) such that $f < g$ and $t = f(t_1, \ldots, t_m)$, $t' = g(t'_1, \ldots, t'_n)$.

   (c) There exist a symbol $f \in \Delta_m$ ($m > 0$) and an index i, $1 \leq i \leq m$, such that $t = f(t_1, \ldots, t_i, \ldots, t_m)$, $t' = f(t'_1, \ldots, t'_i, \ldots, t'_m)$, $t_1 = t'_1, \ldots, t_{i-1} = t'_{i-1}$ and $t_i <_{kbo} t'_i$.

Remark 1. *The following properties hold:*

1. *ξ is the smallest context.*

2. *For any c_1, c_2 in $\mathbb{C}_\Delta(X)$, if $c_1 <_{kbo} c_2$, then $c_1 \cdot c <_{kbo} c_2 \cdot c$ and $c \cdot c_1 <_{kbo} c \cdot c_2$ for every c in $\mathbb{C}_\Delta(X)$.*

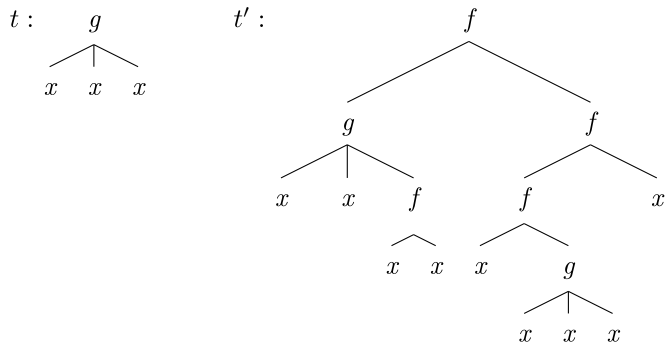
3. *For any t_1, t_2 in $\mathbb{T}_\Delta(X)$, if $t_1 <_{kbo} t_2$, then $t_1 \cdot c <_{kbo} t_2 \cdot c$ for every c in $\mathbb{C}_\Delta(X)$.*

4. *For any t in $\mathbb{T}_\Delta(X)$ and any $c \neq \xi$ in $\mathbb{C}_\Delta(X)$, $t <_{kbo} t \cdot c$.*

5. *For any c, c' in $\mathbb{C}_\Delta(X)$ with $c' \neq \xi$, $c <_{kbo} c' \cdot c$ (a direct consequence of the first two properties).*

Notice that the above mentioned properties hold for any simplification order, but the advantage of using a Knuth-Bendix order consists in the fact that $t <_{kbo} t'$ can be decided in time polynomial in the size of t and t'.

Example 7. Let $\Delta = \{f/2, g/3\}$ and assume that $f < g$. We define $\omega : \Delta \cup X \to \mathbb{R}_0^+$ by $\omega(x) = 1$ for all $x \in X$, $\omega(f) = 2$ and $\omega(g) = 5$. It is clear that ω is an admissible weight function. Let us now consider the trees $t = g(x, x, x)$ and $t' = f(g(x, x, f(x, x)), f(f(x, g(x, x, x)), x))$.



Although $\omega(t) = 8$ and $\omega(t') = 27$, t' is smaller than t in lexicographical path order (recall that $f < g$). Nevertheless, because of the weights, $t <_{kbo} t'$.

Let us note that in many applications no separate leaf alphabets are used, but a special set of nullary symbols is singled out when the need arises. Since this can often be done without any loss of generality, we shall not use variables in our presentation. If $X = \emptyset$ in the above definitions, $\mathbb{T}_\Delta(X)$ becomes the set $\mathbb{T}_\Delta$ of Δ-trees, and $\mathbb{C}_\Delta(X)$ the set $\mathbb{C}_\Delta$ of Δ-contexts and we write $\mathcal{R} = (Q, \Delta, \delta, F)$ instead of $\mathcal{R} = (Q, \Delta, \emptyset, \delta, F)$. Notice that $(\mathbb{T}_\Delta, <_{kbo})$ is an ordered set, and $<_{kbo}$ is a strict total order. Moreover, for $t, t' \in \mathbb{T}_\Delta$, we write that $t \leq_{kbo} t'$ if $t <_{kbo} t'$ or $t = t'$.

2.3 Learning Models

The overall goal of any learning algorithm is to be able to identify a *target language* from a given family of languages after receiving some *information* about that language. The information can be given in different ways: it can be a sequence of positive examples (i.e., strings in the target language), or a sequence of labeled examples (i.e., pairs of strings and labels, the labels indicating whether or not the strings are in the target language). The choice for the sequence of strings can be arbitrary, or they can be drawn according to a fixed distribution. The learning algorithm may also have access to an oracle that can answer specific kind of queries (the *active learning model*). Regardless of which type of information is accessible to the learner, the output of the algorithm might be the target language or a good enough approximation of it (*exact identification* versus *probably approximately correct learning*). On the other hand, the algorithm may output just one hypothesis (*one-shot learners*), a bounded number of hypotheses such that the final one is the good one (*inductive inference* *with bounded mind changes* [Muk92a, LZ93c, LZ93b]), or it can keep outputting hypotheses until it converges to the correct one and never changes its mind afterwards (*learning in the limit model* [Gol67]).

In this dissertation we focus on exact identification and active learning. Moreover, all results concern the learnability of nonempty recursive languages. The systematic study of these families was started by Angluin [Ang79, Ang80a, Ang80b] and justified by potential applications.

We have not investigated the question of when an indexed family of nonempty recursively enumerable languages is inferable from positive data, since the motivating applications all seem to be families of recursive languages. **D. Angluin [Ang80b]**

Let $\mathcal{C}$ be a class of nonempty recursive languages over Σ^*. Then $\mathcal{C}$ is an *indexable class* (or *indexed family*) 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 exists 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 $\mathcal{C}$. Well-known examples of indexed families are the set of all context sensitive languages in canonical enumeration (see Hopcroft and Ullman [HU69]) or the set of all pattern languages in canonical enumeration (see Angluin [Ang79, Ang80a]).

Another important aspect when speaking about a learning model is the choice of the hypotheses space [LZ93a, Lan94]. It is clear that the hypotheses space must contain at least one description for each target language. That is why many authors investigated the case where the indexed family itself is the hypotheses space - the *exact learning* model (for example, Angluin [Ang80a, Ang80b], Shinohara [Shi83], Jantke [Jan91], Mukouchi [Muk92b]).

... looking at potential applications of a learning system, users of such a system might even be highly interested in getting as hypotheses just the descriptions they proposed. That means they might formulate their learning problems just by specifying a particular indexed family.
T. Zeugmann and S. Lange [ZL95]

One may also choose as hypotheses space for a language class $\mathcal{C}$ a sequence $\mathcal{H} = G_0, G_1, \ldots$ of grammars such that each grammar describes a language in the class to be learned (i.e., $\mathcal{L}(G_i) \in \mathcal{C}$). This model is known in the literature as *class preserving*.

From a practical point of view, it is obviously advantageous to use a hypotheses space that is as small as possible in that it provides

exclusively descriptions for languages which are subject to learning.
T. Zeugmann and S. Lange [ZL95]

Of course, there might be various possibilities for choosing a hypotheses space for a given family of languages. Take for example the case of the class *Reg* of all regular languages. One may use as hypotheses space an indexing of all DFAs, as well as the NFAs, or even the regular expressions. Clearly, in a given setting, the same class may be learnable with respect to a certain hypotheses space and not learnable with respect to another. A classical example is the polynomial time learnability of DFAs and NFAs from a Minimally Adequate Teacher[1] (MAT). It has been shown that, under certain cryptographic assumptions, NFAs are not learnable with membership and equivalence queries in time polynomial in the size of a minimum NFA and the length of the longest counterexample returned by the teacher [AK91]. On the other hand, it is well-known that DFAs are polynomial time learnable from MAT [Ang87c].

Sometimes though, allowing a *class comprising* hypotheses space (i.e., a hypotheses space $\mathcal{H} = G_0, G_1, \ldots$ such that for every $L \in \mathcal{C}$ there exists $G_i \in \mathcal{H}$ with $\mathcal{L}(G_i) = L$) might lead to better learnability capabilities. Indeed, just imagine that the language class we want to learn is the class $\mathcal{S}$ of all singleton languages over a fixed alphabet Σ. Clearly, the learner may ask, if allowed to, if the empty set is the target language. The only possible counterexample in this case will disclose the target language. On the other hand, if the learner is constricted to ask only *proper* equivalence queries (that is, queries with languages from the class to be learned), then one may need to ask almost as many questions as languages in the class when faced with an adversary oracle. Angluin tackles the issue of learning with proper equivalence queries versus learning with extended equivalence queries in [Ang01, Ang04].

Actually, choosing an appropriate hypotheses space can be sometimes crucial for achieving the desired learning goal.

> *Results obtained in the setting of PAC-learning impressively show that at least the efficiency of learning can be heavily affected if one insists to learn with respect to a particular hypotheses space (see, e.g., Pitt and Valiant [PV88], Blum and Singh [BS89]). Similar effects have been observed in Gold-style language learning, too (see Lange and Zeugmann [LZ93c]).* **T. Zeugmann and S. Lange [ZL95]**

Furthermore, a language class $\mathcal{C}$ is said to be *absolutely learnable* provided it can be learned with respect to every class preserving hypotheses space for it (see

[1]Details about this model will be given later in this chapter.

Freivalds and Zeugmann [FZ96]). This is obviously a rather strong requirement, not very useful in practice.

Since in our work we extensively make use of results proved in the exact learning framework, the hypotheses space in what follows is either the indexed family itself, when the focus is on general learnability results, or a class preserving family of generators such as DFAs, k-reversible DFAs, patterns, etc.

2.3.1 Query Learning

In this section we closely follow the definitions presented in [LZ04b]. In the query learning model a learner has access to an oracle that truthfully answers queries of a specified kind. A *query learner* is an algorithmic device that, depending on the reply of the previous queries, either computes a new query, or returns a hypothesis and halts.

More formally, let $\mathcal{C}$ be an indexable class, let $L \in \mathcal{C}$ and let $\mathcal{H}$ be a hypotheses space for $\mathcal{C}$ (recall that $\mathcal{H}$ can be an indexed family of either languages $(L_i)_{i\in\mathbb{N}}$ or grammars $(G_i)_{i\in\mathbb{N}}$, depending on the chosen learning model). The query learner *$\mathcal{A}lg$ learns L with respect to $\mathcal{H}$ using some type of queries* if it eventually halts and its only hypothesis, say i, correctly describes L (i.e., $L_i = L$ for the former case, or $\mathcal{L}(G_i) = L$ for the last one). So, $\mathcal{A}lg$ returns its unique and correct guess i after only finitely many queries. Moreover, *$\mathcal{A}lg$ learns the class $\mathcal{C}$ with respect to $\mathcal{H}$ using some type of queries* if it learns every language of that class with respect to $\mathcal{H}$ using queries of the specified type.

The most investigated types of queries are:

- *Membership queries* (MQs). The input is a string w, and the answer is `Yes` or `No`, depending on whether or not w belongs to the target language.

- *Equivalence queries* (EQs). The input is an index j of some language $L_j \in \mathcal{H}$ (grammar $G_j \in \mathcal{H}$, respectively). If $L = L_j$ ($L = \mathcal{L}(G_j)$), the answer is `Yes`. Otherwise together with the answer `No`, a counterexample from $(L_j \backslash L) \cup (L \backslash L_j)$ $((\mathcal{L}(G_j)\backslash L) \cup (L\backslash\mathcal{L}(G_j)))$ is supplied.

The collections of all indexable classes $\mathcal{C}$ for which there is a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using membership (or equivalence) queries are denoted by *MemQ* (*EquQ*, respectively).

2.3.2 Gold-style Learning

In order to present the Gold-style learning models we need some further notions, briefly explained below (for details, see [Gol67, Ang80b, ZL95]).

Let L be a nonempty language. A *text for* L is an infinite sequence $\sigma = w_1, w_2, w_3, \ldots$ such that $\{w_i \mid i \geq 1\} = L$. An *informant for* L is an infinite sequence $\sigma = (w_1, b_1), (w_2, b_2), (w_3, b_3), \ldots$ with $b_i \in \{0, 1\}$ for all $i \geq 1$ such that $\{w_i \mid i \geq 1 \text{ and } b_i = 1\} = L$ and $\{w_i \mid i \geq 1 \text{ and } b_i = 0\} = \Sigma^* \backslash L$.

Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. An *inductive inference machine* (IIM) is an algorithmic device that reads longer and longer initial segments σ of a text (informant), and outputs numbers as its hypotheses. An IIM returning some i is construed to hypothesize the language L_i. Given a text (an informant) σ for a language $L \in \mathcal{C}$, $\mathcal{A}lg$ *learns* L *from* σ if the sequence of hypotheses output by $\mathcal{A}lg$, when fed σ, stabilizes on a number i (i.e., past some point $\mathcal{A}lg$ always outputs the hypothesis i) with $L_i = L$. We say that $\mathcal{A}lg$ *learns* $\mathcal{C}$ *from text (informant)* if it identifies each $L \in \mathcal{C}$ from every corresponding text (informant).

A slightly modified version of the learning in the limit model is the model of *conservative learning* (see [ZLK95, Zeu06] for more details). A *conservative* IIM is only allowed to change its mind in case its actual guess contradicts the data seen so far.

As above, *LimTxt* (*LimInf*) denotes the collection of all indexable classes $\mathcal{C}$ for which there is an IIM $\mathcal{A}lg$ such that $\mathcal{A}lg$ identifies $\mathcal{C}$ from text (informant). One can similarly define *ConsvTxt* and *ConsvInf*, for which the inference machines should be conservative IIMs.

Although an IIM is allowed to change its mind finitely many times before returning its final and correct hypothesis, in general it is not decidable whether or not it has already output its final hypothesis. Hence, the learner must go on processing information forever because there is always the possibility that some future information will force him to change his guess. As opposed to that, in the *finite identification model*, the learner is required to know when his answer is correct, that is, he has to stop the presentation of information at some finite time when he thinks it has received enough, and state the identity of the unknown object (see [Gol67]). The corresponding models *FinTxt* and *FinInf* are defined as above.

2.3.3 A Hierarchy of Learning Models

There has been quite a lot of work done for comparing the aforementioned learning methods and finding nice characterizations for the classes of languages inferable within specific settings. We present in what follows only those results which will be needed in our proofs (see [Lan00] for details).

Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class.

Definition 1 (Angluin [Ang80b])**.** *A set T_i is a* finite tell-tale *of L_i if*

1. *T_i is a finite subset of L_i, and*

2. *for all $j \geq 1$, if $T_i \subseteq L_j$ then L_j is not a proper subset of L_i.*

Theorem 1 (Angluin [Ang80b]). *The class $\mathcal{C}$ is in LimTxt if and only if there exists an effective procedure which on any input $i \geq 1$ enumerates a finite tell-tale of L_i.*

Theorem 2 (Zeugmann, Lange, Kapur [ZLK95]). *The class $\mathcal{C} = (L_i)_{i\geq 1}$ belongs to ConsvTxt if and only if there exists an uniformly computable family $(T_i^j)_{i,j\geq 1}$ of finite sets such that*

1. *for all $L \in \mathcal{C}$, there exists i with $L_i = L$ and $T_i^j \neq \emptyset$ for almost all $j \geq 1$;*

2. *for all $i, j \geq 1$, $T_i^j \neq \emptyset$ implies $T_i^j \subseteq L_i$ and $T_i^j = T_i^{j+1}$;*

3. *for all $i, j, k \geq 1$, $\emptyset \neq T_i^j \subseteq L_k$ implies $L_k \not\subset L_i$*

Definition 2 (Mukouchi [Muk92a]). *A language L is* consistent *with a pair of sets $\langle T, F\rangle$ if $T \subseteq L$ and $F \subseteq \Sigma^*\backslash L$. The 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 $\langle T, F\rangle$, then $L_j = L_i$.*

Theorem 3 (Mukouchi [Muk92a]). *The class $\mathcal{C} = (L_i)_{i\geq 1}$ is in FinInf if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any $i \geq 1$.*

Since the class *FinInf* coincides with *MemQ* (see [LZ04b], for example), we get the following corollary.

Corollary 2. *The 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.*

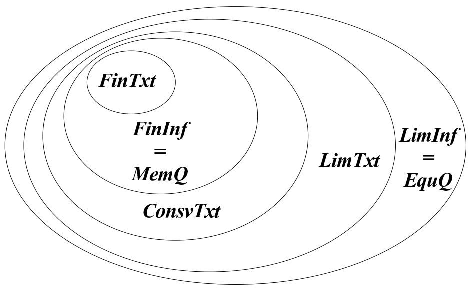


Figure 2.1: *FinTxt* $\subset$ *FinInf* $=$ *MemQ* $\subset$ *ConsvTxt* $\subset$ *LimTxt* $\subset$ *LimInf* $=$ *EquQ*

2.3.4 Polynomial Time Algorithms; Key Ideas

So far we have been talking about language learning in a general setting, where no complexity measures are involved. It is clear though that in practice it is the efficiency of the algorithms that really counts. The word "efficiency" itself raises a few questions and opens plenty of space for discussion.

First of all, when we talk about polynomial algorithms we have to mention the variable with respect to which we measure the time complexity. Clearly, if we take as reference the cardinality of the languages being learned, then even linear algorithms would imply an infinite computational time when the target language is not finite. That is why, when we speak about the size of a language, we understand the size of its smallest representation (e.g., the size of the minimal complete DFA for regular languages, or the smallest context-free grammar for CFLs).

Once we have the size problem settled, one still needs to decide what exactly do we want to bound. Is it the total running time we are concerned with, or just the update time [Pit89]? Of course, we might as well be interested in the number of errors before converging (implicit prediction errors [Pit89]). Or maybe we just need to bound the number of mind changes [AS83, CS83, DS86], or the size of the characteristic sample [dlH97]. In active learning, the two most used complexity measures are the number of queries and the total running time.

2.4 The Algorithm L^*

Since many of the algorithms proposed in this monograph are adaptations of Angluin's original algorithm L^* (Algorithm 7 in Section 4.1.2, Algorithm 9 in Section 5.1.1, Algorithm 11 in Section 5.2.1 and Algorithm 13 in Section 6.1.1), we briefly highlight the concepts used there, we see how L^* runs on an example, and we discuss its time and query complexity. The same target language will be used as running example throughout this monograph, in order to facilitate a comparison between our algorithms and L^*.

Let us recall that in a standard query learning algorithm, the *learner* interacts with a *teacher* who knows the target language $L \subseteq \Sigma^*$ and is assumed to answer correctly. The goal of the algorithm is to come up with a minimal representation for the language L.

For L^*, the target is a regular language, and the hypotheses space is formed by all minimal complete DFAs. Based on the answers to the MQs provided by the teacher, the learner constructs an *observation table* containing 0's and 1's. When the table is *closed* (there is no row in the lower part of the table which cannot be found in the upper part) and *consistent* (if there are two strings

u and v in the upper part of the table with equal row values, then the rows of ua and va should also be equal for any a in Σ), the algorithm constructs an automaton and asks an EQ. If the answer to the EQ is `No`, the teacher returns also a *counterexample*, which is a string in the symmetric difference of the two languages: the target language and the language inferred by the learner. Otherwise, it outputs the conjectured DFA and halts. We will not get into any further details because L* is not the main topic of our research (the reader is referred to [Ang87c] for a complete description of the algorithm and its features).

Let us assume that the language to be learned is $L = (a + bba)^+$ over the alphabet $\Sigma = \{a, b\}$. The minimal DFA $\mathcal{A}_L$ is represented in Figure 2.2.

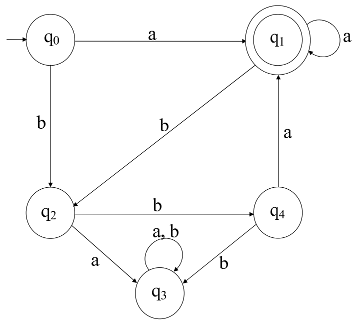


Figure 2.2: Minimal complete DFA for the language $L = (a + bba)^+$

The algorithm starts with $S = \{\lambda\}$, $E = \{\lambda\}$ and the following observation table (see Table 2.3).

Table 2.3: $S = \{\lambda\}, E = \{\lambda\}$

First Table		E λ
S	λ	0
$S\Sigma \backslash S$	a	1
	b	0

Table 2.4: $S = \{\lambda, a\}, E = \{\lambda\}$

Second Table		E λ	State
S	λ	0	q_0
	a	1	q_1
$S\Sigma \backslash S$	b	0	q_0
	aa	1	q_1
	ab	0	q_0

One may notice that this table is not closed, since the value of $row(a)$ does not appear in the upper part of the table. So the algorithm proceeds by adding the string a to S and updating the table (see Table 2.4).

This table is closed and consistent, so the learner constructs the automaton represented in Figure 2.3, and asks an EQ.

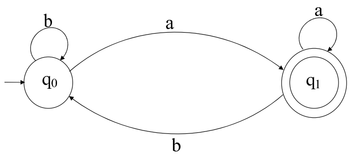


Figure 2.3: The automaton associated with Table 2.4

At this point, the teacher is supposed to answer `No` (since the conjectured automaton is clearly not isomorphic to the target one), and to provide a counterexample. Of course, this can be chosen completely arbitrary. Assume that the teacher returns the string ba as a counterexample (ba is accepted by the conjectured automaton although it is not in the target language). This string and all its prefixes are added to S, and the learner constructs an observation table as the one in Table 2.5.

Table 2.5: $S = \{\lambda, a, b, ba\}, E = \{\lambda\}$

Third Table		E
		λ
S	λ	0
	a	1
	b	0
	ba	0
$S\Sigma\backslash S$	aa	1
	ab	0
	bb	0
	baa	0
	bab	0

Table 2.6: $S = \{\lambda, a, b, ba\}, E = \{\lambda, a\}$

Forth Table		E	
		λ	a
S	λ	0	1
	a	1	1
	b	0	0
	ba	0	0
$S\Sigma\backslash S$	aa	1	1
	ab	0	0
	bb	0	1
	baa	0	0
	bab	0	0

Let us note that the strings λ and b have identical row values while $row(\lambda \cdot a)$ and $row(b \cdot a)$ are different. Since the table is not consistent, the algorithm adds the string a to E and updates the table (see Table 2.6).

Table 2.7: $S = \{\lambda, a, b, ba\}, E = \{\lambda, a, ba\}$

Fifth Table		E			State
		λ	a	ba	
S	λ	0	1	0	q_0
	a	1	1	0	q_1
	b	0	0	1	q_2
	ba	0	0	0	q_3
$S\Sigma\backslash S$	aa	1	1	0	q_1
	ab	0	0	1	q_2
	bb	0	1	0	q_0
	baa	0	0	0	q_3
	bab	0	0	0	q_3

The observation table continues to be not consistent, since $row(b)$ equals $row(ba)$ and the rows corresponding to the strings $b \cdot b$ and $ba \cdot b$ do not share the same value on column a. So, the string $b \cdot a$ is added to E (see Table 2.7).

Now the table is both closed and consistent, so the algorithm constructs the automaton represented in Figure 2.4.

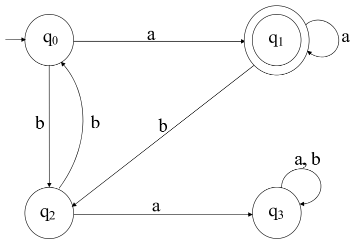


Figure 2.4: The automaton associated with Table 2.7

The conjectured automaton is still not the target one, so the teacher will answer accordingly, providing the learner with a counterexample. We have, of course, no way of knowing what this could be. But just for the sake of keeping the tables as small as possible, let us assume that the counterexample returned by the teacher is the smallest possible one, that is, the string *bbbba*. The algorithm adds *bbbba* and all its prefixes to S, and updates the observation table (see Table 2.8).

Table 2.8: $E = \{\lambda, a, ba\}, S = \{\lambda, a, b, ba, bb, bbb, bbbb, bbbba\}$

Sixth Table		E		
		λ	a	ba
S	λ	0	1	0
	a	1	1	0
	b	0	0	1
	ba	0	0	0
	bb	0	1	0
	bbb	0	0	0
	bbbb	0	0	0
	bbbba	0	0	0
$S\Sigma \backslash S$	aa	1	1	0
	ab	0	0	1
	baa	0	0	0
	bab	0	0	0
	bba	1	1	0
	bbba	0	0	0
	bbbbb	0	0	0
	bbbbaa	0	0	0
	bbbbab	0	0	0

Table 2.9: $E = \{\lambda, a, ba, bba\}, S = \{\lambda, a, b, ba, bb, bbb, bbbb, bbbba\}$

Seventh Table		E				Q
		λ	a	ba	bba	
S	λ	0	1	0	1	q_0
	a	1	1	0	1	q_1
	b	0	0	1	0	q_2
	ba	0	0	0	0	q_3
	bb	0	1	0	0	q_4
	bbb	0	0	0	0	q_3
	bbbb	0	0	0	0	q_3
	bbbba	0	0	0	0	q_3
$S\Sigma \backslash S$	aa	1	1	0	1	q_1
	ab	0	0	1	0	q_2
	baa	0	0	0	0	q_3
	bab	0	0	0	0	q_3
	bba	1	1	0	1	q_1
	bbba	0	0	0	0	q_3
	bbbbb	0	0	0	0	q_3
	bbbbaa	0	0	0	0	q_3
	bbbbab	0	0	0	0	q_3

It is easy to see that the table is not consistent: the strings λ and bb have identical row values, whereas $row(\lambda \cdot b)$ and $row(bb \cdot b)$ differ on column ba. The algorithm adds the string $b \cdot ba$ to E and constructs Table 2.9.

The language accepted by the conjectured automaton is finally the target one, so the algorithm terminates by outputting the right DFA. A simple counting shows us that during its run, the algorithm L* submits to the teacher a total number of 44 distinct strings as MQs, and asks 3 EQs.

Angluin shows in [Ang87c] that, if we denote by m the length of the longest counterexample returned by the teacher and by n is the size of the target language, then the total running time of L* can be bounded by a polynomial function of m and n, and:

- the observation table can be explicitly represented by a finite table of size polynomial in m and n ($\mathcal{O}(m^2n^2 + mn^3)$ suffices),
- the number of MQs and EQs can be bounded by $\mathcal{O}(mn^2)$ and n, respectively,

Chapter 3

Learning with Correction Queries

The way children learn their mother language is an amazing process. They receive examples of sentences in that language, and after some transitory period - in which they still make mistakes and are corrected by adults - they are suddenly able to express themselves fluently and errorless.

> *We have looked at the ways in which children learn language, and we have seen that they do indeed appear to learn a great deal from very little evidence, and that they do appear to build upon grammars that they could not have simply plucked from the language that they hear around them.* **T. Mason [Mas]**

It is clear that a child left alone with all kinds of teaching material would learn to speak slower (if ever) than one integrated in a community. The key difference between the above mentioned cases consists in the possibility for the second child to interact with others, and to be corrected when he or she makes mistakes. As we have already argued in the introductory chapter, this interaction within the process of child acquiring his native language is best described (among the existing learning models) by the query learning model [Ang87c]. First introduced, and in the same time the most used types of queries, are *membership queries* (MQs) and *equivalence queries* (EQs).

There are quite a few reasons though for which people working in grammatical inference, and especially in active learning, have been trying to find effective algorithms able to identify regular languages without the use of EQs. First of all, EQs are computationally costly. Secondly, they are quite unnatural for a real life setting: no child would ever ask if his or her current hypothesis represents the correct grammar of the language. Finally, it might happen that the teacher does not even have a grammar of the target language - take, for example, the case of native speakers that did not ever studied grammar.

On the other hand, membership queries are not informative enough, not being able to capture the feedback received by the child when he or she makes mistakes. Inspired by the way adults guide the process of children's language acquisition by correcting them when necessary, we investigate a modified version of MQs, called *correction query* (CQ), that incorporates this idea. More precisely, the difference consists in the fact that for strings not belonging to the target language, the teacher must provide the learner with a *correction*. Whereas in the case of natural languages the correction for an ungrammatical sentence would be one in which the adult is replacing the error with a correct (sequence of) word(s), in formal language theory different objects may require different types of corrections. Therefore, we are going to consider several types of CQs.

Intuitively, CQs are weaker than EQs and more informative than MQs. But what exactly can we learn with them? In this chapter we investigate the conditions which have to be met by a given class of languages in order to be learnable with CQs, we analyze the relations existing between different types of correction queries, as well as their connection to other well-known Gold-style and query learning models. We focus on the identification of formal languages ranging over indexable classes of nonempty recursive languages as target concepts and we do not take into account time complexity issues (polynomial time learning is discussed in Chapter 4).

3.1 Learning with Prefix Correction Queries

We begin our study with the type of corrections that were chronologically the first ones introduced. We are going to call them *prefix correction queries* (PCQs).

We would like to emphasize here that the particular choice we have made for this first type of correction has to do with the class of languages it was originally designed for, namely the class of regular languages. If we recall that, according to the Myhill-Nerode characterization for regular languages, two strings u, v are equivalent with respect to a language L if and only if they share the same set of tails (i.e., $\mathit{Tail}_L(u) = \mathit{Tail}_L(v)$), then it is clear that taking the correction for w to be the smallest string in the set $\mathit{Tail}_L(w)$ (if $\mathit{Tail}_L(w)$ is not empty) provides the learner with important information about the equivalence class of w.

More formally, if L is a formal language over the alphabet Σ and w is any string in Σ^*, we define the *prefix correcting string* of w with respect to L, and we denote it simply by $C_L(w)$, to be the smallest string in the lex-length order of the set $\mathit{Tail}_L(w)$, if this set is not empty, and the symbol $\Theta \notin \Sigma$ otherwise. Hence, C_L is a function from Σ^* to $\Sigma^* \cup \{\Theta\}$. Note that $C_L(w)$ is λ if and only if $w \in L$, and $C_L(w)$ equals Θ if and only if w is not the prefix of any of the strings in L.

Remark 2. *For any $u, v, w \in \Sigma^*$, if $C_L(u) = v \cdot w$ then $C_L(u \cdot v) = w$.*

Remark 3. *For any $u, v \in \Sigma^*$, if $\mathit{Tail}_L(u) = \emptyset$ then $\mathit{Tail}_L(u \cdot v) = \emptyset$.*

Remark 4. *For any $u \in \Sigma^*$, the following results hold:*

1. *If $C_L(u) \neq \Theta$ then $C_L(u \cdot C_L(u)) = \lambda$.*
2. *If $C_L(u) = \Theta$ then $\forall v \in \Sigma^*$, $C_L(u \cdot v) = \Theta$.*

As already stated in the introduction of this chapter, our goal is to gain a better understanding of what sort of language classes are inferable with these queries. With this in mind, we denote by *PCorQ* the collection of all indexable classes $\mathcal{C}$ for which there exists a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using PCQs.

In what follows we also pay a special attention to those classes of languages for which a teacher can be effectively implemented. More precisely, we look into indexable classes $\mathcal{C} = (L_i)_{i \geq 0}$ that have the following property $\mathbb{A}$: there exists a recursive function $\phi : \mathbb{N} \times \Sigma^* \to \Sigma^* \cup \{\Theta\}$ such that $\phi(i, w) = v$ if and only if $C_{L_i}(w) = v$ for any $w \in \Sigma^*$ and $L_i \in \mathcal{C}$. In other words, an indexable class $\mathcal{C} = (L_i)_{i \geq 0}$ has property $\mathbb{A}$ if and only if for all indexes i, $\mathit{Pref}(L_i)$ is recursive (computing the correction of a string w with respect to a language L can be easily done once we know whether w is in $\mathit{Pref}(L)$). Note that for an arbitrary recursive language L, the prefix $\mathit{Pref}(L)$ is not necessary recursive[1].

So, let us denote by $\mathit{PCorQ}^{\mathbb{A}}$ the collection of those classes of languages in *PCorQ* for which condition $\mathbb{A}$ is satisfied. Similarly, $\mathit{MemQ}^{\mathbb{A}}$ is defined. Clearly, for the language classes in $\mathit{PCorQ}^{\mathbb{A}}$ all answers to the PCQs can be effectively computed. So in this case we could speak about a teacher instead of an oracle.

3.1.1 Necessary and Sufficient Conditions

In this section we give some necessary and sufficient conditions for a class of languages to be learnable with PCQs alone. For this, we need some further definitions and notations.

Recall that Mukouchi gives a characterization for finite learning from informant [Muk92a] (and hence for learning with MQs) in terms of what he calls

[1]See Corollary 5 later in this chapter.

pairs of definite finite tell-tales (see Definition 2 in Section 2.3.3), probably inspired by Angluin's *finite tell-tales* [Ang80b]. Basically, what he shows is that an indexable class $\mathcal{C} = (L_i)_{i \geq 1}$ is learnable with MQs if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any $i \geq 1$. In other words, a language class is inferable with MQs if for each language in the class, there exists a pair of finite sets - one with positive examples and the other one with negative examples - which makes it unique, and such a pair of sets can be effectively computed.

First of all, we claim that this property is no longer enough to characterize the model of learning with PCQs. A formal proof is given later in this chapter, but let us just anticipate a bit, and give the intuition for what comes next. Take the following example: if $K_1, K_2, K_3, \ldots$ is the collection of all finite nonempty sets of positive integers (indexed somehow), and $\Sigma = \{a\}$, we define $L_i = \{a^n \mid n \in K_i\}$ for all $i \geq 1$, and $\mathcal{C} = (L_i)_{i \geq 1}$. One may imagine a simple algorithm that learns any language L in $\mathcal{C}$ with PCQs. Indeed, it is enough to ask PCQs for the strings $w_0, w_1, w_2, \ldots, w_n$ until the oracle returns the answer Θ (i.e., $C_L(w_n) = \Theta$), where $w_0 = \lambda$ and $w_{i+1} = w_i \cdot C_L(w_i) \cdot a$ for all $i \in \{0, \ldots, n-1\}$. On the other hand, for all possible pairs of finite sets $\langle T, F \rangle$ such that $T \subseteq L$ and $F \subseteq \Sigma^* \backslash L$, there is always a bigger language L' in $\mathcal{C}$ which includes T and *avoids* F (i.e., does not contain any element of F). Hence, although the class $\mathcal{C}$ does not admit pairs of definite finite-tell tales, it is still learnable with PCQs.

From this simple example we deduct that in order to learn a class in $PCorQ$ one may also need information about those strings that have the special symbol Θ as correction. Keeping this in mind, we introduce the notion of *triples of definite finite tell-tales.*

We say that a language L is *consistent with a triple of sets* $\langle T, F, U \rangle$ if $T \subseteq L$, $F \subseteq \Sigma^* \backslash L$ and $U \subseteq \Sigma^* \backslash Pref(L)$.

The triple $\langle T, F, U \rangle$ is a *triple of definite finite tell-tales* of the language L in $\mathcal{C} = (L_i)_{i \geq 1}$ if :

1. T, F and U are finite,
2. L is consistent with $\langle T, F, U \rangle$, and
3. for all $j \geq 1$, if L_j is consistent with $\langle T, F, U \rangle$, then $L_j = L$.

Going back to our example, it can be checked that $\langle T_i, F_i, U_i \rangle$ is a triple of definite finite tell-tales for L_i, where $T_i = L_i$, $l = \max\{n \mid n \in K_i\}$, $F_i = \{a^n \mid n \in \{1, \ldots, l\} \backslash K_i\}$ and $U_i = \{a^{l+1}\}$. A formal proof is given in Lemma 4 later in this chapter.

We are going to establish necessary and sufficient conditions for a class of languages to be learnable with PCS based on triples of definite finite tell-tales.

In what follows we use the notion of convergence in the following way: we say that a series of triples of sets $\langle T_j, F_j, U_j\rangle_{j\geq 1}$ converges, in the limit, to some triple $\langle T_*, F_*, U_*\rangle$ if there exists an $N \geq 1$ such that for all $n \geq N$, $\langle T_n, F_n, U_n\rangle = \langle T_*, F_*, U_*\rangle$.

Proposition 2 (Necessary condition)**.** *If the class $\mathcal{C} = (L_i)_{i\geq 1}$ is in PCorQ, then there exists an effective procedure which enumerates, for any input $i \geq 1$, an infinite series of triples $\langle T_j, F_j, U_j\rangle_{j\geq 1}$ that converges in the limit to a triple of definite finite tell-tales of L_i.*

Proof. Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class in *PCorQ*, and $\mathcal{A}lg$ a query learning algorithm that learns $\mathcal{C}$ using PCQs. Since the class $\mathcal{C}$ does not necessarily have property $\mathbb{A}$, the answers to PCQs might not always be computable. We use this observation to design an effective procedure as described above. Whenever the oracle is queried with the string w, our teacher will return the value $C_{L^{\leq n}}(w)$ where n is a fixed natural number and $L^{\leq n} = \{w \in L \mid |w| \leq n\}$. Of course, $C_{L^{\leq n}}(w)$ and $C_L(w)$ might be different, so $\mathcal{A}lg$ is not sure to converge anymore (and even if it does, it might converge to a language that is different from the target one). That is why we only run it for at most a finite number of steps, avoiding possible loops.

The following procedure[2] outputs an infinite series of triples that converges in the limit to a triple of definite finite tell-tales of L.

Algorithm 1 A series convergent to a triple of definite finite tell-tales

```
1: Input: the target language $L$
2: $n := 0$
3: **while** TRUE **do**
4:    $n := n + 1$
5:    run $\mathcal{A}lg$ on $L$ at most $n$ steps, and collect the sequence of queries and
6:    answers from the implemented teacher w.r.t. the language $L^{\leq n}$ in $QA_n$
7:    $T_n := \{wv \mid (w, v) \in QA_n$ and $v \neq \Theta\}$
8:    $F_n := \{wv' \mid (w, v) \in QA_n, v \neq \Theta$ and $v' \prec v\}$
9:    $U_n := \{w \mid (w, \Theta) \in QA_n\}$
10:   output $\langle T_n, F_n, U_n\rangle$
11: **end while**
```

We show that the sequence of triples produced by Algorithm 1 converges to a triple of definite finite tell-tales of L.

Let QA_* be the sequence of queries and answers processed by $\mathcal{A}lg$ when learning L and m the number of time steps that $\mathcal{A}lg$ performs (hence the cardinality of QA_* is m). Take $T_* = \{wv \mid (w, v) \in QA_*, v \neq \Theta\}$, $F_* = \{wv' \mid (w, v) \in QA_*, v \neq \Theta$ and $v' \prec v\}$ and $U_* = \{w \mid (w, \Theta) \in QA_*\}$. Let us first show that $\langle T_*, F_*, U_*\rangle$ is a triple of definite finite tell-tales of L.

[2]The idea of constructing a teacher that provides partial answers belongs to S. Kobayashi.

Clearly, T_*, F_* and U_* are all finite. Note that if $u \in T_*$, then there exist w, v in Σ^* such that $u = wv$ and $v = C_L(w)$. Hence, $u = wv \in L$. If $u \in F_*$, then there exist w, v, v' in Σ^* such that $v' \prec v$, $u = wv'$ and $v = C_L(w)$. Hence, $u = wv' \notin L$. If $u \in U_*$, then $C_L(u) = \Theta$, and hence $u \in \Sigma^* \backslash \mathit{Pref}(L)$. So, $T_* \subseteq L$, $F_* \subseteq \Sigma^* \backslash L$ and $U_* \subseteq \Sigma^* \backslash \mathit{Pref}(L)$.

Let us take i such that L_i is consistent with $\langle T_*, F_*, U_* \rangle$. We compute $C_{L_i}(w)$ for each pair (w, v) in QA_*. If $v = \Theta$, then $w \in U_*$. But $U_* \subseteq \Sigma^* \backslash \mathit{Pref}(L_i)$ implies $w \notin \mathit{Pref}(L_i)$, and hence $C_{L_i}(w) = \Theta$. If $v \in \Sigma^* \backslash \{\Theta\}$, then $wv \in T_*$ and $wv' \in F_*$ for all $v' \prec v$. From $T_* \subseteq L_i$ and $F_* \subseteq \Sigma^* \backslash L_i$, we get $wv \in L_i$ and $wv' \notin L_i$ for all $v' \prec v$, and hence $C_{L_i}(w) = v$.

We have shown that for all $(w, v) \in QA_*$, $C_{L_i}(w) = v = C_L(w)$. Since the algorithm $\mathcal{A}lg$ is assumed to identify a unique language from the class $\mathcal{C}$, we obtain $L_i = L$. Hence, $\langle T_*, F_*, U_* \rangle$ is a triple of definite finite tell-tales of L.

If we take $l = \max\{|wv| \mid (w, v) \in QA_*\}$, where the length of Θ is defined as 0, we have that for all $n \geq l$ and all pairs (w, v) in QA_*, $C_L(w) = v = C_{L^{\leq n}}(w)$. So, if $N = \max\{l, m\}$ then for all $n \geq N$, $\langle T_n, F_n, U_n \rangle = \langle T_*, F_*, U_* \rangle$. □

As a direct consequence we obtain the following corollary.

Corollary 3. *If the class $\mathcal{C} = (L_i)_{i \geq 1}$ is in PCorQ, then a triple of definite finite tell-tales of L_i does exist for any index i.*

So, we know that for all language classes $\mathcal{C}$ in *PCorQ*, every language in $\mathcal{C}$ has a triple of definite finite tell-tales. Proposition 3 shows that having a way of computing such a triple is a sufficient condition for an indexable class of languages to be in *PCorQ*.

Proposition 3 (Sufficient condition)**.** *Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. If a triple of definite finite tell-tales of L_i is uniformly computable for any i, then $\mathcal{C}$ is in PCorQ.*

Proof. Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class for which a triple of definite finite tell-tales $\langle T_i, F_i, U_i \rangle$ of L_i is uniformly computable for any index i, and let $w_1, w_2, \ldots$ be the lex-length enumeration of all strings in Σ^*. If L is the target language, then the following query learning algorithm identifies L using PCQs.

Algorithm 2 A PCQ algorithm for the language L in $\mathcal{C}$

```
1: $T := \emptyset$, $F := \emptyset$, $U := \emptyset$, $j := 1$
2: while TRUE do
3:   get from the oracle the answer to $C_L(w_j)$
4:   if $(C_L(w_j) = \Theta)$ then
5:     $U := U \cup \{w_j\}$, $F := F \cup \{w_j\}$
6:   else
7:     $T := T \cup \{w_j \cdot C_L(w_j)\}$
8:     if $C_L(w_j) \neq \lambda$ then
9:       $F := F \cup \{w_j\}$
10:    end if
11:  end if
12:  for $i := 1$ to $j$ do
13:    if $(T_i \subseteq T$, $F_i \subseteq F$ and $U_i \subseteq U)$ then
14:      output $i$ and halt
15:    end if
16:  end for
17:  $j := j + 1$
18: end while
```

It is not very difficult to see that if Algorithm 2 outputs a hypothesis, then it is the correct one. Indeed, since we have constructed T, F and U such that $T \subseteq L, F \subseteq \Sigma^* \backslash L$ and $U \subseteq \Sigma^* \backslash Pref(L)$, it is clear that as soon as we have $T_i \subseteq T$, $F_i \subseteq F$ and $U_i \subseteq U$ for some $i \geq 1$, the target language L is consistent with the triple $\langle T_i, F_i, U_i \rangle$, and hence the algorithm outputs i such that $L_i = L$.

Now, let us prove that after asking a finite number of queries, the sets T, F and U are large enough to include T_i, F_i and U_i, respectively, where i is the smallest index such that $L_i = L$. Let j_1, j_2, j_3 and l be such that $j_1 = \max\{j \mid w_j \in T_i\}$, $j_2 = \max\{j \mid w_j \in F_i\}$, $j_3 = \max\{j \mid w_j \in U_i\}$ and $l = \max\{j_1, j_2, j_3, i\}$.

Consider the triple $\langle T, F, U \rangle$ constructed after receiving the corrections for the strings $w_1, w_2, \ldots, w_l$.

- For any $w \in T_i$ we have $w \preceq w_l$ and $C_L(w) = \lambda$. Hence, $w \in T$ (line 7).
- For any $w \in U_i$ we have $w \preceq w_l$ and $C_L(w) = \Theta$. Hence, $w \in U$ (line 5).
- For any $w \in F_i$ we have $w \preceq w_l$ and $C_L(w) \neq \lambda$. We distinguish two cases. Either $C_L(w) \in \Sigma^+$ and then w is added to F at line 9 of the algorithm, or $C_L(w) = \Theta$ and w is added to F at line 5 of the algorithm. In both cases, $w \in F$.

We have seen that after getting the corrections of at most l strings, $T_i \subseteq T$, $F_i \subseteq F$ and $U_i \subseteq U$, and since i is smaller than or equal to l, the algorithm outputs the (correct) hypothesis i. $\square$

So, we have found necessary and sufficient conditions for a class of languages to be learnable with PCQs. In the sequel we show that if we restrict to indexable classes with property $\mathbb{A}$, then there exists a characterization of the learning with PCQs model in terms of triples of finite sets. For this, let us first state and prove the following proposition.

Proposition 4 (Necessary condition for $PCorQ^{\mathbb{A}}$)**.** *If $\mathcal{C} = (L_i)_{1\geq 1}$ belongs to $PCorQ^{\mathbb{A}}$, then a triple of definite finite tell-tales of L_i is uniformly computable for any index i.*

Proof. Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class in $PCorQ^{\mathbb{A}}$, and take $\mathcal{A}lg$ to be a query learning algorithm which learns $\mathcal{C}$ using PCQs. Notice that in this case the answer to any PCQ can be effectively computed. Algorithm 3 outputs a triple of definite finite tell-tales for an arbitrary language L in $\mathcal{C}$.

Algorithm 3 Computing a triple of definite finite tell-tales

```
1: Input: the target language $L$
2: run $\mathcal{A}lg$ on $L$, and collect the sequence of queries and answers in $QA$
3: $T := \{wv \mid (w, v) \in QA, v \neq \Theta\}$
4: $F := \{wv' \mid (w, v) \in QA, v \neq \Theta$ and $v' \prec v\}$
5: $U := \{w \mid (w, \Theta) \in QA\}$
6: output $\langle T, F, U\rangle$ and halt.
```

It is straightforward to show that $\langle T, F, U\rangle$ is a triple of definite finite tell-tales of L (a similar argument is used in the proof of Proposition 2). $\square$

The following theorem is a direct consequence of Propositions 3 and 4, and provides a characterization for the class $PCorQ^{\mathbb{A}}$.

Theorem 4. *Let $\mathcal{C} = (L_i)_{1\geq 1}$ be an indexable class with property $\mathbb{A}$. Then $\mathcal{C}$ belongs to $PCorQ$ if and only if a triple of definite finite tell-tales of L_i is uniformly computable for any index i.*

A question that naturally arises is whether or not $PCorQ$ and $PCorQ^{\mathbb{A}}$ are equal. Although intuitively this is not the case, finding a class of languages in $PCorQ \setminus PCorQ^{\mathbb{A}}$ is not immediate. It was A. Okhotin who drew our attention to a recursive language L such that $Pref(L)$ is not recursive.

We have the following lemma.

Lemma 2 (Okhotin [Okh05])**.** *For every Turing machine (TM) M over an input alphabet Σ there exists an alphabet Γ and an encoding of computations $Cod_M : \Sigma^* \to \Gamma^*$, such that the language* $\textsc{Valc}(M) = \{w\natural Cod_M(w) \mid w \in \Sigma^*$ *and* $Cod_M(w)$ *is an accepting computation*$\}$ *over the alphabet* $\Omega = \Sigma\cup\Gamma\cup\{\natural\}$ *is an intersection of two LL(1) linear context-free languages $L_1, L_2 \subseteq \Sigma^*\natural\Gamma^*$.*

Given M, the corresponding LL(1) linear context-free grammars can be effectively constructed.

For the sake of self-containment, we present here the proof in [Okh05].

Sketch of proof. Let $V \supset \Sigma$ be the tape alphabet of M, let Q be its set of states, define $\Gamma = V \cup Q \cup \{\natural\}$. Encode an instantaneous description of M after i steps of computation on $w \in \Sigma^*$ as a word $ID_i = \alpha q a \beta \subseteq V^* Q V^+$, where the machine is in the state q and its head scans the symbol a. A computation history of M on w is encoded as

$$Cod_M(w) = ID_0 \cdot \natural \cdot ID_1 \cdot \natural \cdot \ldots \cdot \natural \cdot ID_{n-1} \cdot \natural\natural \cdot ID_n \cdot \natural \cdot (ID_n)^R \cdot \natural \cdot \ldots \cdot \natural \cdot (ID_1)^R \cdot \natural \cdot (ID_0)^R$$

where by x^R we denote the reverse of the string x.

It remains to construct two LL(k) linear context-free grammars G_1 and G_2, such that $L(G_1) \cap L(G_2) = \textsc{Valc}(M)$. The first grammar specifies the following conditions:

1. $ID_0^R = (q_0 a_1 a_2 \ldots a_m)^R$, i.e., the computation is indeed on the word $a_1 a_2 \ldots a_m$. This is an instance of the construct $\{xcx^R \mid x \in \{a,b\}^*\}$.
2. For all i $(0 \leq i < n)$, ID_i on the left and $(ID_{i+1})^R$ on the right are consecutive configurations of the Turing machine. Since M is deterministic, the symbols in $(ID_{i+1})^R$ are completely determined by the corresponding symbols in ID_i, hence this can be checked using LL(1) rules.
3. ID_n is a final configuration of M. The double marker $\natural\natural$ instructs the grammar to simulate a finite automaton that recognizes final configurations.

The second grammar simply verifies that, for every i, ID_i on the left and $(ID_i)^R$ on the right are indeed reverses of each other. This is another instance of $\{xcx^R \mid x \in \{a,b\}^*\}$. □

Corollary 4. *For every TM M, the language $\textsc{Valc}(M)$ is recursive.*

Corollary 5. *The prefix of a recursive languages is not necessarily recursive.*

Proof. Let us take a TM M_0 such that $\mathcal{L}(M_0) \subseteq \Sigma^*$ for some finite alphabet Σ and $\mathcal{L}(M_0) \in RE \backslash Rec$. By Corollary 4, $\textsc{Valc}(M_0)$ is recursive. On the other hand, $Pref(\textsc{Valc}(M_0)) \cap \Sigma^* \natural = \mathcal{L}(M_0)\natural$, and hence, $Pref(\textsc{Valc}(M_0)) \in RE \backslash Rec$. □

Now that we have a language with this property, we can construct a class of languages $\mathcal{C}^1 = (L_i^1)_{i \geq 1}$ over the alphabet Ω such that $\mathcal{C}^1 = (L_i^1)_{i \geq 1}$ is in

PCorQ. For example, take $L_0^1 = \text{VALC}(M_0)$ and $L_i^1 = \{a^i\natural\}$ for all $i \geq 1$, where a is any symbol in Σ. It is an easy exercise to construct a PCQ algorithm for learning $\mathcal{C}^1 = (L_i^1)_{i\geq 1}$ (asking one PCQ for the string λ suffices). Since $\mathcal{C}^1 = (L_i^1)_{i\geq 1}$ does not have property $\mathbb{A}$, it follows immediately that $\mathcal{C}^1 = (L_i^1)_{i\geq 1} \in PCorQ\backslash PCorQ^{\mathbb{A}}$.

3.1.2 Learning with Prefix Correction Queries versus Learning with Membership Queries

Let us recall that the notion of CQ itself appeared as an extension of the well-known and widely 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 (or phrase). Clearly, CQs can be thought of as some more informative MQs. So, it is only natural to compare the two learning settings (learning with CQs versus learning with MQs), and to analyze their expressive power.

When complexity issues are neglected, learning with MQs in finite time is exactly as powerful as learning in the limit from informant [Lan00]. So, thanks to the characterization we have for the class *FinInf* [Muk92a, ZLK95], we know that a language class is learnable with MQs if and only if for each language in the class a pair of definite finite tell-tales is uniformly computable (see Section 2.3.3, Theorem 3). Following this idea, we proved in Section 3.1.1 that the model of learning with PCQs can be characterized as well by (triples of) finite sets. The above mentioned results allow us to compare the two learning models: learning with PCQs and learning with MQs.

The Sets *MemQ* and $PCorQ^{\mathbb{A}}$ are Incomparable

Let us first show that $MemQ\backslash PCorQ^{\mathbb{A}}$ is not empty.

Lemma 3. *$\mathcal{C}^1 = (L_i^1)_{i\geq 1} \in MemQ\backslash PCorQ^{\mathbb{A}}$.*

Proof. Since $\mathcal{C}^1 = (L_i^1)_{i\geq 1} \notin PCorQ^{\mathbb{A}}$ it is enough to show that $\mathcal{C}^1 = (L_i^1)_{i\geq 1} \in MemQ$, that is, for any $i \geq 0$, a pair of definite finite tell-tales of L_i^1 is uniformly computable (by Corollary 2 from Section 2.3.3).

Clearly, $\langle\{a^i\natural\}, \emptyset\rangle$ is such a pair for $i \geq 1$. Moreover, since M_0 is a recursively enumerable TM, there exists a recursive procedure to print all the elements of $\mathcal{L}(M_0)$ (we know that $\mathcal{L}(M_0) \neq \emptyset$). If w_0 is the first element printed, then we can effectively compute $Cod_{M_0}(w_0)$. Clearly, $\langle\{w_0\natural Cod_{M_0}(w_0)\}, \emptyset\rangle$ is a pair of definite finite tell-tales of L_0^1. $\square$

In order to prove that $\mathit{PCorQ}^{\mathbb{A}}$ is not included in MemQ, we consider again the language class from the beginning of Section 3.1.1. So, if $K_1, K_2, K_3, \ldots$ is the collection of all finite nonempty sets of positive integers (indexed somehow), and $\Sigma = \{a\}$, we denote by L_i^2 the language $\{a^n \mid n \in K_i\}$ for all $i \geq 1$. Clearly, $\mathcal{C}^2 = (L_i^2)_{i \geq 1}$ is an indexable class[3] with property $\mathbb{A}$.

Lemma 4. $\mathcal{C}^2 = (L_i^2)_{i \geq 1} \in \mathit{PCorQ}^{\mathbb{A}} \backslash \mathit{MemQ}$.

Proof. We first prove that $\mathcal{C}^2 = (L_i^2)_{i \geq 1} \in \mathit{PCorQ}^{\mathbb{A}}$ by using the characterization of the class $\mathit{PCorQ}^{\mathbb{A}}$ in terms of triples of definite finite tell-tales (see Theorem 4). For an arbitrary index $i \geq 1$, we define $T_i = L_i^2$, $l = \max\{n \mid n \in K_i\}$, $F_i = \{a^n \mid n \in \{1, \ldots, l\} \backslash K_i\}$ and $U_i = \{a^{l+1}\}$. Obviously, the sets T_i, F_i and U_i are all finite, and the language L_i^2 is consistent with the triple $\langle T_i, F_i, U_i \rangle$. Let us take j such that L_j^2 is consistent with the triple $\langle T_i, F_i, U_i \rangle$. Then, $F_i \subseteq \Sigma^* \backslash L_j^2$ implies $(\{1, \ldots, l\} \backslash K_i) \cap K_j = \emptyset$, and $U_i \subseteq \Sigma^* \backslash \mathit{Pref}(L_j^2)$ implies $K_j \subseteq \{1, \ldots, l\}$. Putting together these last two results we obtain $K_j \subseteq K_i$, and hence $L_j^2 \subseteq L_i^2$. But $T_i \subseteq L_j^2$ implies $L_i^2 \subseteq L_j^2$, and hence $L_j^2 = L_i^2$. So $\langle T_i, F_i, U_i \rangle$ is a triple of definite finite tell-tales of L_i^2, and moreover, it can be uniformly computed.

Let us now assume that $\mathcal{C}^2 = (L_i^2)_{i \geq 1} \in \mathit{MemQ}$. Then, by Corollary 2 (Section 2.3.3), a pair of definite finite tell-tales $\langle T_i, F_i \rangle$ of L_i^2 is uniformly computable for any $i \geq 1$. Let us fix i, take $l = \max\{n \mid a^n \in F_i\}$, and set j to be the index for which $K_j = K_i \cup \{l+1\}$. Then, L_j^2 is also consistent with the pair $\langle T_i, F_i \rangle$ since $T_i \subseteq L_i^2 \subset L_j^2$ and $F_i \subseteq \Sigma^* \backslash L_j^2$ ($F_i \subseteq \Sigma^* \backslash L_i^2$ and $a^{l+1} \notin F_i$), and hence $L_j^2 = L_i^2$, which contradicts $L_j^2 \backslash L_i^2 = \{a^{l+1}\}$. $\square$

This last result can be extended to any alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$, if we set L_i^2 to be $\{a_1 a_2 \ldots a_{n-1} a_n^m \mid m \in K_i\}$ for any index i.

The Set *MemQ* is Strictly Included in *PCorQ*

Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. We have the following theorem.

Theorem 5. *If $\mathcal{C}$ is in MemQ, then $\mathcal{C}$ is in PCorQ.*

Proof. Assume that $\mathcal{C}$ is in MemQ. Then by Corollary 2, a pair of definite finite tell-tales $\langle T_i, F_i \rangle$ of L_i is uniformly computable for any index i. We show that $\langle T_i, F_i, \emptyset \rangle$ is a triple of definite finite tell-tales of L_i. Clearly, T_i is a finite subset of L_i, F_i is a finite subset of $\Sigma^* \backslash L_i$, and the empty set is a finite subset of $\Sigma^* \backslash \mathit{Pref}(L_i)$. Let us now take j such that L_j is consistent with the triple $\langle T_i, F_i, \emptyset \rangle$. Because $\langle T_i, F_i \rangle$ is a pair of definite finite tell-tales of L_i, $T_i \subseteq L_j$ and $F_i \subseteq \Sigma^* \backslash L_j$, we obtain $L_j = L_i$, and hence $\langle T_i, F_i, \emptyset \rangle$ is a triple of definite

[3]This class was proposed by S. Kobayashi.

finite tell-tales for L_i. Using Proposition 3, it follows immediately that $\mathcal{C}$ is in *PCorQ*. □

Notice that the inclusion is strict since $\mathcal{C}^2 = (L_i^2)_{i\geq 1} \in \mathit{PCorQ} \backslash \mathit{MemQ}$ (as a consequence of Lemma 4). Because the class *MemQ* is strictly included in *PCorQ*, we obtain that PCQs are more powerful than MQs, and they cannot be simulated by a finite number of MQs.

3.1.3 Learning with Prefix Correction Queries versus Gold-style Learning Models

Learning from queries is a one-shot learning model (i.e., the learner's first hypothesis is also the correct one) in which the learner receives rather global information (in the sense that, at any point, the oracle can be interrogated about any of the strings in the alphabet) about the object to be learned, being able to affect the sample of information, as opposed to Gold-style learning model where the information received is local (the learner cannot influence the sample) and the learner is allowed to change its current hypothesis when new information is received. Although this two approaches seem rather unrelated at first glance, there are several results that indicate the contrary [Lan00, LZ04a, LZ04b, LZ04c, LZ05]. For example, it has been shown that learning with MQs is equivalent with finite learning from informant, or that an indexable class is learnable using extra superset queries if and only if there is a conservative IIM that identifies this class from text [LZ04b].

In the sequel we show where our model is placed in the existing hierarchy of both query and Gold-style learning models. The first step was already done in Section 3.1.2: we proved that the classes *MemQ* and $\mathit{PCorQ}^{\mathbb{A}}$ are incomparable, and that *MemQ* is strictly included in *PCorQ* (consequently, *FinInf* and $\mathit{PCorQ}^{\mathbb{A}}$ are incomparable and *FinInf* is strictly included in *PCorQ*).

The Set $\mathit{PCorQ}^{\mathbb{A}}$ is Strictly Included in *ConsvTxt*

Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class. We have the following theorem.

Theorem 6. *If $\mathcal{C}$ is in $\mathit{PCorQ}^{\mathbb{A}}$, then $\mathcal{C}$ is in ConsvTxt.*

Proof. If $\mathcal{C} = (L_i)_{i\geq 1}$ is in $\mathit{PCorQ}^{\mathbb{A}}$ then, by Proposition 4, a triple of definite finite tell-tales $\langle T_i^*, F_i^*, U_i^* \rangle$ of L_i is uniformly computable for any index i. Moreover, we can assume without loss of generality that for all $i \geq 1$, T_i^* is not empty.

For all $i, j \geq 1$, we define T_i^j to be the set T_i^*. Clearly, $(T_i^j)_{i,j\geq 1}$ is a uniformly computable family of finite sets. Let us show that it also satisfies the three conditions of Theorem 2.

1. *for all $L \in \mathcal{C}$, there exists i with $L_i = L$ and $T_i^j \neq \emptyset$ for almost all $j \geq 1$;*
   True - they are all nonempty.

2. *for all $i, j \geq 1$, $T_i^j \neq \emptyset$ implies $T_i^j \subseteq L_i$ and $T_i^j = T_i^{j+1}$;*
   True.

3. *for all $i, j, k \geq 1$, $\emptyset \neq T_i^j \subseteq L_k$ implies $L_k \not\subset L_i$.*
   This last condition translates to: *for all $i, k \geq 1$, $T_i^* \subseteq L_k$ implies $L_k \not\subset L_i$*. So, let us assume that there exist $i, k \geq 1$ such that $T_i^* \subseteq L_k$ and $L_k \subset L_i$. It follows that $\mathit{Pref}(L_k) \subseteq \mathit{Pref}(L_i)$, and hence $\Sigma^* \backslash \mathit{Pref}(L_k) \supseteq \Sigma^* \backslash \mathit{Pref}(L_i)$. Keeping in mind that $U_i^* \subseteq \Sigma^* \backslash \mathit{Pref}(L_i)$ we obtain that $U_i^* \subseteq \Sigma^* \backslash \mathit{Pref}(L_k)$. Moreover, $F_i^* \subseteq \Sigma^* \backslash L_i$ and $\Sigma^* \backslash L_k \supseteq \Sigma^* \backslash L_i$ imply $F_i^* \subseteq \Sigma^* \backslash L_k$. Since L_k is consistent with the triple $\langle T_i, F_i, U_i \rangle$, we have $L_i = L_k$ which contradicts our assumption.

By Theorem 2, we obtain that $\mathcal{C}$ is in *ConsvTxt*. □

Let us now show that the inclusion is strict. For this, we denote by $I(n)$ the set of all positive integral multiples of n. Let the collection of all finite nonempty sets of prime positive integers be $P_1, P_2, P_3, \ldots$ indexed, for example, in order of increasing $\prod_{p \in P_i} p$. Then, take $\Sigma = \{a\}$, $R_i = \cup_{p \in P_i} I(p)$ and $L_i^3 = \{a^n \mid n \in R_i\}$. Clearly, $\mathcal{C}^3 = (L_i^3)_{i \geq 1}$ is an indexable class (the example is taken from [Ang80b]).

Lemma 5. *$\mathcal{C}^3 = (L_i^3)_{i \geq 1}$ is in $\mathit{ConsvTxt} \backslash \mathit{PCorQ}$.*

Proof. First notice that one can easily construct a conservative IIM that learns the class $\mathcal{C}^3 = (L_i^3)_{i \geq 1}$: it is enough to update the hypothesis only when in the presentation of information a string a^n appears, and n is either a prime number or a power of a prime number that was not seen before.

Now let us assume that $\mathcal{C}^3 = (L_i^3)_{i \geq 1}$ is in *PCorQ*. By Corollary 3, a triple of definite finite tell-tales $\langle T_i, F_i, U_i \rangle$ of L_i^3 does exist, for any index i. If we denote by $\mathit{Len}(L)$ the set of all possible lengths of strings in L (that is, $\mathit{Len}(L) = \{|w| \mid w \in L\}$), then $T_i \subseteq L_i^3$ is equivalent to $\mathit{Len}(T_i) \subseteq R_i$, and $F_i \subseteq \Sigma^* \backslash L_i^3$ is equivalent to $\mathit{Len}(F_i) \cap R_i = \emptyset$. Finally, $U_i \subseteq \Sigma^* \backslash \mathit{Pref}(L_i^3)$ implies $U_i = \emptyset$.

Let us now choose a prime number p such that $I(p) \cap \mathit{Len}(F_i) = \emptyset$ and $p \notin P_i$, and take j such that $P_j = P_i \cup \{p\}$. Clearly, $L_i^3 \subset L_j^3$. We show that L_j^3 is consistent with $\langle T_i, F_i, U_i \rangle$.

Indeed, $T_i \subseteq L_j^3$ because $T_i \subseteq L_i^3$ and $L_i^3 \subset L_j^3$. Also, $\mathit{Len}(F_i) \cap R_j = \emptyset$ because $\mathit{Len}(F_i) \cap R_i = \emptyset$, $\mathit{Len}(F_i) \cap I(p) = \emptyset$ and $R_j = R_i \cup I(p)$. Hence, $F_i \subseteq \Sigma^* \backslash L_j^3$. The empty set is trivially included in any set, and hence $U_i \subseteq \Sigma^* \backslash \mathit{Pref}(L_j^3)$.

We found an index j such that L^3_j is consistent with $\langle T_i, F_i, U_i\rangle$ and $L^3_j \neq L^3_i$ which is a contradiction, so $\mathcal{C}^3 = (L^3_i)_{i\geq 1}$ is not in *PCorQ*. □

As a direct consequence we obtain that $\mathcal{C}^3 = (L^3_i)_{i\geq 1} \in \mathit{ConsvTxt}\backslash \mathit{PCorQ}^{\mathbb{A}}$.

The Set *PCorQ* is Strictly Included in *LimTxt*

Let $\mathcal{C} = (L_i)_{i\geq 1}$ be an indexable class. We have the following theorem.

Theorem 7. *If $\mathcal{C}$ is in PCorQ, then $\mathcal{C}$ is in LimTxt.*

Proof. To prove this theorem, we use Angluin's characterization for the class of languages identifiable in the limit from positive data. Thus, by Theorem 1, it is enough to show that there exists an effective procedure which on any input $i \geq 1$, enumerates a finite tell-tale of L_i.

Let us fix the target language L, and consider the Algorithm 1 described in the proof of Proposition 2. We can modify it to output, instead of the triple $\langle T_i, F_i, U_i\rangle$, only the elements of the set T_i which did not appear previously, for all $i \geq 1$ (in order to avoid duplications). We show that the set $T := \cup_{i\geq 1} T_i$ is finite, and moreover, it is a finite tell-tale of L. Indeed, we have seen in the proof of Proposition 2 that for all $n \geq N$, $\langle T_n, F_n, U_n\rangle = \langle T_*, F_*, U_*\rangle$ is a triple of definite finite tell-tales for L. Clearly, $T_n = T_*$ is a finite tell-tale for L (a similar argument was used in the proof of Theorem 6). But $T = \cup_{i\geq 1} T_i = \cup_{i=1}^{N} T_i$ is a finite set included in L, and hence it is also a finite tell-tale of L. □

Moreover, the inclusion is strict since $\mathcal{C}^3 = (L^3_i)_{i\geq 1}$ is in *LimTxt* and not in *PCorQ*.

The Sets *PCorQ*, *ConsvTxt* and *LimTxt*

We have seen that both *PCorQ* and *ConsvTxt* are strictly included in *LimTxt*. In the sequel we show that there are languages in *LimTxt* which are not in *ConsvTxt* $\cup$ *PCorQ*. For this, consider the class of languages described by Angluin in [Ang80b], pp. 131-132.

Let us fix the alphabet $\Sigma = \{a, b\}$, and take a standard enumeration $\mathcal{A}lg_1$, $\mathcal{A}lg_2$, $\mathcal{A}lg_3$, ... of IIMs and some computable pairing function $\langle\cdot,\cdot\rangle : \mathbb{N}_+ \times \mathbb{N}_+ \to \mathbb{N}_+$. For all $k \geq 1$ define $L^4_{\langle k,1\rangle}, L^4_{\langle k,2\rangle}, L^4_{\langle k,3\rangle}, \dots$ as follows. $L^4_{\langle k,1\rangle} = \{a^{p_k^m} \mid m \geq 1\}$, where p_k is the kth prime number. Let σ_k be the sequence $a^{p_k}, a^{p_k^2}, a^{p_k^3}, a^{p_k^4}, \dots$, of strings of length positive powers of p_k. For all $j > 1$, run the computation of $\mathcal{A}lg_k$ on input σ_k for j steps. If during this computation $\mathcal{A}lg_k$ guesses $\langle k, 1\rangle$, then let $U_{\langle k,j\rangle}$ be the set of input strings read by $\mathcal{A}lg_k$ up to the first time it guesses $\langle k, 1\rangle$. If $\mathcal{A}lg_k$ does not guess $\langle k, 1\rangle$ during the first j steps of its computation on σ_k then let $U_{\langle k,j\rangle} = \emptyset$. Define $L^4_{\langle k,j\rangle} = U_{\langle k,j\rangle} \cup \{a^{p_k}\}$

and $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$. Clearly, $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ is an indexed family of nonempty recursive languages.

Lemma 6. *$\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ is in $LimTxt \backslash (ConsvTxt \cup PCorQ)$.*

Proof. Angluin shows in [Ang80b] that $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ is in $LimTxt \backslash ConsvTxt$, so we just need to prove that $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ is not in $PCorQ$. Assume by contrary that it is. Then, since $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ has property $\mathbb{A}$, it follows by Theorem 4 that $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$ is in $ConsvTxt$, a contradiction. $\square$

Next we give an example[4] of an indexed family of nonempty recursive languages in $PCorQ \backslash ConsvTxt$. We use as starting point the class $\mathcal{C}^4 = (L_i^4)_{i\geq 1}$, modifying it in order to become learnable with PCQs. The key idea is to add some evidence to each language L_i^4, $i = \langle k, j\rangle$ that indicates whether $\mathcal{A}lg_k$ on input $\sigma_k = a^{p_k}, a^{p_k^2}, a^{p_k^3}, a^{p_k^4}, \ldots$, guesses $\langle k, 1\rangle$ at least once or not. In what follows, whenever we say that a given inference machine guesses $\langle k, j\rangle$, it is with respect to the indexed family of recursive languages $(L^5_{\langle k,j\rangle})_{k,j\geq 1}$ that is subsequently defined.

Given an IIM $\mathcal{A}lg_k$, one can construct a deterministic Turing machine M_k which accepts the input string b if and only if $\mathcal{A}lg_k$ on σ_k guesses $\langle k, 1\rangle$ at some finite step. Otherwise, M_k will loop forever, thus not accepting the input string b. Clearly, $\mathcal{L}(M_k)$ is not recursive, but $\textsc{Valc}(M_k)$ is (by Corollary 4). Then, let $E_k = \textsc{Valc}(M_k) \cap b\natural\Gamma^*$. By the construction of M_k, it holds that $E_k = \emptyset$ if and only if $\mathcal{A}lg_k$ on input σ_k never guesses $\langle k, 1\rangle$. For all $j, k \geq 1$, define $L^5_{\langle k,j\rangle} = L^4_{\langle k,j\rangle} \cup E_k$. The class $\mathcal{C}^5 = (L_i^5)_{i\geq 1}$ is an indexable class of recursive languages.

Lemma 7. *$\mathcal{C}^5 = (L_i^5)_{i\geq 1}$ is in $PCorQ \backslash ConsvTxt$.*

Proof. To show that $\mathcal{C}^5 = (L_i^5)_{i\geq 1}$ is learnable with PCQs, we consider the following learning procedure. Assume L is the target language. First, get from the oracle the value of $C_L(a^2)$. Clearly, if k is such that $p_k = 2 + |C_L(a^2)|$ we can conclude that the target language should be from $L^5_{\langle k,1\rangle}, L^5_{\langle k,2\rangle}, L^5_{\langle k,3\rangle}, \ldots$ Next, ask a PCQ with $b\natural$. There are two cases to consider.

1. If the answer is θ, we know that $\mathcal{A}lg_k$ never guesses $\langle k, 1\rangle$. So, $L^5_{\langle k,j\rangle}$ with $j > 1$ contains only a^{p_k}. Then, ask a PCQ with the string $a^{p_k^2}$. If the answer is not θ, it means that the target language is $L^5_{\langle k,1\rangle}$ so we can output $\langle k, 1\rangle$. Otherwise, it can be any language in $L^5_{\langle k,2\rangle}, L^5_{\langle k,3\rangle}, \ldots$, but since they are all equal, it is enough to output $\langle k, 2\rangle$.

[4]We are indebted to S. Kobayashi for the construction of the language class $\mathcal{C}^5 = (L_i^5)_{i\geq 1}$ and the proof of Lemma 7.

2. If the answer is not θ, we know that $\mathcal{A}lg_k$ eventually guesses $\langle k,1\rangle$. Then, we run $\mathcal{A}lg_k$ on $\sigma_k = a^{p_k}, a^{p_k^2}, a^{p_k^3}, ...$ and wait for the first time $\mathcal{A}lg_k$ guesses $\langle k,1\rangle$. Let j be the first such step and let $T = \{a^{p_k}, a^{p_k^2}, ..., a^{p_k^l}\}$ be the set of initial segment of strings in σ_k read by $\mathcal{A}lg_k$ up to the step j. Then, we know that:

   - $L^5_{\langle k,1\rangle} = \{a^{p_k}, a^{p_k^2}, a^{p_k^3}, \ldots\} \cup E_k$
   - $L^5_{\langle k,m\rangle} = \{a^{p_k}\} \cup E_k$ for $m = 2, ..., j-1$
   - $L^5_{\langle k,m\rangle} = \{a^{p_k}\} \cup T \cup E_k$ for $m = j, j+1, j+2, ...$

   To differentiate between the three languages, first ask a PCQ with the string $a^{p_k^2}$. If the answer is θ, output $\langle k,2\rangle$. Otherwise, ask a PCQ with $a^{p_k^{l+1}}$. If the answer is θ, output $\langle k,j\rangle$. Otherwise, output $\langle k,1\rangle$.

This algorithm learns the class $\mathcal{C}^5 = (L^5_i)_{i\geq 1}$ with PCQs.

We next show that $\mathcal{C}^5 = (L^5_i)_{i\geq 1}$ is not in *ConsvTxt* using the same argument Angluin employed in the proof of Theorem 4 on page 131-132 [Ang80b]. Consider the computation of an inference machine $\mathcal{A}lg_k$ on $\sigma_k = \{a^{p_k}, a^{p_k^2}, \ldots\}$. There are two cases to consider.

1. If $\mathcal{A}lg_k$ on σ_k never guesses $\langle k,1\rangle$, then $L^4_{\langle k,j\rangle} = \{a^{p_k}\}$ for all $j > 1$. Furthermore, since $E_{\mathcal{A}lg_k} = \emptyset$, $L^5_{\langle k,j\rangle} = L^4_{\langle k,j\rangle}$ holds for every $j \geq 1$. Therefore, σ_k is a positive presentation for $L^5_{\langle k,1\rangle}$. So, we can conclude that $\mathcal{A}lg_k$ fails to infer $L^5_{\langle k,1\rangle}$ from positive data.

2. If $\mathcal{A}lg_k$ on σ_k eventually guesses $\langle k,1\rangle$, then E_k contains a unique string, say $b\natural w$. Let j be the first step at which $\mathcal{A}lg_k$ on σ_k guesses $\langle k,1\rangle$. Let $\hat{\sigma}_k$ be the finite initial segment of σ_k read by $\mathcal{A}lg_k$ up to step j, followed by the unique string $b\natural w$ and an infinite sequence of a^{p_k}'s. Then, $\hat{\sigma}_k$ is a positive presentation of $L^5_{\langle k,j\rangle}$. Consider $\mathcal{A}lg_k$ on input $\hat{\sigma}_k$. We know that at step j, $\mathcal{A}lg_k$ guesses $\langle k,1\rangle$. If $\mathcal{A}lg_k$ never subsequently changes its guess, it fails to infer $L^5_{\langle k,j\rangle}$ from positive data. On the other hand, if $\mathcal{A}lg_k$ subsequently changes its guess, it fails to be conservative on $L^5_{\langle k,1\rangle}, L^5_{\langle k,2\rangle}, \ldots$, because $L^5_{\langle k,1\rangle}$ is consistent with every initial segment of $\hat{\sigma}_k$.

Thus in either case $\mathcal{A}lg_k$ must either fail to infer $L^5_{\langle k,1\rangle}, L^5_{\langle k,2\rangle}, \ldots$ from positive data or fail to be conservative on $L^5_{\langle k,1\rangle}, L^5_{\langle k,2\rangle}, \ldots$. Hence, the family $(L^5_{\langle k,j\rangle})_{k,j\geq 1}$ is an indexed family of nonempty recursive languages such that no inference machine can both infer the family from positive data and be conservative on it. Therefore, there is no conservative IIM that learns $\mathcal{C}^5 = (L^5_i)_{i\geq 1}$ from text. $\square$

3.1.4 An Example: Learning k-Reversible Languages with Prefix Correction Queries

The conditions introduced in Section 3.1.1 can be used not only to show the relation of our model with other well-known learning models, but they are useful as well in proving that a given class is learnable with PCQs (or the contrary, for that matter). Let us consider for example the class of k-reversible languages introduced by Angluin back in '82, and shown to be inferable from positive data in the limit [Ang82]. We will see that for any k-reversible language L, a triple of definite finite tell-tales of L is uniformly computable and hence, by Proposition 3, *k-Rev* is in *PCorQ*. Note that the class *k-Rev* is not learnable with MQs, as we will show later in this dissertation (see Theorem 14, Section 4.1.3).

We first state and prove the following observation.

Lemma 8. *If L is a k-reversible language and u_1, u_2 are arbitrary strings in Σ^*, then $u_1 \equiv_L u_2$ if and only if $C_L(u_1v) = C_L(u_2v)$ for all v in $\Sigma^{\leq k}$.*

Proof. Note that for all regular languages L and for any $v \in \Sigma^*$, $u_1 \equiv_L u_2$ implies $u_1v \equiv_L u_2v$, so $C_L(u_1v) = C_L(u_2v)$ is trivially true. Thus, we just have to show that if $C_L(u_1v) = C_L(u_2v)$ for all $v \in \Sigma^{\leq k}$, then $u_1 \equiv_L u_2$.

Indeed, assume by contrary that there are strings u_1, u_2 in Σ^* such that $u_1 \not\equiv_L u_2$ and $C_L(u_1v) = C_L(u_2v)$ for all v in $\Sigma^{\leq k}$. Hence, there must exist $w \in \Sigma^*$ 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 hence $C_L(u_1w) = C_L(u_2w)$. But $u_1w \in L$ implies $C_L(u_1w) = \lambda$, and so $C_L(u_2w) = \lambda$ which contradicts $u_2w \notin L$.
2. If $|w| > k$, then there must exist $v, w' \in \Sigma^*$ such that $w = vw'$ and $|v| = k$. By assumption, $u_1vw' \in L$ and $u_2vw' \notin L$, so $u_1v \not\equiv_L u_2v$. On the other hand, since $v \in \Sigma^{\leq k}$ we have $C_L(u_1v) = C_L(u_2v) = v'$. Because $u_1v \cdot w' \in L$, $Tail_L(u_1v) \neq \emptyset$ and hence $C_L(u_1v) \in \Sigma^*$. Since L is k-reversible, $u_1vv' \in L, u_2vv' \in L$ and $|v| = k$, we get $Tail_L(u_1v) = Tail_L(u_2v)$ (by Theorem 1) which contradicts $u_1v \not\equiv_L u_2v$.

We showed that if $C_L(u_1v) = C_L(u_2v)$ for all v in $\Sigma^{\leq k}$, then $u_1 \equiv_L u_2$ which concludes our proof. □

Let L be a k-reversible language over the alphabet Σ. We denote by $\tilde{L}$ the set $\{w \mid \forall v \in [w]_L, w \preceq v\}$ containing the smallest elements in each equivalence class with respect to $\equiv_L$. Clearly, $\tilde{L}$ has exactly n elements where n is the index of L, and it is computable in polynomial time for any regular language. The following algorithm takes as input a k-reversible language and outputs a triple of definite finite tell-tales for it.

Algorithm 4 Algorithm for computing a triple of definite finite tell-tales

```
1: Input: the target language $L_0$
2: $T_0 := \emptyset, F_0 := \emptyset, U_0 := \emptyset$
3: for all $u$ in $\tilde{L}_0\Sigma^{\leq k+1}$ do
4:   if $(C_{L_0}(u) = \lambda)$ then
5:     add $u$ to $T_0$
6:   else
7:     if $(C_{L_0}(u) = \Theta)$ then
8:       add $u$ to $U_0$
9:     else
10:      add $u \cdot C_{L_0}(u)$ to $T_0$
11:      for all $v \prec C_{L_0}(u)$ do
12:        add $uv$ to $F_0$
13:      end for
14:    end if
15:  end if
16: end for
```

We show that the triple $\langle T_0, F_0, U_0\rangle$ computed by Algorithm 4 is indeed a triple of definite finite tell-tales of L_0. For this, we need several lemmas.

Lemma 9. *Let L be a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$. For any u in $\tilde{L}_0\Sigma^{\leq k+1}$, $C_{L_0}(u) = C_L(u)$.*

Proof. Let u be an arbitrary string in $\tilde{L}_0\Sigma^{\leq k+1}$. Since L is consistent with $\langle T_0, F_0, U_0\rangle$, we have $T_0 \subseteq L$, $F_0 \subseteq \Sigma^*\backslash L$ and $U_0 \subseteq \Sigma^*\backslash Pref(L)$.

One of the following three possible situations may occur.

- $C_{L_0}(u) = \lambda$, so $u \in T_0$ (added to T_0 at line 5). But $T_0 \subseteq L$, and hence $u \in L$. We obtain $C_L(u) = \lambda = C_{L_0}(u)$.
- $C_{L_0}(u) = \Theta$, so $u \in U_0$ (added to U_0 at line 8). But $U_0 \subseteq \Sigma^*\backslash Pref(L)$, and hence $u \notin Pref(L)$. We obtain $C_L(u) = \Theta = C_{L_0}(u)$.
- $C_{L_0}(u) = v \in \Sigma^+$, so $uv \in T_0$ and $uv' \in F_0$ for all $v' \prec v$. Thus, $uv \in L$ and $uv' \in \Sigma^*\backslash L$ for all $v' \prec v$. We obtain $C_L(u) = v = C_{L_0}(u)$.

So in all three cases, $C_L(u) = C_{L_0}(u)$ which concludes our proof. □

Lemma 10. *Let L be a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$. If $u_1, u_2 \in \tilde{L}_0$ are such that $[u_1a]_{L_0} = [u_2]_{L_0}$ for some $a \in \Sigma$, then $[u_1a]_L = [u_2]_L$.*

Proof. Let u_1, u_2 be two elements of $\tilde{L}_0$ such that $[u_1a]_{L_0} = [u_2]_{L_0}$ for some $a \in \Sigma$. We want to show that $[u_1a]_L = [u_2]_L$, that is, $C_L(u_1av) = C_L(u_2v)$ for all $v \in \Sigma^{\leq k}$ (by Lemma 8). So, let us choose v in $\Sigma^{\leq k}$ arbitrarily. Since both u_1av and u_2v are in $\tilde{L}_0\Sigma^{\leq k+1}$, $C_{L_0}(u_1av) = C_L(u_1av)$ and $C_{L_0}(u_2v) = C_L(u_2v)$ (by Lemma 9). Keeping in mind that $[u_1a]_{L_0} = [u_2]_{L_0}$ and $v \in \Sigma^{\leq k}$, we obtain $C_{L_0}(u_1av) = C_{L_0}(u_2v)$, and hence $C_L(u_1av) = C_L(u_2v)$. □

Lemma 11. *If L is a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$, then the index of L is greater than or equal to the index of L_0.*

Proof. Let u_1, u_2 be two distinct elements of $\tilde{L}_0$. Since $u_1 \not\equiv_{L_0} u_2$, there must exist a v in $\Sigma^{\leq k}$ such that $C_{L_0}(u_1v) \neq C_{L_0}(u_2v)$ (by Lemma 8). Because both u_1v and u_2v are in $\tilde{L}_0\Sigma^{k+1}$, we get $C_{L_0}(u_1v) = C_L(u_1v)$ and $C_{L_0}(u_2v) = C_L(u_2v)$ (by Lemma 9). This implies $C_L(u_1v) \neq C_L(u_2v)$, and hence $u_1 \not\equiv_L u_2$ (again by Lemma 8). So, L has at least as many equivalence classes as the cardinality of the set $\tilde{L}_0$, which means that the index of L is greater than or equal to the index of L_0. □

For any regular language L over Σ and any string $u \in \Sigma^*$, we denote by $\tilde{u}_L$ the unique element $v \in \tilde{L}$ such that $[u]_L = [v]_L$. The following two lemmas hold[5].

Lemma 12. *If L is a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$, then $[u]_L = [\tilde{u}_{L_0}]_L$ for any $u \in \Sigma^*$.*

Proof. Let us proceed by induction on the length of u. In case of $|u| = 0$, the claim holds clearly. Assume that the claim holds for the case of $|u| \leq l$, and consider the case of $|u| = l + 1$. We can write $u = va$ for some $v \in \Sigma^l$ and $a \in \Sigma$. Then by induction hypothesis, we have $[v]_L = [\tilde{v}_{L_0}]_L$, and therefore $[va]_L = [\tilde{v}_{L_0}a]_L$ holds. Note that $\tilde{u}_{L_0}, \tilde{v}_{L_0} \in \tilde{L}_0$ and $[\tilde{v}_{L_0}a]_{L_0} = [\tilde{u}_{L_0}]_{L_0}$, which implies, by Lemma 10, that $[\tilde{v}_{L_0}a]_L = [\tilde{u}_{L_0}]_L$. Finally, we have $[u]_L = [va]_L = [\tilde{v}_{L_0}a]_L = [\tilde{u}_{L_0}]_L$. □

Lemma 13. *If L is a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$, then for any $u, v \in \Sigma^*$, $[u]_{L_0} = [v]_{L_0}$ implies $[u]_L = [v]_L$.*

Proof. Assume $[u]_{L_0} = [v]_{L_0}$. Since $[u]_{L_0} = [\tilde{u}_{L_0}]_{L_0}$ and $[v]_{L_0} = [\tilde{v}_{L_0}]_{L_0}$, we immediately get that $[\tilde{u}_{L_0}]_{L_0} = [\tilde{v}_{L_0}]_{L_0}$, and hence $\tilde{u}_{L_0}$ and $\tilde{v}_{L_0}$ must coincide (the set $\tilde{L}_0$ contains only one representative for each equivalence class). By Lemma 12, we have $[u]_L = [\tilde{u}_{L_0}]_L$ and $[v]_L = [\tilde{v}_{L_0}]_L$. Therefore, $[u]_L = [\tilde{u}_{L_0}]_L = [\tilde{v}_{L_0}]_L = [v]_L$ holds. □

[5]Many thanks to S. Kobayashi for introducing the notation $\tilde{u}_L$, thus simplifying the proofs of Lemma 12 and Lemma 13.

Theorem 8. *If L is a k-reversible language consistent with $\langle T_0, F_0, U_0\rangle$, then $L_0 = L$ holds.*

Proof. Lemma 13 implies that $\equiv_L$ is coarser than $\equiv_{L_0}$. Thus by Lemma 11, we can conclude that $\equiv_L$ is equivalent to $\equiv_{L_0}$ which implies that $\tilde{w}_{L_0} = \tilde{w}_L$ for any $w \in \Sigma^*$. Therefore, we have $w \in L_0$ if and only if $C_{L_0}(w) = \lambda$ if and only if $C_{L_0}(\tilde{w}_{L_0}) = \lambda$ if and only if $C_L(\tilde{w}_{L_0}) = \lambda$ (by Lemma 9) if and only if $C_L(\tilde{w}_L) = \lambda$ if and only if $C_L(w) = \lambda$ if and only if $w \in L$. □

We would like to stress here that later in this monograph (Chapter 4, Section 4.1.2) we prove a stronger result, namely that k-reversible languages are polynomial time learnable with PCQs. However, we decided to include the proof of its learnability in the general setting (where there are no constraints regarding time complexity) for two reasons. First of all, we consider that this proof is an exemplification of how theoretical results may arise from using the conditions described in Section 3.1.1. Secondly, we might not even have thought of finding an algorithm for the class *k-Rev* unless we knew that it is in *PCorQ*.

3.2 Learning with Length Bounded Correction Queries

Let us recall that the particular choice we have made for the prefix correcting string is closely related to the intrinsic structure of DFAs and their properties. One may argue though that the smallest correcting string might be arbitrarily long. For example, if we take n to be any natural number, then there exists a string w in Σ^* and a (regular) language L such that the correction of w with respect to L is longer than n (take $w = \lambda$ and $L = \{a^{n+1}\}$ where a is an arbitrary symbol in Σ). So, let us see what happens when, instead of returning the smallest string in $Tail_L(w)$, we return all tails shorter than a given fixed natural number. Clearly, if there is no possible short correction, the oracle will return the empty set.

Let us fix an integer l. Given a language L and a string w, we define the *l-bounded correction of w with respect to L* (denoted $C^l_L(w)$) as the set of all strings v of length at most l such that $w \cdot v$ is in L. Formally, C^l_L is a function from Σ^* to $\mathcal{P}(\Sigma^*)$ such that for any w in Σ^*, $C^l_L(w) = \{v \in Tail_L(w) \mid |v| \leq l\}$. Note that $\lambda \in C^l_L(w)$ if and only if $w \in L$.

So, let us consider the model in which the learner must identify the target language after asking a finite number of *l-bounded correction queries* (lBCQs). Since any 0BCQ can be simulated by an MQ and the other way around, it is clear that for $l = 0$, learning with lBCQs is equivalent to learning with MQs. It is though less straightforward that the same property holds for an arbitrary l. We show in the sequel that for any l, a language class is learnable with MQs if and only if it is learnable with lBCQs. So let us denote by *lBCorQ* the collection of all indexable classes $\mathcal{C}$ for which there exists a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using lBCQs. The following result holds.

Proposition 5. *For any $l \geq 0$, lBCorQ = MemQ.*

Proof. Since for any language L and any string w in Σ^*, if we know the answer to $C_L^l(w)$ we also know if the string w is in L or not, it is clear that *lBCorQ* includes *MemQ*. Hence, we have to show only that $lBCorQ \subseteq MemQ$.

Let us consider a language class $\mathcal{C}$ in *lBCorQ*, and let $\mathcal{A}lg$ be a query learner such that $\mathcal{A}lg$ learns $\mathcal{C}$ using lBCQs. We modify $\mathcal{A}lg$ such that instead of submitting an lBCQ for a given string w and the target language L, it submits MQs for all the strings wu with $u \in \Sigma^{\leq l}$, and uses this information to construct the answer for $C_L^l(w)$ (i.e., $C_L^l(w) := \{u \in \Sigma^{\leq l} \mid wu \in L\}$). Clearly, this modified version of $\mathcal{A}lg$ is a query learning algorithm that learns $\mathcal{C}$ using MQs. □

This proposition is basically saying that having an oracle that can return at once the answers for more than one MQ (one lBCQ contains the answer for $1 + |\Sigma| + \ldots + |\Sigma|^l$ MQs) does not increase the learnability power of the model (that is, the learning with MQs model). The result was somehow expected if we recall that time complexity issues are neglected in our analysis. Moreover, this allows us to talk about the model of learning with *length bounded correction queries* (LBCQs) in general, without specifying a given length bound. Therefore, we denote by *LBCorQ* the collection of all language classes $\mathcal{C}$ for which there exists an $l \geq 0$ and a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using a finite number of lBCQs.

Combining the results of Proposition 5 and Corollary 2, we get the following characterization of our model.

Corollary 6. *The class $\mathcal{C} = (L_i)_{i \geq 1}$ is learnable with LBCQs if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any index i.*

3.3 Learning with Edit Distance Based Correction Queries

The idea of using corrections as an alternative to MQs has been enthusiastically embraced by other members of the grammatical inference community [BBdlHJT07, JK07, Kin08]. While in [JK07] the correction of a string not in the target language is defined as being the *nearest* positive example, where the distance is the usual distance between numbers[6], both [BBdlHJT07] and [Kin08] define the correcting string with respect to the edit distance.

Since in [Kin08] the correction returned by the teacher heavily depends on the algorithm chosen (on the sequence of queries previously asked by the learner, to be precise), we chose not to include here a discussion about this model. We will revisit this topic with details in Chapter 4, Section 4.3.

So, let us turn first to the corrections proposed by Jain and Kinber [JK07]. The authors introduce two models:

- in the first one, the learner receives the nearest positive example if the submitted element is not in the target language; we refer to this model as *NPMemQ*.
- in the second one, the nearest positive example not exceeding the size of the negative example (i.e., an element submitted by the learner which is not in the target language) is provided (if any); this second model is motivated by the fact that in the first approach *a teacher may have difficulties providing the nearest (correcting) positive example, as it can still be too complex - far larger than the negative example* [JK07]. We refer to this model as *BNPMemQ* (*B* stands for *bounded*).

Notice that in all those models the learner is allowed to query only elements of $\bigcup_{L \in \mathcal{C}} L$ - we indicate that by adding the prefix *R* (for *restricted*) in front of *MemQ*. The authors of [JK07] point out that if one allows membership queries for any member of $\mathbb{N}$, the model - including the cases for the (bounded) nearest positive examples - would collapse to learning from informant (that is, $NPMemQ = BNPMemQ = MemQ$). In Figure 3.1 the results of Theorems 8, 9 and 10 from the same paper are summarized.

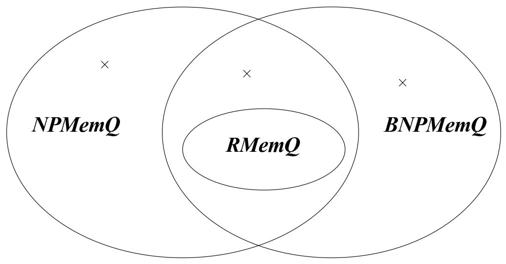


Figure 3.1: Learning with nearest positive example

We detail these results in the sequel, adapting them to our terminology (and using as well Theorem 1 of the same paper [JK07]).

[6]The languages in their model are subsets of $\mathbb{N}$.

Theorem 9 (Jain, Kinber [JK07]). *$NPMemQ \backslash BNPMemQ \neq \emptyset$.*

Theorem 10 (Jain, Kinber [JK07]). *$BNPMemQ \backslash NPMemQ \neq \emptyset$.*

Theorem 11 (Jain, Kinber [JK07]). *$(NPMemQ \cap BNPMemQ) \backslash RMemQ \neq \emptyset$.*

In other words, when the learner is not allowed to ask arbitrary MQs, learning with nearest positive example is strictly more powerful than learning with MQs, even if the length of the example is bounded. We show that this is no longer the case when the learner can query any string in Σ^*, and the distance between strings is the edit distance. Even though our study does not directly take into account the case where the length of the nearest positive example is bounded by the negative one, the results can be easily extended to cover it.

Following [BBdlHJT07], we define the *edit distance correction* of a string w with respect to the language L by:

$$\textsc{Edc}_L(w) = \begin{cases} \texttt{Yes}, & \text{if } w \in L, \text{ and} \\ \text{one string of } \{w' \in L \mid d(w, w') \text{ is minimum}\}, & \text{if } w \notin L. \end{cases}$$

Similarly, we denote by $\textsc{Mq}_L(w)$ the oracle's answer when it is queried with the string w for the target language L. That is, $\textsc{Mq}_L(w) = \texttt{Yes}$ if $w \in L$, and `No` otherwise.

Note that $\textsc{Edc}_L(w) = \texttt{Yes}$ if and only if w is in L. Clearly, any oracle answering *edit distance correction queries* (EDCQs) would also give us the answer for the corresponding MQ. If we denote by *EditCorQ* the collection of all indexable classes $\mathcal{C}$ for which there exists a query learner $\mathcal{A}lg$ such that $\mathcal{A}lg$ learns $\mathcal{C}$ using edit distance correction queries, it is clear that *EditCorQ* includes *MemQ*. The following equality holds.

Theorem 12. *$EditCorQ = MemQ$.*

Proof. Let us first show that having an MQ oracle allows us to compute the value of $\textsc{Edc}_L(w)$ for any language $L \neq \emptyset \subseteq \Sigma^*$ and any w in Σ^* using a finite number of MQs. So, let us assume that we have a target language L and the learner submits the string w to the oracle. Algorithm 5 computes the value of $\textsc{Edc}_L(w)$ by asking only MQs.

If L is not empty, then the algorithm terminates by outputting a string $u \in L$ such that there is no $v \in L$ with $d(w, v) < d(w, u)$. Note that for a given $w \in \Sigma^*$ and $r \in \mathbb{N}$ there are only a finite number of strings $v \in \Sigma^*$ such that $d(w, v) = r$, and there exists an algorithm who can generate all these strings (remember that we are not concerned with the complexity of the resulting algorithm - the only requirement is to return the answer after finite steps).

Algorithm 5 An algorithm that computes $\text{EDC}_L(w)$ with an MQ oracle

```
1: input: $L, w$
2: if $\text{MQ}_L(w) = \texttt{Yes}$ then
3:   output Yes
4: else
5:   while TRUE do
6:     $i := 1$
7:     for all $u$ such that $d(w,u) = i$ do
8:       if $\text{MQ}_L(u) = \texttt{Yes}$ then
9:         output $u$ and halt
10:      end if
11:    end for
12:    $i := i + 1$
13:  end while
14: end if
```

Now, if we take $\mathcal{C}$ to be a language class in *EditCorQ*, then there exists an algorithm $\mathcal{A}lg$ that learns $\mathcal{C}$ using EDCQs. We can modify $\mathcal{A}lg$ to use the MQ oracle to get the answers for the EDCQs as described above. We obtained an algorithm that learns $\mathcal{C}$ using MQs only, so *EditCorQ* $\subseteq$ *MemQ* which concludes our proof. □

We can now use Theorem 12 and Corollary 2 to give a characterization of the model of learning with EDCQs in terms of finite sets.

Corollary 7. *The class $\mathcal{C} = (L_i)_{i\geq 1}$ belongs to EditCorQ if and only if a pair of definite finite tell-tales of L_i is uniformly computable for any index i.*

3.4 Remarks and Further Research

Chapter 3 was dedicated to the learnability power of several types of CQs used alone when complexity issues are neglected. A complete picture displaying the relations between all discussed versions of query learning and Gold-style learning is obtained (Figure 3.2).

Our results can be summarized as follows:

- learning with LBCQs and learning with EDCQs are as powerful as learning with MQs;
- learning with PCQs is strictly more powerful than learning with MQs, and strictly less powerful than learning in the limit from text;
- the sets *PCorQ* and *ConsvTxt* are incomparable, but any class which is in *PCorQ* and not in *ConsvTxt* has the following property: at least one of the languages in the class has the prefix not recursive.

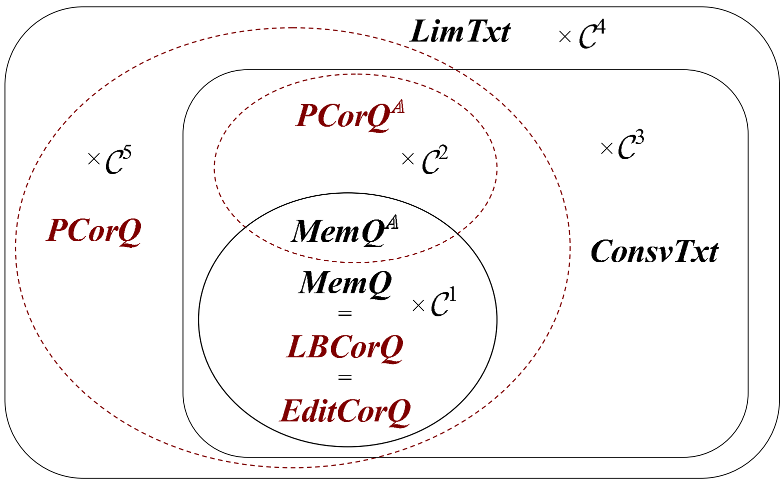


Figure 3.2: The hierarchy of Gold-style and query learning models

There are several directions that deserve further investigation. For example, one may study language learning with CQs in other standard settings like *bounded learning* [Wat94] or *incremental learning* (see [LG02] and the references therein), or completely new ones: learning from positive examples and CQs, or learning with CQs and (a bounded number of) EQs.

Chapter 4

Polynomial Time Learning with Correction Queries

In the previous chapter we have investigated the power of the query learning model when an oracle answering correction queries is available. Several types of CQs have been considered, and for each of them a characterization in terms of finite sets has been given. Moreover, we compared the newly introduced models with other well-known query learning and Gold-style learning models.

Although from a theoretical point of view it is important to know what language classes are inferable in finite time steps, what matters in practice is the efficiency of the algorithms. And indeed, if we need a machine capable of learning a given language class, it does not help us too much if we know that we will get the answer in 153 years. That is why this chapter is dedicated to polynomial time query learners that have access to CQ oracles. We show that there are several nontrivial language classes which are polynomial time learnable with CQs, and we investigate the relations existing between the different types of CQs introduced so far when complexity issues are taken into consideration. Moreover, we distinguish situations when although two query types are equally powerful in the general case, the equality is not preserved under efficiency constraints.

The reader may notice that in this chapter we switch from learning languages to learning grammars. Why? First of all, because in practice what we usually want to learn is a grammar, and not a sequence of words, whereas from a theoretical point of view it is less important if a given language class is learnable with respect to a specific (and hence restrictive) hypotheses space. Secondly, if we talk about efficient algorithms, then one needs to define first what *polynomial time learning* is, and hence a reasonable measure for the size of a language is required. Note that for infinite languages, having a polynomial algorithm in the number of elements of the language does not make much sense. Moreover, any grammar is a compact way of describing the language generated by it. That is why from now on, by the *size of a language* we understand the size of the smallest grammar (from a given hypotheses space) generating it.

4.1 Polynomial Time Learning with Prefix Correction Queries

Let us start our study with the type of CQ that was chronologically the first one introduced, and that gives, in the same time, the biggest power to the learner amongst all of them. Intuitively, one should be looking for language classes for which this sort of corrections offers some information about the intrinsic structure of the language. We investigate two well-know language classes: the class of pattern languages and the class of k-reversible languages.

4.1.1 The Class of Pattern Languages

Initially introduced by Angluin [Ang79] to show that there are nontrivial 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 [MK87], or just superset queries [Ang88], or for learning k-variables pattern languages from examples [KP89], etc.

Recall that by $\mathbb{P}$ we denote the class of all pattern languages over a fixed alphabet Σ. We exhibit an algorithm that learns any pattern language in polynomial time using a finite number of PCQs.

The Algorithm

Suppose that the target language is a pattern language $L(\pi)$, where π is in normal form. Our algorithm is based on the following simple observations.

If w is the smallest string (in lex-length order) in $L(\pi)$ and $n = |w|$, then for all i in $\{1, \ldots, n\}$, we have:

- if $\pi[i] = a$ for some a in Σ, then for all $b \in \Sigma \backslash \{a\}$, $C_L(w[1 \ldots i-1]b)$ is either Θ, or longer than $w[i+1 \ldots n]$.
- if $\pi[i]$ is a variable x such that i is the position of the leftmost occurrence of x in π, then $|C_L(w[1 \ldots i-1]a)| = |w[i+1 \ldots n]|$ 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 $C_L(w[1 \ldots i-1]a)$ and $w[i+1 \ldots n]$ do not coincide, where a is any symbol in $\Sigma \backslash \{w[i]\}$;

The following algorithm outputs the pattern π after asking a finite number of PCQs (recall that, by convention, the length of the symbol Θ is 0).

Algorithm 6 An algorithm for learning the class $\mathbb{P}$ with PCQs

```
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:         $v := C_L(w[1 \ldots i-1]a), m := |v|$
9:         if $(v = \Theta$ or $m > n - i)$ then
10:            $\pi[i] := w[i]$
11:        else
12:            $var := var + 1, \pi[i] := x_{var}$
13:            for all $j \in \{1, \ldots, m\}$ for which $v[j] \neq w[i+j]$ do
14:                $\pi[i+j] := x_{var}$
15:            end for
16:        end if
17:    end if
18: end for
19: output $\pi$
```

From the above mentioned observations, it is clear that the algorithm terminates in finite steps with a correct output. Moreover, 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 (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 $n+1$ PCQs.

Running Example

Let us see how the algorithm works on an example. Assume that the target pattern is $1x0xy$ and the alphabet is $\Sigma = \{0, 1\}$. The algorithm starts by asking a PCQ for the string λ. The length of the returned string (the shortest one in the language) gives us the size of the pattern. In our case, the smallest string in the language is 10000, so w gets the value 10000, n is initialized with 5, and var becomes 0 (i.e., we did not identify any variable yet). Moreover, all the elements of π become $null$, meaning "they are not yet known".

Next, the algorithm enters in the **for** loop (lines 5-18).

- $i = 1$. Since $\pi(i)$ is obviously *null*, the algorithm continues by choosing an a in $\Sigma \backslash \{w[1]\}$. The only possibility in this case is $a = 0$. On line 8, a PCQ is asked for the string $w[1 \ldots i-1]a$, that is, for 0. Because none of the strings in the target language starts with the letter 0, the answer returned by the teacher is Θ, so v becomes Θ and $m = 0$. Because the condition $v = \Theta$ holds, $\pi[1]$ is initialized with 1 (line 10).

i	a	$v = C_L(0)$	$w[2 \ldots 5]$	π				
1	0	Θ	0000	1	*null*	*null*	*null*	*null*

- $i = 2$. Since $\pi[2]$ is not defined yet, the algorithm chooses an a different from $w[2]$ (i.e., $a = 1$). Then, it gives to v the value of the correction of the string $w[1]a = 11$ (i.e., $v = 010$) and m becomes 3. One can easily check that $m = n - i$, so $\pi[2]$ becomes x_1. Moreover, because $v[2] \neq w[4]$, $\pi[4]$ also takes the value x_1.

i	a	$v = C_L(11)$	$w[3 \ldots 5]$	π				
2	1	010	000	1	x_1	*null*	x_1	*null*

- $i = 3$. $\pi[3]$ is still not known, so a takes the value 1. Since the correction of the string 101 is 0010, $v = 0010$ and $m = 4$. Moreover, because $m > n - i$, the algorithm proceeds by initializing $\pi[3]$ with the value $w[3] = 0$.

i	a	$v = C_L(101)$	$w[45]$	π				
3	1	0010	00	1	x_1	0	x_1	*null*

- $i = 4$. The algorithm already knows that the forth symbol of the pattern is a variable ($\pi[4] \neq null$), namely the same variable that is on the second position of the pattern.

- $i = 5$. The algorithm has to identify now the fifth symbol of the pattern. First of all, a is initialized with the letter 1, and then, the teacher is asked to provide a correction for the string 10001. But 10001 clearly belongs to the target language, so $v = \lambda$ and $m = 0$. Moreover, since $m = n - i$, *var* becomes 2 and $\pi[5]$ is initialized with x_2.

The algorithm terminates by outputting the pattern $1x_10x_1x_2$, which is exactly the target one, up to a renaming of the variables.

4.1.2 The Class of k-Reversible Languages

Angluin introduces the class of k-reversible languages (henceforth denoted by *k-Rev*) in [Ang82], 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 [Ang90].

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

In order to do so, we exhibit one important property of k-reversible languages. Let us fix the alphabet Σ and the k-reversible language $L \subseteq \Sigma^*$. For any string u in Σ^*, we define the function $row_k(u) : \Sigma^{\leq k} \to \Sigma^* \cup \{\Theta\}$ by $row_k(u)(v) = C_L(uv)$. Recall that Lemma 8 from Section 3.1.4 of the previous chapter is stating that given a k-reversible language L, two strings u_1 and u_2 are equivalent with respect to the language L if and only if $C_L(u_1v) = C_L(u_2v)$ for all v in $\Sigma^{\leq k}$. This means that each equivalence class in $\Sigma^*/_{\equiv_L}$ is uniquely identified by the values of function row_k on $\Sigma^{\leq k}$.

Proposition 6. *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 $row_k(u_1) = row_k(u_2)$.*

This result tells us that if $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ is the minimal complete DFA for the k-reversible language L, then the values of function $row_k(u)$ on $\Sigma^{\leq k}$ uniquely identify the state $\delta(q_0, u)$.

The Algorithm

The algorithm follows the lines of L*. We have an *observation table* denoted by $(S, \Sigma^{\leq k}, C)$ in which lines are indexed by the elements of a prefix-closed set S, columns are indexed by the elements of $\Sigma^{\leq k}$, 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 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 number of columns of the table is never modified during the run of the algorithm (in L*, there is only one column in the beginning, and then more columns are gradually added when needed).

We say that the observation table $(S, \Sigma^{\leq k}, C)$ is *k-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, $(S, \Sigma^{\leq k}, C)$ is *k-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 $(S, \Sigma^{\leq k}, C)$ is k-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 k-closed and k-consistent table $(S, \Sigma^{\leq k}, C)$, we construct the automaton $\mathcal{A}(S, \Sigma^{\leq k}, C) = (Q, \Sigma, \delta, q_0, F)$ 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 nonempty prefix-closed set, it must contain λ, so q_0 is defined. Because S is k-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 $(S, \Sigma^{\leq k}, C)$ is k-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 k-consistent, this u' is unique. So δ is well defined.

Remark 5. *The following statements hold.*

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 an algorithm that learns any k-reversible language L with PCQs and has a total running time bounded by a polynomial in the size of the target language.

Algorithm 7 An algorithm for learning the class *k-Rev* with PCQs

```
1: $S := \{\lambda\}$
2: $closed$ := TRUE
3: update the table by asking PCQs for strings in $\Sigma^{\leq k+1}$
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 PCQs for strings in $\{uabv \mid b \in \Sigma, v \in \Sigma^{\leq k}\}$
8:       $closed$ := FALSE
9:    end if
10: until $closed$
11: output $\mathcal{A}(S, \Sigma^{\leq k}, C)$ and halt.
```

Note that since the algorithm adds to S only elements with distinct row values, the table $(S, \Sigma^{\leq k}, C)$ is always k-consistent. We will see that as long as $|S| < n$, it is not k-closed.

Lemma 14. *If $|S| < n$, then $(S, \Sigma^{\leq k}, C)$ is not k-closed.*

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

We define the function $\phi : Q \to Q'$ by $\phi(row_k(u)) = \delta'(q_0', u)$. 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 $\phi(row_k(u_1)) = \phi(row_k(u_2))$ implies $\delta'(q_0', u_1) = \delta'(q_0', u_2)$ which is equivalent to $u_1 \equiv_L u_2$, and furthermore to $row_k(u_1) = row_k(u_2)$ (by Proposition 6). We show that ϕ is an automata morphism from $\mathcal{A}(S, \Sigma^{\leq k}, C)$ to $\mathcal{A}_L$, that is: $\phi(q_0) = q_0'$, $\phi(F) \subseteq F'$, and $\phi(\delta(row_k(u), a)) = \delta'(\phi(row_k(u)), a)$ for all $u \in S$ and $a \in \Sigma$.

Clearly, $\phi(q_0) = \phi(row_k(\lambda)) = \delta'(q_0', \lambda) = q_0'$. Let us now take $row_k(u)$ in F, that is, $u \in S$ and $C_L(u) = \lambda$. Since $\phi(row_k(u)) = \delta'(q_0', u)$ and u is in L, it follows that $\phi(row_k(u)) \in F'$. Finally, $\phi(\delta(row_k(u), a)) = \phi(row_k(ua))$ $= \phi(row_k(v))$ for some v in S such that $row_k(ua) = row_k(v)$ (the table is k-closed), and $\delta'(\phi(row_k(u)), a) = \delta'(\delta'(q_0', u), a) = \delta'(q_0', ua)$. It is enough to see that $\phi(row_k(v)) = \delta'(q_0', v) = \delta'(q_0', ua)$ (because by Proposition 6, $row_k(v) = row_k(ua)$ implies $v \equiv_L ua$, and $\mathcal{A}_L$ is the minimal automaton accepting L) to conclude the proof.

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

Algorithm 7 does not work in general for arbitrary regular languages, as one can see from Example 8.

Example 8. Let us fix $k \geq 0$, and consider the finite (and hence regular) language $L_k = \{ab^ka, ab^kb, b^{k+1}a\}$. The minimal complete DFA for L_k is represented in Figure 4.1.

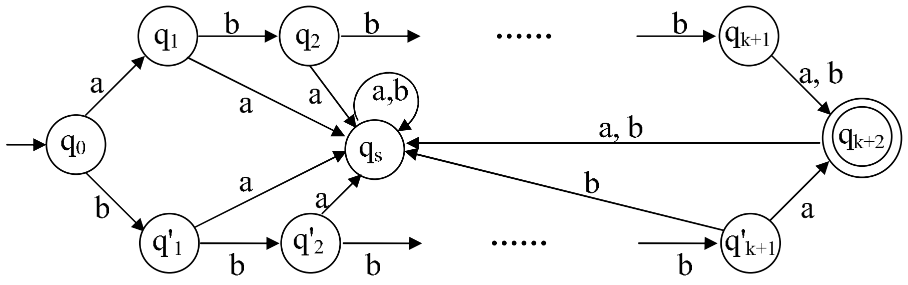


Figure 4.1: The minimal DFA for the language $L_k = \{ab^ka, ab^kb, b^{k+1}a\}$

By Proposition 1 (see Section 2.1.4) the language L_k is not k-reversible since the strings $a \cdot b^k \cdot a$ and $b \cdot b^k \cdot a$ are both in L_k, and $\mathit{Tail}_{L_k}(a \cdot b^k) = \{a, b\} \neq \{a\} = \mathit{Tail}_{L_k}(b \cdot b^k)$.

When running the algorithm on L_k, the set S is initialized with the value $\{\lambda\}$. Then, since both $\mathit{row}_k(a)$ and $\mathit{row}_k(b)$ are different from $\mathit{row}_k(\lambda)$, one of the two elements is added to S. Note that for all u in $\Sigma^{\leq k}$, $\mathit{row}_k(a)(u) = \mathit{row}_k(b)(u)$ 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 $u \in \Sigma^{\leq k} \backslash \{b^i \mid 0 \leq i \leq k\}$, then $C_{L_k}(au) = \Theta = C_{L_k}(bu)$.

Hence, $\mathit{row}_k(a) = \mathit{row}_k(b)$. But this implies that in the automaton output by the algorithm, the strings a and b represent the same state, a contradiction.

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

We have seen that as long as $|S| < n$, the table is not k-closed, so there will always be an u in S and a symbol a in Σ such that $\mathit{row}_k(ua) \notin \mathit{row}_k(S)$. 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 $(S, \Sigma^{\leq k}, C)$ is k-closed and k-consistent, and $\mathcal{A}(S, \Sigma^{\leq k}, C)$ is isomorphic to $\mathcal{A}_L$. Indeed if $|S| = n$, then the set $\{\mathit{row}_k(u) \mid u \in S\}$ has cardinality n, since the elements of S have distinct row values. Thus for all $u \in S$ and $a \in \Sigma$, $\mathit{row}_k(ua) \in \mathit{row}_k(S)$ (otherwise $[ua]_L$ would be the $(n+1)^{\text{th}}$ equivalence class of $\Sigma^*/_{\equiv_L}$), and hence the table is k-closed.

To see that $\mathcal{A}(S, \Sigma^{\leq k}, C)$ and $\mathcal{A}_L$ are isomorphic, let us take $\mathcal{A}(S, \Sigma^{\leq k}, C) = (Q, \Sigma, \delta, q_0, F)$, $\mathcal{A}_L = (Q', \Sigma, \delta', q'_0, F')$, and the function $\phi : Q \to Q'$ defined by $\phi(\mathit{row}_k(u)) = \delta'(q'_0, u)$ for all $u \in S$. As in the proof of Lemma 14, 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 $\phi(F) = F'$. Indeed, take $q \in F'$. Because ϕ is bijective, there exists u in S such that $\phi(\mathit{row}_k(u)) = q$. It follows immediately that $\delta'(q'_0, u) \in F'$, and hence $u \in L$. Thus, $C_L(u) = \lambda$ and $\mathit{row}_k(u) \in F$. Clearly, $\phi(\mathit{row}_k(u)) = q \in \phi(F)$. So, $F' \subseteq \phi(F)$, and since $\phi(F) \subseteq F'$, $\phi(F) = F'$ which concludes the proof.

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 $\mathit{row}_k(ua)$ is distinct from all $\mathit{row}_k(v)$ with $v \in S$. This can be done using at most $|S|^2 \cdot |\Sigma| \cdot |\Sigma^{\leq k}|$ operations: there are $|S|$ possibilities for choosing u (and the same number for v), $|\Sigma|$ for choosing a, and $|\Sigma^{\leq k}|$ 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 $\mathcal{A}(S, \Sigma^{\leq k}, C)$ we need n comparisons to determine the final states, and at most $n^2 \cdot |\Sigma| \cdot |\Sigma^{\leq k}|$ operations to construct 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 $\mathcal{O}(n^3 l^k)$.

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

Running Example

Let us trace the run of Algorithm 7 on the language $L = (a+bba)^+$. It is easy to check that the language L is 1-reversible. So, the algorithm starts with $S = \{\lambda\}$ and the following observation table (Table 4.1).

Table 4.1: $S = \{\lambda\}$

T_1	λ	a	b
λ	a	λ	ba
a	λ	λ	ba
b	ba	Θ	a

We can see that the observation table is not 1-closed since both $row_1(a)$ and $row_1(b)$ are different from $row_1(\lambda)$. The algorithm proceeds by adding the strings a and b to S, and updating the table (see Table 4.2).

Table 4.2: $S = \{\lambda, a, b\}$

T_2	λ	a	b
λ	a	λ	ba
a	λ	λ	ba
b	ba	Θ	a
aa	λ	λ	ba
ab	ba	Θ	a
ba	Θ	Θ	Θ
bb	a	λ	Θ

Note that Table 4.2 is still not 1-closed, since $row_1(ba)$ and $row_1(bb)$ are not in $row_1(S)$, so the elements ba and bb are added to S (see Table 4.3).

Table 4.3: $S = \{\lambda, a, b, ba, bb\}$

T_3	λ	a	b	State
λ	a	λ	ba	q_0
a	λ	λ	ba	$q_1 \in F$
b	ba	Θ	a	q_2
ba	Θ	Θ	Θ	q_3
bb	a	λ	Θ	q_4
aa	λ	λ	ba	$q_1 \in F$
ab	ba	Θ	a	q_2
baa	Θ	Θ	Θ	q_3
bab	Θ	Θ	Θ	q_3
bba	λ	λ	ba	$q_1 \in F$
bbb	Θ	Θ	Θ	q_3

The table is now 1-closed, and the algorithm terminates by outputting the automaton $\mathcal{A}(S, \Sigma^{\leq k}, C)$ shown in Figure 4.2. Clearly, $\mathcal{A}(S, \Sigma^{\leq k}, C)$ is a minimal complete DFA accepting L.

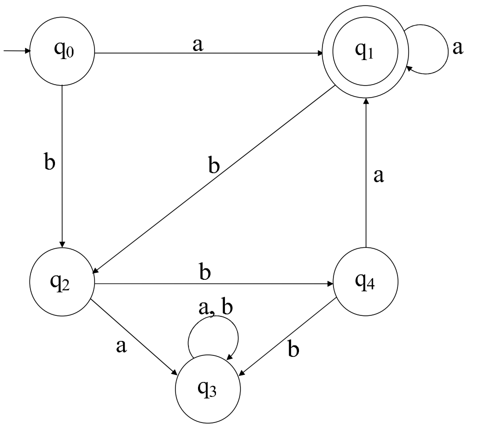


Figure 4.2: The automaton associated with Table 4.3

4.1.3 Polynomial Time Learning with Prefix Correction Queries versus Membership Queries

In this section we make a step further toward understanding the differences and similarities between MQs and PCQs by taking into consideration the efficiency of the learning algorithms. We have seen in Section 3.1.2 that learning with MQs is a strictly weaker model then learning with PCQs when time complexity issues are neglected. One may think that from this it can be automatically inferred that polynomial time learnability with MQs implies polynomial time learnability with PCQs. We show by an example that one should not rush into drawing such conclusions.

Indeed, let us first recall that learning with EQs is strictly more powerful than learning with PCQs when ignoring time complexity: *PCorQ* is strictly included in *LimTxt* (by Theorem 7) and *LimTxt* is strictly included in *EquQ* (see Section 2.3.3). On the other hand, although the class of 0-reversible languages is polynomially learnable with PCQs (one may use Algorithm 7), it is not identifiable in polynomial time with EQs [Ang90].

Going back to the comparison between MQs and PCQs, we introduce first some terminology and notations. Let $\mathcal{C} = (L_i)_{i \geq 1}$ be an indexable class. We say that $\mathcal{C}$ is *polynomially learnable with MQs* (or *with PCQs*) if there exists a polynomial $\mathtt{p}(\cdot)$ and an algorithm $\mathcal{A}lg$ that learns any language L in $\mathcal{C}$ in time $\mathcal{O}(\mathtt{p}(size(L)))$ by asking a finite number of MQs (PCQs, respectively). We denote the collection of all indexable classes $\mathcal{C}$ which are polynomially learnable with MQs by *PolMemQ* (*PolPCorQ* is defined similarly).

Lemma 15. *PolMemQ ⊆ PolPCorQ.*

Proof. Recall that if $C_L(w) = \lambda$, then the string w is in the target language L, and an MQ oracle would return the answer `Yes` when queried with the string w; in all other cases, the same oracle would answer `No`.

Assume that $\mathcal{C}$ is a language class in *PolMemQ* and let *Alg* be a polynomial time algorithm that learns every L of $\mathcal{C}$ after asking a finite number of MQs. Obviously, the number of MQs asked while *Alg* is running with input L is also bounded by a polynomial, let us say $\mathtt{p}(n)$ where n is the size of the target language L. If we modify *Alg* so that instead of asking the oracle a MQ for the string w, to ask a PCQ for the same string, we obtain another algorithm Alg' which learns $\mathcal{C}$ with PCQs (it just uses the information received from asking PCQs to determine whether or not the given string is in the target language). The only thing left to be shown is that Alg' is still polynomial. But this is clear if we note that Alg' performs at most $\mathtt{p}(n)$ more operations than *Alg* (for each queried string w it compares $C_L(w)$ with λ), where n is the size of the target concept. So Alg' is a polynomial time algorithm that learns $\mathcal{C}$ using PCQs. □

We show that the inclusion is strict using pattern languages as the separating case.

Theorem 13. *The class $\mathbb{P}$ is in PolPCorQ\PolMemQ.*

Proof. It is clear that $\mathbb{P}$ is in *PolPCorQ* since Algorithm 6 is a polynomial time algorithm that identifies any pattern language using PCOs (see Section 4.1.1).

Assume now that $\mathbb{P}$ is in *PolMemQ*, and consider the class $\mathcal{S}$ of singletons over the fixed alphabet Σ. Because every language $L_w = \{w\}$ in $\mathcal{S}$ can be written as a pattern language ($L_w = L(w)$, where w is a pattern without any variables), our assumption would imply that $\mathcal{S}$ is also in *PolMemQ*. It is clear though that any algorithm that learns $\mathcal{S}$ using MQs might need to ask $|\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^{|w|}$ MQs in the worst case to learn a given language L_w, which leads to a contradiction. □

Note that although $\mathbb{P}$ is not polynomially learnable with MQs, it is in *MemQ* (see [Muk92a], page 266). However, there are classes of languages in *PolPCorQ* which cannot be learned at all (polynomially or not) using MQs, as we will see in the sequel.

Theorem 14. *The class k-Rev is in PolPCorQ*$\setminus$*MemQ.*

Proof. Since Algorithm 7 learns any k-reversible language using PCQs in polynomial time (see Section 4.1.2), it follows immediately that k-*Rev* $\in$ *PolPCorQ*.

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. Recall 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 $\Sigma^*\setminus\{a\}$. Let us take m to be $\max\{|w| \mid w \in F\}$ if $F \neq \emptyset$ and 0 otherwise, and consider 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 k-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 PCQs. For example, if we take $\bar{\mathcal{S}}$ to be $\bar{\mathcal{S}} = (\bar{L}_w)_{w\in\Sigma^*}$, where $\bar{L}_w = \Sigma^*\setminus\{w\}$, then any algorithm would require at least $1 + |\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^{|w|}$ PCQs in order to learn $\bar{L}_w$.

Figure 4.3 displays the relations between the two models.

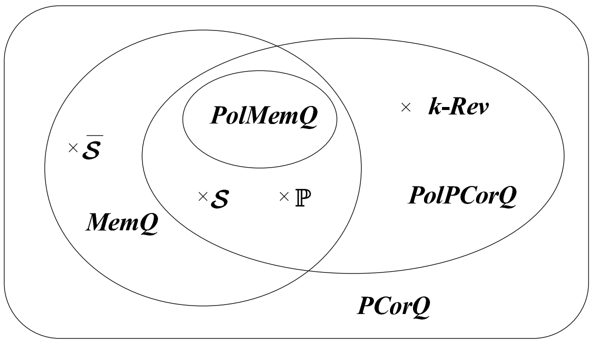


Figure 4.3: PCQ learning versus MQ learning

4.2 Polynomial Time Learning with Length Bounded Correction Queries

Let l be a fixed nonnegative integer, and let us denote by *PollBCorQ* the collection of all indexable classes $\mathcal{C}$ for which there exists a polynomial $\mathtt{p}(\cdot)$ and an algorithm $\mathcal{A}lg$ that learns any language L in $\mathcal{C}$ in time $\mathcal{O}(\mathtt{p}(size(L)))$ by asking a finite number of lBCQs. Clearly, we only consider those language classes $\mathcal{C}$ for which the size of its languages is independent of l (more details are presented further on in this section).

Lemma 16. *Pol0BCorQ = PolMemQ.*

Proof. The result is straightforward from the definition of $C_L^0(w)$: for any language L over Σ and any string w in Σ^*,

$$C_L^0(w) = \begin{cases} \lambda, & \text{if } w \in L, \text{ and} \\ \emptyset, & \text{if } w \notin L. \end{cases}$$

□

Lemma 17. *Pol(l-1)BCorQ = PollBCorQ for any $l \geq 1$.*

Proof. Since one can easily extract the answer to an $(l-1)$BCQ from the corresponding lBCQ, it is clear that *Pol(l-1)BCorQ* is included in *PollBCorQ*. Let us now show that *PollBCorQ* is included in *Pol(l-1)BCorQ*. Assume $\mathcal{C}$ is an indexed family of languages in *PollBCorQ* and let $\mathcal{A}lg$ be a polynomial time algorithm that learns $\mathcal{C}$ with lBCQs. Note that for any language L over Σ, any $w \in \Sigma^*$ and any $l \geq 1$,

$$\begin{aligned} C_L^l(w) &= \{u \in \Sigma^{\leq l} \mid wu \in L\} \\ &= \{u \in \Sigma^{\leq l-1} \mid wu \in L\} \cup \{au \mid a \in \Sigma, u \in \Sigma^{l-1} \text{ and } wau \in L\} \\ &= C_L^{l-1}(w) \cup \{au \mid a \in \Sigma, u \in C_L^{l-1}(wa)\} \\ &= C_L^{l-1}(w) \cup \textstyle\bigcup_{a\in\Sigma} aC_L^{l-1}(wa). \end{aligned}$$

So, one can modify $\mathcal{A}lg$ such that instead of asking an lBCQ for the string w, to ask a finite number[1] of $(l-1)$BCQs ($|\Sigma|+1$ queries to be precise) for the strings wa with a in $\{\lambda\}\cup\Sigma$. Clearly, the modified algorithm is still polynomial. Hence, *Pol(l-1)BCorQ* equals *PollBCorQ*. □

The following theorem is a direct consequence of Lemma 16 and Lemma 17.

Theorem 15. *PollBCorQ = PolMemQ for any $l \geq 0$.*

Therefore, we can introduce the notation *PolLBCorQ* for the collection of all language classes that are efficiently learnable with LBCQs. Moreover, by

[1]Thanks are due to C. de la Higuera for suggesting this.

combining the theorem above with Lemma 15 and Theorem 13, one gets the following corollary.

Corollary 8. $PollBCorQ \subsetneq PolPCorQ$.

In the beginning of this section we pointed out that we should think of l as a constant. By failing to do so, we may end up with contradictory results, as one can see in Example 9.

Example 9. Let $\mathcal{S}_l$ be the class of all singleton languages of size $l+2$, i.e., $\mathcal{S}_l = (L_w)_{w \in \Sigma^l}$ where $L_w = \{w\}$. We can imagine a very simple lBCQ algorithm to learn this class. The learner would simply ask one lBCQ, for the string λ, and then output the string received as an answer. Since in this case the set of possible corrections contains just one string of length $size(L)-2$ where L is the target language, it means that such an algorithm would work in linear time in the size of the language. On the other hand, with an MQ oracle, any learner would have to ask at least $|\Sigma|^l - 1$ MQs, hence an exponential number of queries in the size of the target language. Thus, one might think that this language class invalidates the result of Theorem 15. The trick here is that l is no longer a constant for the class $\mathcal{S}_l$, and that if we do think of l as a constant, then $|\Sigma|^l - 1$ is a constant as well.

So, what we have learned in this section is that having the possibility to get answers for more than one MQ at once does not add any more learning power, even if we impose time restrictions.

4.3 Polynomial Time Learning with Edit Distance Based Correction Queries

We continue the analysis done in Section 3.3 on the power of learning with edit distance based correction queries, this time by taking into account time complexity issues. We have seen that what happens in the general model does not necessary carry on to the polynomially bounded model. Let us recall the results we have so far:

$PCorQ \subset EquQ$	$PolPCorQ \not\subset PolEquQ$
$MemQ \subsetneq PCorQ$	$PolMemQ \subsetneq PolPCorQ$
$MemQ = LBCorQ$	$PolMemQ = PolLBCorQ$
$MemQ = EditCorQ$	$PolMemQ \overset{?}{=} PolEditCorQ$

Table 4.4: Relations between various learning models

So in the case of LBCQs versus MQs, as well as in the case of PCQs versus MQs, the relation existing in the general case is preserved in the polynomial learning model, while for the EQs versus PCQs it does not happen. On the other hand, we already know that *MemQ* = *EditCorQ*, and the question is whether or not *PolMemQ* = *PolEditCorQ*, where by *PolEditCorQ* we denote, as usual, the collection of all indexable classes $\mathcal{C}$ for which there exists a polynomial $\mathtt{p}(\cdot)$ and an algorithm $\mathcal{A}lg$ that learns any language L in $\mathcal{C}$ in time $\mathcal{O}(\mathtt{p}(size(L)))$ by asking a finite number of EDCQs.

In this section we answer this question in the negative way, and we describe some of the algorithms which use alternative CQs based on edit distance existing in the literature.

Lemma 18. *PolMemQ* $\subsetneq$ *PolEditCorQ*.

Proof. One may show that *PolMemQ* is included in *PolEditCorQ* using an argument similar with the one presented in the proof of Lemma 15, the only difference being that in the case of EDCQs, the new algorithm $\mathcal{A}lg'$ has to check whether or not $\text{EDC}_L(w)$ equals `Yes`, for all the strings w submitted to the oracle by the learner of the original algorithm $\mathcal{A}lg$ (L is the target language).

Moreover, if $\mathcal{S}$ is the class of singleton languages over the alphabet Σ, then $\mathcal{S}$ can be used as a separating language class:

- $\mathcal{S} \notin$ *PolMemQ* (see the proof of Theorem 13),
- one may imagine a very simple edit distance query algorithm for this class. Indeed, it is enough to ask an EDCQ for an arbitrarily chosen string w. The algorithm just outputs w, if the oracle's answer is `Yes`, and w' if the answer returned by the oracle is the string w'. Since the algorithm described above is polynomial in the size of the target language, we conclude that $\mathcal{S} \in$ *PolEditCorQ*.

□

So, we have seen that singleton languages can be learned in polynomial time using EDCQs. But these are rather simple objects and their learnability not very interesting from a practical point of view. We will see in the sequel that EDCQs can be a useful tool in learning more complicated language classes.

The first language class we will be discussing about is the class of *balls of strings*, named like this because of its resemblance to a disk when we imagine its geometrical interpretation. We recall that given a string w and a real number r, the ball of center w and radius r is $B_r(w) = \{v \in \Sigma^* \mid d(v,w) \leq r\}$. In [BBdlHJT07], an algorithm for learning the class of all balls of strings using EDCQ is given. Moreover, the authors of the above mentioned paper show that the number of EDCQs used by this algorithm can be bounded by $\mathcal{O}(|\Sigma|+|o|+r)$. Furthermore, for what they called $\mathtt{q}$-good balls (the balls $B_r(w)$ for which there exists a polynomial $\mathtt{q}(\cdot)$ such that the radius r is no longer than $\mathtt{q}(|w|)$), the algorithm runs in polynomial time in the size of (the representation of) the language.

Let $L = B_r(w)$ be the target language over the alphabet $\Sigma = \{a_1, \ldots, a_l\}$. The idea of the algorithm is based on the following two observations (both enunciated and thoroughly proved in [BBdlHJT07]):

- given a string of maximum length in L and the radius r, one may find the center w of the ball by asking $\mathcal{O}(|w| + r)$ EDCQs (actually, in this case asking MQs suffices).

- for a given k big enough (details about how to compute this k can be found in [BBdlHJT07]; the important thing is it can be computed using $\mathcal{O}(|\Sigma| + \log_2(|w| + r))$ EDCQs), the string $u = \textsc{Edc}_L((a_1 \cdots a_l)^k)$ is a string of maximum length in L, and the radius r can be computed with the formula
$$r = \frac{k - |u|}{l - 1}$$

A slightly different type of EDCQs is used in [Kin08] for learning a subclass of regular expressions, and the class of pattern languages: if the queried string is not in the target language, then the oracle returns the positive example with a smallest distance from the queried string and **previously not used in the learning process**. Moreover, preference is given to correcting strings of the same length, if any, and among those having the same length, the smallest one with respect to the lexicographical order is returned. More formally, Kinber's EDCQ of a string w with respect to the language L is given by Algorithm 8.

Algorithm 8 Computing the EDCQ of w w.r.t. L given A

```
1: input: $w, L, A$
2: if $w \in L$ then
3:    output Yes
4: else
5:    $V := \{v \in L \backslash A \mid |v| = |w| \text{ and } d(v, w) = \min\{d(u, w) \mid u \in L \backslash A\}\}$
6:    if $V \neq \emptyset$ then
7:       output $\min_{v \in V} v$
8:    else
9:       $V' := \{v \in L \backslash A \mid d(v, w) = \min\{d(u, w) \mid u \in L \backslash A\}\}$
10:      if $V' \neq \emptyset$ then
11:         output $\min_{v \in V'} v$
12:      else
13:         output $\epsilon$
14:      end if
15:   end if
16: end if
```

In the above definition, ϵ is a symbol not in Σ, and A is the set of all strings for which the learner has received the answer `Yes` on previous steps and all the correcting strings received by the learner on previous steps. Moreover, the minimum between strings is taken with respect to the lex-length order.

In [Kin08] two algorithms are given. The one for learning the class of pattern languages with this special type of EDCQs works in time $\mathcal{O}(n^3)$, asking $\mathcal{O}(n^2)$ queries. The other one for learning regular expressions of type

$$(*) \quad u_1(v_1)^+u_2(v_2)^+\dots u_n(v_n)^+u_{n+1}$$

uses only one EDCQ and $\mathcal{O}(n^3)$ MQs and runs in $\mathcal{O}(n^4)$. The reader is referred to [Kin08] for further details. The reason for mentioning these two results here will be apparent in the next section.

4.4 Remarks and Further Research

In the end of the previous chapter we exhibited a complete picture of the relations existing between several models of learning with CQs and other learning models (both Gold-style and query learning). We have seen that when we neglect time complexity issues we can characterize the newly introduced query models in terms of finite sets, and that learning with LBCQs and EDCQs is basically the same as learning with MQs, whereas PCQs are the only ones adding some power to the model.

When we restrict to polynomial time algorithms, things are changing. And although having an LBCQs oracle does still not improve on the learnability power with respect to the MQ learning model, an EDCQ oracle or a PCQ oracle does. It is not clear what relation is between learning with EDCQs and learning with PCQs when we restrict to efficient algorithms. We conjecture that the two classes are incomparable (see Figure 4.4).

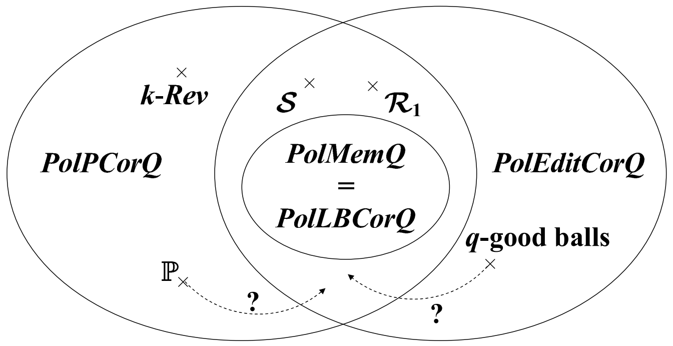


Figure 4.4: Different types of correction queries

Let us first notice that the class $\mathcal{R}_1$ of regular expressions of type $(*)$ is learnable with EDCQs and with PCQs, since we only need one CQ to get the shortest string in the language (i.e., the string $u_1v_1u_2\dots u_nv_nu_{n+1}$) and then MQs (which can be simulated with both types of CQs). It can be easily shown that the class of all singleton languages also belongs to *PolPCorQ* $\cap$ *PolEditCorQ*.

Moreover, we argue that *PolPCorQ*$\setminus$*PolEditCorQ* $\neq \emptyset$. Indeed, the class *k-Rev* is in *PolPCorQ* and not in *MemQ* (by Theorem 14), and since *EditCorQ* = *MemQ* (by Theorem 12), we obtain that k-reversible languages are not learnable with EDCQs either. Hence, *k-Rev* $\in$ *PolPCorQ*$\setminus$*PolEditCorQ*.

If for the class of k-reversible languages it was easy to decide whether or not it is in *PolEditCorQ*, we cannot say the same thing for the class of pattern languages. Kinber describes an efficient algorithm that learns $\mathbb{P}$ with the modified type of EDCQs with the strong requirement that the oracle must not return as a correction any of the strings which appeared before. We strongly believe that this requirement is actually mandatory, i.e., there is no algorithm that can learn the class $\mathbb{P}$ with standard EDCQs. Far from being a proof, Example 10 shows, nevertheless, on what our intuition is based on.

Example 10. Let $\pi = 1xyyy111$ and $w_0 = 11000111$ a string of minimum length in $L(\pi)$. Then, although it is easy to determine the position of single variables in the pattern (asking whether or not $w = 1\mathbf{0}000111$ is in the language suffices), it is quite hard to find a way to distinguish between constants and multiple variables (i.e., variables which appear more than once in the pattern) if we are faced with an unfriendly oracle. Suppose the learner asks, for example, if $11\mathbf{1}00111$ is in the target language and the oracle returns as a correction the string w_0. In this case, there is no way the learner can deduce whether the 3rd symbol of the pattern is 0, or a variable which appears more then once in π. On the other hand, changing pairs (or triples, quadruples, etc.) of symbols at once increases dramatically the number of queries needed.

To complete the picture, we would like to be able to say if there are language classes in *PolEditCorQ*$\setminus$*PolPCorQ*. A possible candidate is the class of q-good balls of strings, which is known to be learnable with EDCQs in polynomial time.

Another future work direction is to find characterizations for the language classes polynomial time learnable with (each type of) CQs. One can investigate, for example, the *teaching dimension of a concept class* [GK91] (that is, the minimum number of corrections a teacher must reveal to uniquely identify any concept in the class), or the computational power of polynomial time query learning systems for different correction query types as in [Wat90, WG94]. Slightly relaxed learning criteria may lead to new learnability results. For example, in the *bounded learning* framework introduced by Watanabe [Wat94], the learning condition does not insist in exactly identifying a target language L, but only requires that a learner should return, for any given length bound m, a language L' such that L and L' coincide up to length m (i.e., $L \cap \Sigma^{\leq m} = L' \cap \Sigma^{\leq m}$).

Chapter 5

Learning Deterministic Finite Automata with Correction Queries and Equivalence Queries

Efficient learning of DFAs is a challenging research problem in grammatical inference. It is known that both exact and approximate (in the PAC sense) identifiability of DFA is hard [PH97]. There is a lot of work done on learning more general types of automata or learning them in various learning formalisms (see, for examples, the surveys by Pitt [Pit89], Ron [Ron95], Balcázar, Díaz, Gavaldà and Watanabe [BDGW97]). The first important result obtained on learning DFAs is essentially negative: there is no algorithm that can identify DFAs from text in the limit (a direct consequence of the fact that no superfinite class of languages is learnable from text in the limit [Gol67]). From that, one can easily infer that CQs alone are not enough to learn the class of DFAs, no matter which (of the several types of) corrections are used (recall that *LBCorQ* = *EditCorQ* = *MemQ*, and both *MemQ* and *PCorQ* are strictly included in *LimTxt* - see Chapter 3).

The first positive result concerning the learnability of DFAs is due to Angluin, who gives a polynomial query learning algorithm, called L*, that identifies any minimal complete DFA using MQs and EQs [Ang87c]. We propose[1] to investigate what happens when we modify L* such that the learner gets, instead of a simple `No` answer for strings not in the target language, a correction for

[1]The idea of replacing MQs in L* with more powerful queries belongs to L. Becerra Bonache.

the given input string. On one hand, we know that, in general, CQs are more powerful than MQs, and on the other hand, implementing a teacher that can answer these types of queries is *easy* (can be done in polynomial time). In this chapter we design polynomial time algorithms for learning DFAs with PCQs and EQs, and with LBCQs and EQs, respectively, and we compare them with L^* and with each other. One may argue that we left out from our study the edit distance correction queries. We will see why we did so in the end of this chapter.

5.1 Learning Deterministic Finite Automata with Prefix Correction Queries and Equivalence Queries

Recall that we defined the prefix correcting string $C_L(w)$ of w with respect to L to be the minimum string in the lex-length order of the set $Tail_L(w)$, if this set is not empty, and the symbol Θ otherwise.

5.1.1 The Algorithm LCA

Let L be the unknown regular set and Σ the alphabet of L. 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 map C which is the restriction of C_L 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 u and column e equal to $C(u \cdot e)$. If u is an element of $S \cup S\Sigma$ then $row(u)$ denotes the finite function from E to $\Sigma^* \cup \{\Theta\}$ defined by $row(u)(e) = C(u \cdot e)$. By $row(S)$ we understand the set $\{row(u) \mid u \in S\}$.

The algorithm LCA uses the observation table to build a DFA. 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.

An observation table (S, E, C) is called *closed* if for every u in S and $a \in \Sigma$ there exists a v in S such that $row(ua) = row(v)$, and *consistent* if for any u_1, u_2 in S such that $row(u_1) = row(u_2)$, we have $row(u_1 \cdot a) = row(u_2 \cdot a)$ for all a in Σ.

Let (S, E, C) be a closed and consistent observation table. We define the automaton $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$ as follows:

- $Q = \{row(u) \mid u \in S\}$
- $q_0 = row(\lambda)$
- $F = \{row(u) \mid u \in S \text{ and } C(u) = \lambda\}$
- $\delta(row(u), a) = row(u \cdot a)$

It can be easily shown that q is a sink state if and only if q belongs to $\{row(u) \mid u \in S \text{ and } C(u) = \Theta\}$ (from Remark 4 we know that $C(u) = \Theta$ implies $C(u \cdot a) = \Theta$, $\forall a \in \Sigma$).

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 u_1 and u_2 are elements of S such that $row(u_1)$=$row(u_2)$, then $C(u_1) = C(u_1 \cdot \lambda) = row(u_1)(\lambda)$ and $C(u_2) = C(u_2 \cdot \lambda) = row(u_2)(\lambda)$ are defined and equal to each other, hence F is well defined. To see that δ is well defined, suppose u_1 and u_2 are elements of S such that $row(u_1) = row(u_2)$. Then, since the observation table $\mathcal{A}(S, E, C)$ is consistent, for each a in Σ, $row(u_1 \cdot a) = row(u_2 \cdot a)$, and since it is closed, this common value is equal to $row(u)$ for some u in S.

The learner algorithm (denoted LCA) uses as its main data structure the observation table that we described above. Initially $S = E = \{\lambda\}$. To determine the entries for the table (S, E, C), LCA asks PCQs for λ and each a in Σ.

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

When the learner's automaton is closed and consistent the learner asks an EQ. The teacher's answer can be `Yes` (in which case the algorithm terminates with the output $\mathcal{A}(S, E, C)$) or `No` (in which case a counterexample is provided, all its prefixes are added to S and the table is updated using PCQs).

The algorithm itself is very similar to L^*. The main difference consists in the fact that the entries for the observation table are strings instead of binary elements.

Algorithm 9 LCA: Learning DFAs with PCQs and EQs

```
1: Initialize $S$ and $E$ with $\{\lambda\}$
2: Ask PCQs for $\lambda$ and each $a \in \Sigma$
3: Construct the initial observation table $(S, E, C)$
4: repeat
5:   repeat
6:     while $(S, E, C)$ is not closed do
7:       find $u$ in $S$ and $a$ in $\Sigma$ such that $row(u \cdot a) \notin row(S)$
8:       add $u \cdot a$ to $S$
9:       extend $C$ to $(S \cup S\Sigma)E$ using PCQs
10:    end while
11:    while $(S, E, C)$ is not consistent do
12:      find $u_1$,$u_2 \in S$ and $a \in \Sigma$, $e \in E$ such that $row(u_1) = row(u_2)$ and
         $C(u_1 \cdot a \cdot e) \neq C(u_2 \cdot a \cdot e)$
13:      add $a \cdot e$ to $E$
14:      extend $C$ to $(S \cup S\Sigma)E$ using PCQs
15:    end while
16:  until $(S, E, C)$ is closed and consistent
17:  Construct the conjecture $\mathcal{A}(S, E, C)$
18:  if the teacher replies with a counterexample $u$ then
19:    add $u$ and all its prefixes to $S$
20:    extend $C$ to $(S \cup S\Sigma)E$ using PCQs
21:  end if
22: until the teacher replies Yes to the conjecture
23: Halt and output $\mathcal{A}(S, E, C)$
```

We show that the algorithm runs in polynomial time, and that it terminates by outputting the minimal complete DFA for the target language.

5.1.2 Correctness, Termination and Running Time

It is clear that if LCA terminates, its output is the target language. Recall that the teacher's last answer to an EQ before halting is `Yes`.

Assume that (S, E, C) is a closed and consistent observation table. We say that the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is *consistent with the function* C *with respect to the observation table* (S, E, C) if for every u in $S \cup S\Sigma$ and e in E, the following statements hold:

1. $C(u \cdot e) = \Theta$ if and only if $\delta(q_0, u \cdot e)$ is a sink state,

2. $C(u \cdot e) = v \in \Sigma^*$ if and only if $(\delta(q_0, u \cdot e \cdot v) \in F$ and for all $v' \in \Sigma^*$, $\delta(q_0, u \cdot e \cdot v') \in F$ implies $v \preceq v')$.

To prove that LCA terminates in finite steps, we show that:

- when the observation table (S, E, C) is closed and consistent, $\mathcal{A}(S, E, C)$ is the minimal complete DFA consistent with C with respect to the given table (S, E, C) (note that $\mathcal{A}(S, E, C)$ has exactly as many states as the number of distinct rows in S), and

- the number of distinct rows in S is initially one, it cannot exceed n (the size of the target DFA), and increases by at least one each time the table is found not closed or not consistent; moreover, each time a counterexample is added to S, the table becomes not consistent (see Lemma 23).

Let us first state and prove the following lemmas.

Lemma 19. *Assume that (S, E, C) is a closed and consistent observation table. For the automaton $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$ and for every u in $S \cup S\Sigma$, $\delta(q_0, u) = row(u)$.*

Proof. We prove this lemma by induction on the length of u. When $u = \lambda$, $\delta(q_0, u) = row(u)$ becomes $q_0 = row(\lambda)$ which is true by the definition of q_0. Assume that the equality holds for all strings of length at most k, and let u be an arbitrary string in $S \cup S\Sigma$ such that $|u| = k+1$. Since S is prefix-closed, $u = va$ for some $v \in S$ and $a \in \Sigma$. By the induction hypothesis we get $\delta(q_0, v) = row(v)$, so we immediately obtain: $\delta(q_0, u) = \delta(q_0, va) = \delta(\delta(q_0, v), a) = \delta(row(v), a) = row(va) = row(u)$. $\square$

Lemma 20. *Assume that (S, E, C) is a closed and consistent observation table. For the automaton $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$ and for every u in $S \cup S\Sigma$ and $e \in E$, there exists v in S such that $\delta(q_0, u \cdot e) = \delta(q_0, v)$ and $C(u \cdot e) = C(v)$.*

Proof. Let u be an arbitrary string in $S \cup S\Sigma$. Since the table (S, E, C) is closed, there exists v in S such that $row(v) = row(u)$. So, $\delta(q_0, v) = \delta(q_0, u)$ (by Lemma 19) and $C(u \cdot e) = C(v \cdot e)$ for any e in E . The proof is by induction on the length of e.

- If $e = \lambda$, then the equalities $\delta(q_0, u \cdot \lambda) = \delta(q_0, v)$ and $C(u \cdot \lambda) = C(v)$ obviously hold.
- Suppose the result is true for all strings 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. By the induction hypothesis on e', there exists v' in S such that $\delta(q_0, va \cdot e') = \delta(q_0, v')$ and $C(va \cdot e') = C(v')$. Thus, $\delta(q_0, u \cdot e) = \delta(q_0, v')$ and $C(u \cdot e) = C(v')$.

We have shown that for any e in E there exists v' in S such that $C(u \cdot e) = C(v')$ and $\delta(q_0, u \cdot e) = \delta(q_0, v')$ which concludes our proof. $\square$

Lemma 21. *Assume that (S, E, C) is a closed and consistent observation table. Then the automaton $\mathcal{A}(S, E, C)$ is consistent with the function C with respect to (S, E, C).*

Proof. Let w be in $S \cup S\Sigma$ and e in E. From Lemma 20 we know that there exists u_0 in S such that $\delta(q_0, w \cdot e) = \delta(q_0, u_0)$ and $C(w \cdot e) = C(u_0)$. Hence, we have to show that:

1. $C(u_0) = \Theta$ if and only if $\delta(q_0, u_0)$ is a sink state,
2. $C(u_0) = u \in \Sigma^*$ if and only if ($\delta(q_0, u_0 \cdot u) \in F$ and for all $v \in \Sigma^*$, $\delta(q_0, u_0 \cdot v) \in F$ implies $u \preceq v$).

Clearly, $C(u_0) = \Theta \Leftrightarrow row(u_0)$ is a sink state $\Leftrightarrow \delta(q_0, u_0)$ is a sink state. Assume now that $C(u_0) \neq \Theta$. We are going to show that:

- if $C(u_0) = u$ then $\delta(q_0, u_0 \cdot u) \in F$, and
- if v is the smallest string in Σ^* with the property $\delta(q_0, u_0 \cdot v) \in F$, then $C(u_0) = v$.

So, let $C(u_0) = u = a_1 \cdot a_2 \cdots a_n$ for some $a_1, a_2, \ldots, a_n$ in Σ, $n \geq 0$. Because the table (S, E, C) is closed, we can inductively find the strings $u_i \in S$, $i \in \{1, \ldots, n\}$ such that $row(u_{i-1} \cdot a_i) = row(u_i)$. Therefore, $\delta(q_0, u_{i-1} \cdot a_i) = \delta(q_0, u_i)$ and $C(u_{i-1} \cdot a_i) = C(u_i)$. Furthermore, we get by induction that $\delta(q_0, u_0 \cdot u) = \delta(q_0, u_n)$. Now, $C(u_0) = a_1 \cdot a_2 \cdots a_n$ implies $C(u_0 \cdot a_1) = a_2 \cdots a_n$, that is, $C(u_1) = a_2 \cdots a_n$. Reasoning in the same way we obtain $C(u_n) = \lambda$. But this means that $row(u_n)$ is in F, and so is $\delta(q_0, u_n)$. If we recall that $\delta(q_0, u_n) = \delta(q_0, u_0 \cdot u)$, we get $\delta(q_0, u_0 \cdot u) \in F$.

Now, assume $v = b_1 \cdot b_2 \cdots b_m$ ($m \geq 0$) is the smallest string in Σ^* with the property $\delta(q_0, u_0 \cdot v) \in F$. Let $v_1, v_2, \ldots, v_m$ be strings in S such that $row(u_0 \cdot b_1) = row(v_1)$ and for all $j \in \{2, \ldots, m\}$, $row(v_{j-1} \cdot b_j) = row(v_j)$ (the table (S, E, C) is closed). As before, we get $\delta(q_0, u_0 \cdot v) = \delta(q_0, v_m)$ and $C(v_m) = \lambda$. Because $\delta(q_0, u_0 \cdot v) = \delta(q_0, v_m)$ and $\delta(q_0, u_0 \cdot v) \in F$ it follows that $C(v_m) = \lambda$ and hence $C(v_{m-1} \cdot b_m) = \lambda$ which implies $b_m \in Tail_L(v_{m-1})$. Assuming that $C(v_{m-1}) \neq b_m$ leads to a contradiction. Hence $C(v_{m-1}) = b_m$. Reasoning in the same manner we obtain that $C(u_0) = v$, which concludes the proof. □

Lemma 22. *Assume that (S, E, C) is a closed, consistent observation table. Assume the automaton $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$ has n states. If $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ is any automaton consistent with C with respect to (S, E, C) that has n or fewer states, then $\mathcal{A}'$ is isomorphic with $\mathcal{A}(S, E, C)$.*

Proof. We define the relation $\phi \subseteq Q \times Q'$ as follows. For all $u \in S$, $row(u)\ \phi\ q'$ if $q' = \delta'(q_0', u)$. We show that:

- *ϕ is a bijective function.* For this, we first prove that ϕ is an injective relation. Indeed, let us assume the contrary, namely that there exist q' in Q' and $u_1, u_2 \in S$ such that $row(u_1)\ \phi\ q'$, $row(u_2)\ \phi\ q'$ and $row(u_1) \neq row(u_2)$. So $\delta'(q'_0, u_1) = q' = \delta'(q'_0, u_2)$, and there exists $e \in E$ such that $row(u_1)(e) \neq row(u_2)(e)$ which is equivalent to $C(u_1 \cdot e) \neq C(u_2 \cdot e)$. We can distinguish two cases: one of the values of $C(u_1 \cdot e)$, $C(u_2 \cdot e)$ is Θ, or both of them are in Σ^*.

  I) Assume $C(u_1 \cdot e) = \Theta$ and $C(u_2 \cdot e) = v \neq \Theta$ (the case $C(s_1 \cdot e) \in \Sigma^*$ and $C(u_2 \cdot e) = \Theta$ is symmetric). Because $\mathcal{A}'$ is consistent with C with respect to (S, E, C) we get that $\delta'(q'_0, u_1 \cdot e)$ is a sink state and $\delta'(q'_0, u_2 \cdot e \cdot v) \in F'$. But $\delta'(q'_0, u_1) = \delta'(q'_0, u_2)$ implies $\delta'(q'_0, u_1 \cdot e) = \delta'(q'_0, u_2 \cdot e)$, so $\delta'(q'_0, u_2 \cdot e)$ is a sink state and hence $\delta'(q'_0, u_2 \cdot e \cdot v)$ is also a sink state, which contradicts $\delta'(q'_0, u_2 \cdot e \cdot v) \in F'$.

  II) Assume now that $C(u_1 \cdot e) = v_1$ and $C(u_2 \cdot e) = v_2$ for some $v_1, v_2 \in \Sigma^*$ such that $v_1 \neq v_2$. Because $\mathcal{A}'$ is consistent with C with respect to (S, E, C) we get that $\delta'(q'_0, u_1 \cdot e \cdot v_1) \in F'$, $\delta'(q'_0, u_2 \cdot e \cdot v_2) \in F'$ and v_1, v_2 are the smallest strings with this property. But $\delta'(q'_0, u_1) = \delta'(q'_0, u_2)$ implies $\delta'(q'_0, u_1 \cdot e \cdot v_1) = \delta'(q'_0, u_2 \cdot e \cdot v_1)$, so $\delta'(q'_0, u_2 \cdot e \cdot v_1) \in F'$ and $v_2 \preceq v_1$. In a similar way it can be shown that $v_1 \preceq v_2$, and hence $v_1 = v_2$ which leads to a contradiction.

  Because ϕ is an injection we deduce that $|Q| \leq |\phi(Q)|$. From the hypothesis we know that $|Q'| \leq |Q|$ and hence $|Q| = |\phi(Q)| = |Q'|$, which makes our relation ϕ a function. It follows immediately that ϕ is bijective since it is injective and has the domain and range finite and of the same cardinality.

- *ϕ is an automata isomorphism*, that is $\phi(q_0) = q'_0$, $\phi(F) = F'$ and for all $u \in S$, $a \in \Sigma$, $\phi(\delta(row(u), a)) = \delta'(\phi(row(u)), a)$. The proofs for the first two statements are straightforward. For the last one, we take $v \in S$ such that $row(v) = row(u \cdot a)$. We have that $\phi(\delta(row(u), a)) = \phi(row(u \cdot a)) = \phi(row(v)) = \delta'(q'_0, v)$ and $\delta'(\phi(row(u)), a) = \delta'(\delta'(q'_0, u), a) = \delta'(q'_0, u \cdot a)$. Since $\delta'(q'_0, v)$ and $\delta'(q'_0, u \cdot a)$ have identical row values, namely $row(v)$ and $row(u \cdot a)$, they must be the same state of $\mathcal{A}'$.

We have constructed an automata isomorphism from $\mathcal{A}(S, E, C)$ to $\mathcal{A}'$, which concludes our proof. □

The following result can be deduced from the above mentioned lemmas.

Theorem 16. *If (S, E, C) is a closed and consistent observation table, then the automaton $\mathcal{A}(S, E, C)$ is consistent with the finite function C with respect to (S, E, C). Any other automaton consistent with C with respect to (S, E, C) but inequivalent to $\mathcal{A}(S, E, C)$ must have more states.*

Proof. Lemma 21 states that $\mathcal{A}(S, E, C)$ is consistent with C with respect to (S, E, C), and Lemma 22 states that any other automaton consistent with C with respect to (S, E, C) is either isomorphic to $\mathcal{A}(S, E, C)$ or contains at least one more state. Thus, $\mathcal{A}(S, E, C)$ is the unique smallest DFA consistent with C with respect to (S, E, C). □

Now, let us notice that the injectivity of function ϕ defined on Lemma 22 implies that for any closed and consistent observation table (S, E, C), if k denotes the number of different values of $row(u)$ for u in S then any automaton consistent with C must have at least k states.

Suppose that n is the size of the target DFA. As the number of distinct values of $row(u)$ for u in S increases by at least one when the table is found not closed or not consistent, the total number of operations of either type cannot be more than $n - 1$ (this number is initially 1 and it is bounded by n). Hence LCA always eventually finds a closed, consistent observation table (S, E, C) and makes a conjecture $\mathcal{A}(S, E, C)$. Moreover, LCA can make at most $n-1$ incorrect conjectures, since the size of the conjectured automaton is initially at least one, and may not exceed $n - 1$ (whenever the teacher answers by a counterexample, the number of distinct values of $row(u)$ for u in S increases by at least 1 - see Lemma 23). Since LCA has to make another conjecture as long as it is running, it must terminate by making a correct conjecture.

Let us now discuss the worst case query complexity of LCA. We denote by n the size of the target DFA and by m the length of the longest counterexample returned by the teacher. As we already mentioned, the algorithm cannot ask more than $n - 1$ EQs. Moreover, the number of PCQs asked by LCA during its run can be bounded by the number of elements of the final observation table (S, E, C). It is easy to see that the number of strings in S cannot exceed $n + m(n - 1)$ (a counterexample requires the addition of at most m strings to S, and this can happen at most $n - 1$ times), that $S \cup S\Sigma$ has at most $(|\Sigma| + 1)|S|$ elements and that in E the number of strings cannot be greater than n. So, the total number of PCQs is bounded by $(|\Sigma| + 1)(n + m(n - 1))n$.

The total running time of the algorithm can also be expressed as a polynomial in m and n, as we will see in the sequel. We have seen that the table can be found not closed or not consistent at most $n - 1$ times. When the observation table is closed and consistent, the algorithm constructs the automaton $\mathcal{A}(S, E, C)$ and asks an EQ, but this can happen no more then $n - 1$ times.

To check for closure, LCA considers each pair (u, v) in $S \times (S\Sigma \backslash S)$ and compares the values of the two rows, $row(u)$ and $row(v)$ (in a worst case situation). This task can be done in time $\mathcal{O}(|S| \cdot |S\Sigma \backslash S| \cdot |E| \cdot n)$ (note that the entries of the observation table are strings of length at most n).

To check for consistency, the algorithm first finds the strings u_1, u_2 in S such that $row(u_1) = row(u_2)$. This requires, in the worst case, $|S|^2 \cdot |E| \cdot n$ operations. Then it tries to find $a \in \Sigma$ such that $row(u_1 \cdot a) \neq row(u_2 \cdot a)$. In the worst case, the algorithm would have to perform $|\Sigma| \cdot |E| \cdot n$ operations. Summing up, we obtain that LCA needs at most $\mathcal{O}((|S|^2 + |\Sigma|) \cdot |E| \cdot n)$ time steps to check if the table is consistent or not.

Since the conjectured automaton $\mathcal{A}(S, E, C)$ can be constructed in time polynomial in the size of the observation table, the total running time of the learning process can be bounded by a polynomial function in m and n.

5.1.3 Running Example

We explain how our algorithm runs by tracing the evolution of the observation table for the language $L = (a + bba)^+$ over the alphabet $\Sigma = \{a, b\}$. Initially the learner starts with $S = \{\lambda\}, E = \{\lambda\}$ and the observation table described as Table 5.1.

Table 5.1: $S = \{\lambda\}, E = \{\lambda\}$

T_1	λ
λ	a
a	λ
b	ba

We can observe that the information for the string a and the experiment λ is known from the corresponding query for λ, since $C(\lambda) = a$ implies $C(a) = \lambda$. The table is not closed because $row(a)$ and $row(b)$ do not belong to $row(S)$. The algorithm adds the strings a and b to S, and extends the table (Table 5.2).

Table 5.2: $S = \{\lambda, a, b\}, E = \{\lambda\}$

T_2	λ
λ	a
a	λ
b	ba
aa	λ
ab	ba
ba	Θ
bb	a

The observation table is still not closed since $row(ba)$ is not in $row(S)$. The algorithm adds the string ba to S (see Table 5.3). Notice that the corrections for the strings baa and bab are already known, since $C(ba) = \Theta$ implies $C(baa) = C(bab) = \Theta$ (by Remark 4, Section 3.1).

Table 5.3: $S = \{\lambda, a, b, ba\}, E = \{\lambda\}$

T_3	λ	State
λ	a	q_0
a	λ	$q_1 \in F$
b	ba	q_2
ba	Θ	q_3
aa	λ	$q_1 \in F$
ab	ba	q_2
bb	a	q_0
baa	Θ	q_3
bab	Θ	q_3

In this moment, we can see that the observation table is closed and consistent and the algorithm proceeds by asking the teacher an EQ. The conjectured automaton is represented in Figure 5.1.

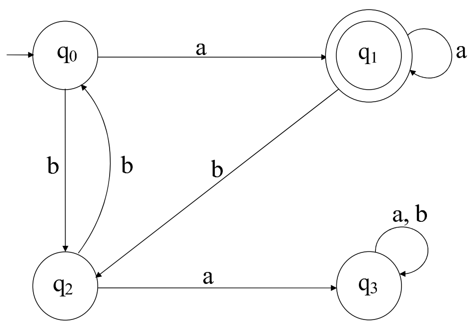


Figure 5.1: The automaton associated with Table 5.3

This is obviously not isomorphic with $\mathcal{A}_L$, and hence the teacher answers with a counterexample. Suppose that the counterexample returned is the string $bbbba$. The algorithm adds this string and all its prefixes to S and updates the table (see Table 5.4).

Table 5.4: $S = \{\lambda, a, b, ba, bb, bbb, bbbb, bbbba\}$, $E = \{\lambda\}$

T_4	λ
λ	a
a	λ
b	ba
ba	Θ
bb	a
bbb	Θ
$bbbb$	Θ
$bbbba$	Θ
aa	λ
ab	ba
baa	Θ
bab	Θ
bba	λ
$bbba$	Θ
$bbbbb$	Θ
$bbbbaa$	Θ
$bbbbab$	Θ

Table 5.5: $S = \{\lambda, a, b, ba, bb, bbb, bbbb, bbbba\}$, $E = \{\lambda, b\}$

T_5	λ	b	State
λ	a	ba	q_0
a	λ	ba	$q_1 \in F$
b	ba	a	q_2
ba	Θ	Θ	q_3
bb	a	Θ	q_4
bbb	Θ	Θ	q_3
$bbbb$	Θ	Θ	q_3
$bbbba$	Θ	Θ	q_3
aa	λ	ba	$q_1 \in F$
ab	ba	a	q_2
baa	Θ	Θ	q_3
bab	Θ	Θ	q_3
bba	λ	ba	$q_1 \in F$
$bbba$	Θ	Θ	q_3
$bbbbb$	Θ	Θ	q_3
$bbbbaa$	Θ	Θ	q_3
$bbbbab$	Θ	Θ	q_3

The current observation table is not consistent, since $row(\lambda)$ equals $row(bb)$ but $row(\lambda \cdot b)$ and $row(bb \cdot b)$ have different values. The algorithm adds the string b to E, and updates the table accordingly (see Table 5.5). The table is now closed and consistent, and the conjectured automaton $\mathcal{A}(S, E, C)$ is isomorphic with $\mathcal{A}_L$ (see Figure 5.2).

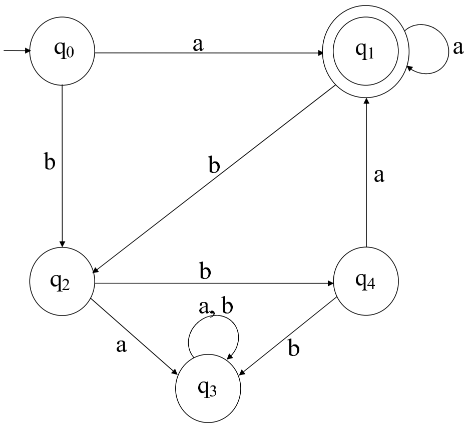


Figure 5.2: The automaton associated with Table 5.5

We notice that during the whole algorithm's execution, the learner asks only 2 EQs (the last one was successful) and 8 PCQs. Recall that for the same language and the same returned counterexample, L* asks 3 EQs and 44 MQs.

5.1.4 When does LCA perform better than L*?

Analyzing the complexity of the two algorithms we notice that in the worst case they perform the same. Nevertheless, tests performed on randomly generated DFAs indicate that this is not the case in general (the experiments are done by A.H. Dediu - the automata and the results of the tests can be found in [BB06]). In order to compare LCA and L* we focus on the query complexity, since the total running time of the two algorithms is closely related to the number of queries asked.

Intuitively, our algorithm should perform much better when the extra information provided by correcting strings is actually helping the process of learning. For example, imagine the case where all the states of the target DFA have different *correcting strings*[2], as opposed to the case in which the total number of possible correcting strings is very small compared to the number of states. Clearly, in the first one we would be able to differentiate two states by asking just one PCQ for each of them. On the other hand, the same strategy would not be as efficient in the second case. To illustrate this idea, we first introduce the notion of *injectivity degree* and then we present two extreme cases: one language with injectivity degree 0, and another one with injectivity degree 9.

For any regular language L over the alphabet Σ, we denote by $InjDeg(L)$ the *injectivity degree* of the language L, that is:

$$InjDeg(L) = index(L) - |\{C_L(u) \mid u \in \Sigma^*\}|$$

Moreover, we say that L is *injective* or that it has the *injectivity property* if $InjDeg(L) = 0$.

Remark 6. *A regular language L is injective if and only if $C_L(u) = C_L(v)$ implies $u \equiv_L v$ for all $u, v \in \Sigma^*$.*

Notice that the reverse implication always holds: $u \equiv_L v \Rightarrow C_L(u) = C_L(v)$ for all $u, v \in \Sigma^*$.

Now let us come back to our examples. In Figure 5.3, we have an automaton with 10 states, each of them having distinct correcting strings as one can see in Table 5.6.

Table 5.6: Possible corrections for strings in Σ^* w.r.t the language $\mathcal{L}(\mathcal{A}_1)$

States	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
$C_L(q)$	a	λ	$aaba$	$aaaaba$	aba	$bbaaba$	ba	$baba$	$baaba$	$aaaba$

[2]Given a DFA $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, by *correcting string of a state q* we understand the string $C_L(w_q)$ where $w_q \in \Sigma^*$ is such that $\delta(q_0, w_q) = q$.

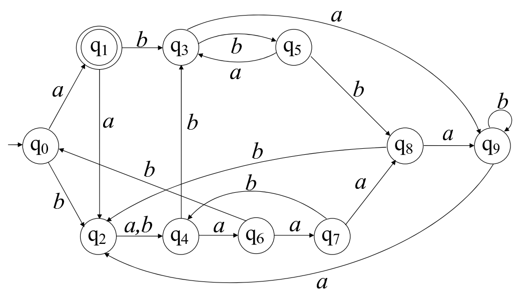


Figure 5.3: The automaton $\mathcal{A}_1$ with $\mathit{InjDeg}(\mathcal{L}(\mathcal{A}_1)) = 0$

Counting the number of queries asked by the two algorithms having as input the automaton from Figure 5.3, we obtain that LCA uses 11 PCQs and 1 EQ, and that L* asks 269 MQs and 6 EQs. We have to mention here that in our implementation the teacher always returns the shortest counterexample. Choosing to return an arbitrary long counterexample might generate unfair results: imagine that the two algorithms LCA and L* submit as an hypothesis the same automaton, and one of them gets a very long counterexample.

Note that each string is counted only once: even if the algorithm reaches a point where the learner should submit to the teacher a string which was previously submitted we will not count this as another MQ (or PCQ). Moreover, in the case of PCQs, the learner may obtain some implicit answers due to Remark 2 and Remark 4. Therefore, the algorithm does not ask these questions and we do not count them as new PCQs.

Figure 5.4 presents an automaton with 13 states which has only 4 possible correcting strings: λ, a, b and aa, as one can see in Table 5.7.

Table 5.7: Possible corrections for strings in Σ^* w.r.t the language $\mathcal{L}(\mathcal{A}_2)$

The state	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}	q_{12}
$C_L(q)$	b	λ	λ	aa	a	b	a	λ	λ	λ	λ	λ	λ

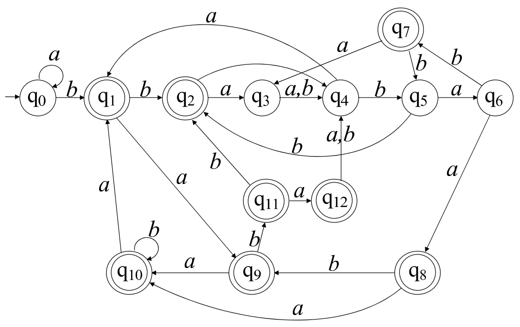


Figure 5.4: The automaton $\mathcal{A}_2$ with $\mathit{InjDeg}(\mathcal{L}(\mathcal{A}_2)) = 9$

As expected, in this case the algorithm LCA cannot make use of the embedded information normally brought by a correcting string. During its run it asks 122 PCQs and 5 EQ, slightly more than L* (113 MQs and 4 EQs).

We have seen examples showing that the smallest the injectivity degree of a language is, the better LCA works when compared with L*, and the other way around. Therefore, the next question we will be addressing is: how is the query complexity of LCA influenced by the injectivity degree of the input language?

With that in mind, let us first fix the notation. Assume that the language L we want to learn has index n and injectivity degree $\mathit{InjDeg}(L) = k$. We denote by m the length of the longest counterexample returned by the teacher, and by S_{F} and E_{F} the final configuration of the sets S and E when we run either LCA or L* on input L.

Clearly, the number of distinct elements in $(\mathrm{S}_{\mathrm{F}} \cup \mathrm{S}_{\mathrm{F}}\Sigma)\mathrm{E}_{\mathrm{F}}$ represents, on one hand, the number of MQs asked by L* when running on input L, and on the other hand, an upper bound for the number of PCQs asked by LCA. Moreover, for both algorithms, the size of S_{F} cannot exceed $n + m(n-1)$, and this is a tight upper-bound. So, if the injectivity degree plays any role in determining the number of queries asked by these algorithms, then it is the size of the set of distinguishing experiments we should be looking into.

We start by showing that the cardinality of the set E is increasing by at least one every time the teacher's answer to an EQ is `No`, which implies that the number of EQs is bounded by the cardinality of the set E_{F}. The proof is done for the algorithm LCA, but it can easily be adapted to L*.

Lemma 23. *The total number of EQs asked by LCA during its run on input L is smaller than or equal to $|\mathrm{E}_{\mathrm{F}}|$.*

Proof. Let us assume that (S, E, C) is a closed and consistent observation table such that $\mathcal{A}(S, E, C)$ is not isomorphic with $\mathcal{A}_L$. We show that if w is the string returned by the teacher as a counterexample, then the table (S', E, C) obtained by adding w and all its prefixes to S is necessarily not consistent. This implies that each EQ asked by the algorithm with the exception of the final one increases the cardinality of the set E by at least 1. Hence, the total number of EQs is smaller than or equal to $|\mathrm{E}_{\mathrm{F}}|$.

Let us first note that $w \notin S \cup S\Sigma$. Indeed, if w is in $S \cup S\Sigma$ then $\delta(q_0, w)$ equals $\mathit{row}(w)$, where $\mathcal{A}(S, E, C) = (Q, \Sigma, \delta, q_0, F)$, and hence $w \in L \Leftrightarrow C_L(w) = \lambda \Leftrightarrow \mathit{row}(w) \in F \Leftrightarrow \delta(q_0, w) \in F \Leftrightarrow w \in \mathcal{L}(\mathcal{A}(S, E, C))$ which contradicts the fact that w is in the symmetric difference of L and $\mathcal{L}(\mathcal{A}(S, E, C))$. Now, because S is a nonempty prefix-closed set and $w \notin S \cup S\Sigma$, there exist $u_0 \in S$ and $a_0, a_1, \ldots a_n \in \Sigma$ with $n \geq 1$ such that $w = u_0 a_0 a_1 \cdots a_n$ and $u_0 a_0 \in S\Sigma \backslash S$.

So, let us assume by contrary that (S', E, C) is consistent, and let S'' be the set obtained by adding to S' as many elements as it takes until the table becomes closed. Clearly, (S'', E, C) is also consistent since we only added elements with distinct row values.

We construct the sequence of strings $u_1, \ldots, u_n$ in S such that $row(u_i a_i) = row(u_{i+1})$ for all $i \in \{0, \ldots, n\}$ (recall that (S, E, C) is closed). This implies $\delta(q_0, u_i a_i) = \delta(q_0, u_{i+1})$ (by Lemma 19) for all $i \in \{0, \ldots, n\}$, so $\delta(q_0, u_{n+1}) = \delta(q_0, u_0 a_0 \cdots a_n)$. If we recall that the table (S'', E, C) is consistent (by our assumption), we get that $row(u_0 a_0) = row(u_1)$ implies $row(u_0 a_0 a_1) = row(u_1 a_1)$, and furthermore $row(u_0 a_0 a_1) = row(u_2)$. By applying a simple induction we obtain that $row(u_0 a_0 \cdots a_n) = row(u_{n+1})$, so $C_L(u_0 a_0 \cdots a_n) = C_L(u_{n+1})$.

We have $w \in L \Leftrightarrow C_L(w) = \lambda \Leftrightarrow C_L(u_{n+1}) = \lambda \Leftrightarrow \delta(q_0, u_{n+1}) \in F \Leftrightarrow \delta(q_0, w) \in F \Leftrightarrow w \in \mathcal{L}(\mathcal{A}(S, E, C))$ which again contradicts the fact that w is in the symmetric difference of L and $\mathcal{L}(\mathcal{A}(S, E, C))$. Hence, the observation table $(S'E, C)$ cannot be consistent. $\square$

Next, we show that in the case of LCA the injectivity degree can be used as an upper bound for $|\text{EF}|$.

Lemma 24. *If we run LCA on input L, the cardinality of the set* EF *is smaller than or equal to $k + 1$.*

Proof. Let $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ with $|Q| = n$ be the minimal complete DFA accepting L. Based on the final set $\text{EF} = \{e_0, e_1, \ldots, e_m\}$ $(m \geq 0)$ of distinguishing experiments (the indexing reflects the order in which these elements have been added to E; hence, $e_0 = \lambda$), we are going to construct $m + 2$ partitions of the set Q, denoted by $\rho_0, \rho_1, \ldots, \rho_{m+1}$, such that $\rho_0 = \{Q\}$ and for all $i \in \{1, \ldots, m+1\}$ we have:

- $\rho_i = \{Q_1^i, \ldots, Q_{k_i}^i\}$ for some $k_i \geq 1$, and
- two states q_1, q_2 belong to the same block of the partition ρ_i if and only if
  - q_1, q_2 are in the same block of the partition ρ_{i-1}, and
  - the states $\delta(q_1, e_{i-1})$ and $\delta(q_2, e_{i-1})$ have the same correction (i.e., $C_L(w_1) = C_L(w_2)$ for some $w_1, w_2 \in \Sigma^*$ such that $\delta(q_0, w_1) = \delta(q_1, e_{i-1})$ and $\delta(q_0, w_2) = \delta(q_2, e_{i-1})$).

We observe that:

- $1 \leq k_i \leq n$ for any $i \in \{1, \ldots, m+1\}$, because any partition of a set of n elements has at most n blocks,
- ρ_i is finer than ρ_{i-1} for all $i \in \{1, \ldots, m+1\}$, since the algorithm LCA adds e_{i-1} to the set of distinguishing experiments only when it separates

two states which could not have been distinguished with the help of the previous experiments,

- ρ_1 has exactly $n-k$ blocks, because there are $n-k$ possible correcting strings and by definition, two states q_1, q_2 are in the same block of ρ_1 if and only if they have the same correction,
- ρ_{m+1} has exactly n blocks (each of them containing one state), because for any two states q_1, q_2 in Q there exists an experiment e_j in E_F ($j \in \{0,\ldots,m\}$) such that $\delta(q_1, e_j)$ and $\delta(q_2, e_j)$ have different corrections (so, q_1 and q_2 cannot be in the same block of the partition ρ_{j+1}).

From $n-k = |\rho_1| < |\rho_2| < \ldots |\rho_{m+1}| = n$ we conclude that $m \leq k$, and hence $|\text{E}_\text{F}| \leq k+1$. □

Putting together these last two results, we get the following bounds for the number of PCQs and EQs asked by LCA.

Theorem 17. *When running on input L, LCA asks at most $k+1$ EQs and $(n+m(n-1))(|\Sigma|k+|\Sigma|-k)+k+1$ PCQs.*

Proof. The fact that the number of EQs is bounded by $k+1$ is an immediate consequence of the previous two lemmas. Now, it is clear that the number of PCQs is bounded by the number of distinct elements in $(\text{S}_\text{F} \cup \text{S}_\text{F}\Sigma)\text{E}_\text{F}$, that is, by $|\text{S}_\text{F}| + ((|\Sigma|-1)|\text{S}_\text{F}|+1) \cdot |\text{E}_\text{F}|$ according to the following observation.

Remark 7. *For any prefix-closed set S and any suffix-closed set E, we have:*

- *The number of distinct elements in $(S \cup S\Sigma)E$ is $|S| + |S\Sigma \backslash S| \cdot |E|$*
- *The number of elements of the set $S\Sigma \backslash S$ is $(|\Sigma|-1)|S|+1$*

Taking into account that $|\text{E}_\text{F}| \leq k+1$ and $|\text{S}_\text{F}| \leq n+m(n-1)$ concludes the proof. □

So, for any regular language of injectivity degree k, the final set of distinguishing experiments produced by LCA has at most $k+1$ elements, whereas the only bound for the one produced by L^* is the size of the language itself (see Example 11). On the other hand, there is no way we can bound the number of rows of the final observation table other then using the length of the longest counterexample as parameter, since the teacher is free to return arbitrarily long strings. Hence, it is only when the injectivity degree of a language is rather small in comparison with its size that one can get an important improvement in the number of queries asked by replacing MQs with PCQs (smaller tables means less queries asked).

Example 11. Take $L = \{a\} \cup \{a^i \mid i \geq 3, i \not\equiv 2(\mod 3)\}$ over the alphabet $\Sigma = \{a\}$. We have *index*$(L) = 4$, *InjDeg*$(L) = 2$ and $|\text{EF}| = 4$ when running L* on L.

Let us see how does Theorem 17 apply to classes of languages with zero injectivity degree. We fix the alphabet Σ and define $\mathcal{I}nj$ to be the class of all regular languages over the alphabet Σ that have the injectivity property. Probably the most well-known class of injective languages is the class of 0-reversible languages.

Proposition 7. *Any 0-reversible language is injective.*

Proof. Assume that there exists L a 0-reversible language such that L is not injective, i.e., there exists u, v in Σ^* such that $u \not\equiv_L v$ and $C_L(u) = C_L(v)$. Let us denote by w the correcting string of u and v with respect to the language L. Clearly, $w \neq \Theta$, since $w = \Theta$ implies $Tail_L(u) = Tail_L(v) = \emptyset$, and furthermore $u \equiv_L v$, a contradiction. So, $w \in \Sigma^*$. Moreover, since uw and vw are both in L, we obtain that $u \equiv_L v$ (by Corollary 1) which contradicts our assumption. □

We can derive the following corollaries.

Corollary 9. *For any language L in $\mathcal{I}nj$, the number of PCQs needed by LCA to learn L is $n|\Sigma| + 1$, and the number of EQs is one.*

Proof. Theorem 17 states that LCA asks $(n + m(n-1))(|\Sigma|k + |\Sigma| - k) + k + 1$ PCQs and $k + 1$ EQs while learning the language L of index n and injectivity degree k. This means that for $k = 0$, LCA asks only one EQ and $n|\Sigma| + 1$ PCQs (since the only EQ asked by the algorithm is answered in the positive way, the length m of the longest counterexample is 0). □

Corollary 10. *The class $\mathcal{I}nj$ is in PolPCorQ.*

Proof. One can modify LCA to output the conjectured automaton $\mathcal{A}(S, E, C)$ and halt when the table is closed and consistent. The restricted version of LCA for the class of injective languages is presented in Algorithm 10.

Algorithm 10 The algorithm LCA$^{\text{inj}}$ for injective languages

```
1: Initialize $S$ and $E$ with $\{\lambda\}$
2: Ask PCQs for $\lambda$ and each $a \in \Sigma$
3: Construct the initial observation table $(S, E, C)$
4: while $(S, E, C)$ is not closed do
5:    find $s$ in $S$ and $a$ in $\Sigma$ such that $row(s \cdot a) \notin row(S)$
6:    add $s \cdot a$ to $S$
7:    extend $C$ to $(S \cup S\Sigma)E$ using PCQs
8: end while
9: Construct the conjecture $\mathcal{A}(S, E, C)$
10: Halt and output $\mathcal{A}(S, E, C)$
```

Moreover, since LCA$^{\text{inj}}$ is a polynomial time algorithm, we get that the class $\mathcal{I}nj$ is in *PolPCorQ*. □

As expected, LCA performs much better than L* when running on injective languages. First of all, LCA can learn any language in $\mathcal{I}nj$ by asking only one EQs (the final one). This is not the case with L*: the class of 0-reversible languages is not polynomially learnable with MQs alone[3], which implies that L* asks more than one EQ, in general, when learning the class $\mathcal{I}nj$[4]. Secondly, the number of PCQs asked is linear in the size of the target language, and this does not happen with the MQs (see Example 12).

Example 12. Let $\mathcal{S} = (L_w)_{w \in \Sigma^*}$ be the class of all singleton languages $L_w = \{w\}$ over an arbitrary alphabet Σ. Obviously, $\mathcal{S} \subset \mathcal{I}nj$. Let us compute the number of MQs asked by L* when running on a language L_w in $\mathcal{S}$.

Lemma 25. *For any L_w in $\mathcal{S}$, the number of MQs asked by L* in order to identify L_w is at least*

$$2(|\Sigma| - 1)|w|^2 + (4|\Sigma| - 1)|w| + 2|\Sigma| + 1.$$

Proof. We give the proof only for the case $|\Sigma| = 2$. Similar reasoning applies to any other alphabets and it is left to the reader. Let us denote by n the length of w. Then, we have to show that the number of MQs asked by L* is at least $2n^2 + 7n + 5$.

Suppose the string w to be learned is $w = a_1 a_2 \cdots a_n$ with a_i in $\Sigma = \{0, 1\}$ for all $i \in \{1, 2, \ldots, n\}$. We consider only the case $n \geq 2$, because the proof for $n = 0$ and $n = 1$ is a simple exercise. The algorithm starts by constructing the observation table represented in Table 5.8.

[3]See Theorem 14 in Section 4.1.2; the result presented there is actually stronger, stating that k-reversible languages are not finitely learnable with MQs.

[4]If L* was asking just one EQ for any injective language, than a slightly modified version of L* would be a polynomial time algorithm for learning the class *0-Rev* with MQs, a contradiction.

Table 5.8: $S = \{\lambda\}, E = \{\lambda\}$

T_1	λ	State
λ	0	q_0
0	0	q_0
1	0	q_0

The automaton associated with the current observation table does not accept any string (see Figure 5.5).

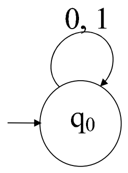


Figure 5.5: The automaton associated with the observation Table 5.8

The counterexample returned (in this case the only possible one) is the string w, so the algorithm L* adds w and all its prefixes to S and generates the observation Table 5.9 (for any letter a of the alphabet Σ we denote by $\overline{a}$ the only symbol in the set $\Sigma\backslash\{a\}$, i.e., $\overline{0}$ is 1 and $\overline{1}$ is 0).

Table 5.9: $E = \{\lambda\}, S = \{\lambda, a_1, \ldots, a_1a_2\cdots a_n\}$

T_2	λ
λ	0
a_1	0
$\vdots$	
$a_1a_2\cdots a_{n-1}$	0
$a_1a_2\cdots a_n$	1
$\overline{a}_1$	0
$a_1\overline{a}_2$	0
$\vdots$	
$a_1\cdots a_n0$	0
$a_1\cdots a_n1$	0

Table 5.10: $E = \{\lambda, a_n\}$, $S = \{\lambda, a_1, \ldots, a_1a_2\cdots a_n\}$

T_3	λ	a_n
λ	0	0
a_1	0	0
$\vdots$		
$a_1a_2\cdots a_{n-1}$	0	1
$a_1a_2\cdots a_n$	1	0
$\overline{a}_1$	0	0
$a_1\overline{a}_2$	0	0
$\vdots$		
$a_1\cdots a_n0$	0	0
$a_1\cdots a_n1$	0	0

This table is not consistent because $row(a_1a_2\cdots a_{n-2}) = row(a_1a_2\cdots a_{n-1})$ but $row(a_1a_2\cdots a_{n-2}\cdot a_n) \neq row(a_1a_2\cdots a_{n-1}\cdot a_n)$ (their values in λ do not coincide). So the algorithm adds the value a_n to the set E of experiments, and updates the table (see Table 5.10).

The new table is still not consistent, because although $row(a_1a_2\cdots a_{n-3})$ $= row(a_1a_2\cdots a_{n-2})$, $row(a_1a_2\cdots a_{n-3}\cdot a_{n-1}) \neq row(a_1a_2\cdots a_{n-2}\cdot a_{n-1})$ (the two functions have different values in a_n).

Therefore, L* adds the experiment $a_{n-1} \cdot a_n$ to E. This procedure is continued until $E = \{\lambda, a_n, a_{n-1}a_n, \cdots, a_2a_3 \cdots a_n\}$ (see Table 5.11).

Table 5.11: $S = \{\lambda, a_1, \ldots, a_1a_2 \cdots a_n\}, E = \{\lambda, a_n, a_{n-1}a_n, \ldots, a_2a_3 \cdots a_n\}$

T_{n+1}	λ	a_n	$a_{n-1}a_n$	....	$a_2a_3 \cdots a_n$	State
λ	0	0	0		0	q_0
a_1	0	0	0		1	q_1
a_1a_2	0	0	0		0	q_2
$\vdots$						
$a_1a_2 \cdots a_{n-1}$	0	1	0		0	q_{n-1}
$a_1a_2 \cdots a_n$	1	0	0		0	$q_n \in F$
$\overline{a}_1$	0	0	0		0	q_0
$a_1\overline{a}_2$	0	0	0		0	q_0
$\vdots$						
$a_1 \cdots a_{n-1}\overline{a}_n$	0	0	0		0	q_0
$a_1 \cdots a_n0$	0	0	0		0	q_0
$a_1 \cdots a_n1$	0	0	0		0	q_0

This table is closed and consistent and the learner L* constructs the automaton represented in Figure 5.6.

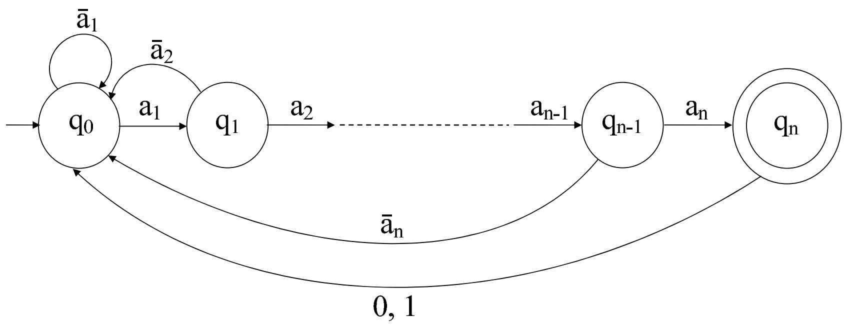


Figure 5.6: The automaton associated with the observation Table 5.11

The conjectured automaton is not the target one, so the teacher's answer to the EQ is a counterexample. Let us assume that the counterexample returned by the teacher is the string $\overline{a}_1a_1a_2 \cdots a_n$ (since this is the shortest string in the symmetric difference of $\mathcal{L}(\mathcal{A}_{L_w})$ and $\mathcal{L}(\mathcal{A}(S, E, C))$, any other counterexample would lead to even more MQs). Then, L* proceeds by adding this string and all its prefixes to S. By updating the observation table we get Table 5.12.

Table 5.12: $S = \{\lambda, \dots, \overline{a}_1 a_1 a_2 \cdots a_n\}, E = \{\lambda, \dots, a_2 a_3 \cdots a_n\}$

T_{n+2}	λ	a_n	....	$a_2 a_3 \cdots a_n$
λ	0	0		0
a_1	0	0		1
$\vdots$				
$a_1 a_2 \cdots a_n$	1	0		0
$\overline{a}_1$	0	0		0
$\overline{a}_1 a_1$	0	0		0
$\vdots$				
$\overline{a}_1 a_1 a_2 \cdots a_n$	0	0		0
$a_1 \overline{a}_2$	0	0		0
$a_1 a_2 \overline{a}_3$	0	0		0
$\vdots$				
$a_1 \cdots a_{n-1} \overline{a}_n$	0	0		0
$a_1 \cdots a_{n-1} a_n 0$	0	0		0
$a_1 \cdots a_{n-1} a_n 1$	0	0		0
$\overline{a}_1 \overline{a}_1$	0	0		0
$\overline{a}_1 a_1 \overline{a}_2$	0	0		0
$\vdots$				
$\overline{a}_1 a_1 \cdots a_{n-1} \overline{a}_n$	0	0		0
$\overline{a}_1 a_1 \cdots a_{n-1} a_n 0$	0	0		0
$\overline{a}_1 a_1 \cdots a_{n-1} a_n 1$	0	0		0

This table is not consistent because $row(\lambda) = row(\overline{a}_1)$ and $row(\lambda \cdot a_1) \neq row(\overline{a}_1 \cdot a_1)$ (they differ in the column corresponding to the experiment $a_2 a_3 \cdots a_n$). So L* adds the experiment $a_1 a_2 \cdots a_n$ to E and updates the observation table (see Table 5.13). The conjectured automaton is represented in Figure 5.7.

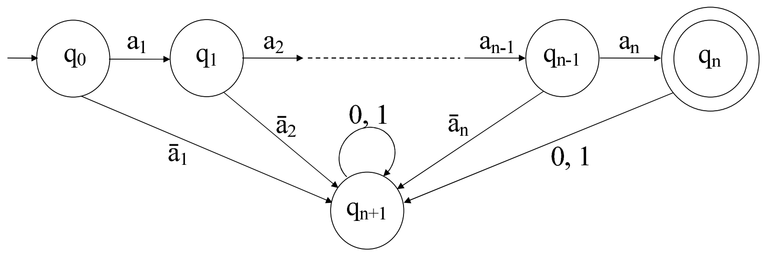


Figure 5.7: The automaton associated with the observation Table 5.13

Table 5.13: $S = \{\lambda, \ldots, \overline{a}_1 a_1 a_2 \cdots a_n\}, E = \{\lambda, \ldots, a_1 a_2 \cdots a_n\}$

T_{n+3}	λ	a_n	....	$a_2 a_3 \cdots a_n$	$a_1 a_2 \cdots a_n$	State
λ	$\underline{0}$	0		0	1	q_0
a_1	$\underline{0}$	0		1	0	q_1
$\vdots$						
$a_1 a_2 \cdots a_n$	$\underline{1}$	0		0	0	$q_n \in F$
$\overline{a}_1$	$\underline{0}$	0		0	0	q_{n+1}
$\overline{a}_1 a_1$	$\underline{0}$	0		0	0	q_{n+1}
$\overline{a}_1 a_1 a_2$	$\underline{0}$	0		0	0	q_{n+1}
$\vdots$						
$\overline{a}_1 a_1 a_2 \cdots a_n$	$\underline{0}$	0		0	0	q_{n+1}
$a_1 \overline{a}_2$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$a_1 a_2 \overline{a}_3$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\vdots$						
$a_1 \cdots a_{n-1} \overline{a}_n$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$a_1 \cdots a_{n-1} a_n 0$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$a_1 \cdots a_{n-1} a_n 1$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\overline{a}_1 \overline{a}_1$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\overline{a}_1 a_1 \overline{a}_2$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\vdots$						
$\overline{a}_1 a_1 \cdots a_{n-1} \overline{a}_n$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\overline{a}_1 a_1 \cdots a_{n-1} a_n 0$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}
$\overline{a}_1 a_1 \cdots a_{n-1} a_n 1$	$\underline{0}$	$\underline{0}$		$\underline{0}$	$\underline{0}$	q_{n+1}

One can see that the conjectured automaton is isomorphic with the target one, so the teacher's answer to the EQ will be `Yes`. All we have to do now is to count how many distinct MQs have been asked.

For this, let us first notice that only the elements that are marked as underlined in Table 5.13 should be counted. By Remark 7, the number of distinct elements in $(S \cup S\Sigma)E$ is $|S| + (|S| + 1)|E|$, that is, $2n + 2 + (2n + 3)(n + 1)$. Hence, the number of distinct MQs is $2n^2 + 7n + 5$. $\square$

In conclusion, for any language in the class $\mathcal{S}$, L* asks a quadratic number of MQs and at least 3 EQs, while LCA needs a linear number of PCQs and only one EQ (see Lemma 26).

Lemma 26. *For any L_w in $\mathcal{S}$, the number of PCQs asked by L^* in order to identify L_w is*

$$(|\Sigma| - 1)|w| + |\Sigma| + 1.$$

Proof. We give the proof only for the case $|\Sigma| = 2$. Similar arguments can be used for larger alphabets. Suppose the string w to be learned is $w = a_1 a_2 \cdots a_n$.

The algorithm starts with $S = \{\lambda\}$, $E = \{\lambda\}$ and the following observation table.

Table 5.14: $S = \{\lambda\}, E = \{\lambda\}$

T_1	λ
λ	$a_1a_2\cdots a_n$
a_1	$a_2\cdots a_n$
$\overline{a}_1$	Θ

Since $row(a_1)$ is not in $row(S)$, Table 5.14 is not closed. The algorithm proceeds by adding to the set S, one by one, the strings: a_1 , $\overline{a}_1$, $a_1a_2 \ldots a_1a_2a_{n-1}$, and finally $a_1a_2 \ldots a_n$. The corresponding observation table is Table 5.15.

Table 5.15: $S = \{\lambda, \ldots, a_1a_2\cdots a_n, \overline{a}_1\}, E = \{\lambda\}$

T_{n+2}	λ	State
λ	$a_1a_2\cdots a_n$	q_0
a_1	$a_2\cdots a_n$	q_1
$\vdots$		
$a_1\cdots a_{n-1}$	a_n	q_{n-1}
$a_1a_2\cdots a_n$	λ	$q_n \in F$
$\overline{a}_1$	Θ	q_{n+1}
$a_1\overline{a}_2$	Θ	q_{n+1}
$a_1a_2\overline{a}_3$	Θ	q_{n+1}
$\vdots$		
$a_1\cdots a_{n-1}\overline{a}_n$	Θ	q_{n+1}
$a_1\cdots a_n0$	Θ	q_{n+1}
$a_1\cdots a_n1$	Θ	q_{n+1}
$\overline{a}_10$	Θ	q_{n+1}
$\overline{a}_11$	Θ	q_{n+1}

Clearly, this last table is closed and consistent, and the conjectured automaton (the one represented in Figure 5.7) is the target one. Hence the teacher's answer to the EQ is `Yes` and the algorithm ends by outputting $\mathcal{A}_{L_w}$.

All we have to do now is count the PCQs. Some of the answers are implicit, like those for $C(a_1a_2\cdots a_i)$, for all $i \in \{1, 2, \ldots, n\}$ (from $C(\lambda) = a_1a_2\cdots a_n$ we know that $C(a_1a_2\cdots a_i) = a_{i+1}\cdots a_n$). Some basic counting shows that LCA is asking a total of $n + 3$ questions in order to identify the target language. □

5.2 Learning Deterministic Finite Automata with Length Bounded Correction Queries and Equivalence Queries

Recall that we defined the l-bounded correction of the string u with respect to L (denoted $C_L^l(u)$) as the set of all strings v of length at most l such that $u \cdot v$ is in L. Formally, C_L^l is a function from Σ^* to $\mathcal{P}(\Sigma^*)$ such that for any u in Σ^*, $C_L^l(u) = \{v \in \mathit{Tail}_L(u) \mid |v| \leq l\}$. Moreover, $\lambda \in C_L^l(u)$ if and only if $u \in L$.

So let us investigate what happens when we allow the learner to ask the teacher lBCQs and EQs.

5.2.1 The Algorithm LlBCA

Basically what we do is adapting the algorithm LCA to the new type of queries. For this, we keep S to be a nonempty prefix-closed set of strings, E a nonempty suffix-closed set of experiments, but the entry for the element in the table that is found at the intersection between row u and column e (for any u in $S\Sigma \cup S$ and e in E) is now a set instead of a simple string, namely the set $C_L^l(ue)$. So, we denote the observation table by (S, E, C^l), and for any u in S, we define a function $\mathit{row}^l(u) : E \to \mathcal{P}(\Sigma^*)$ by $\mathit{row}^l(u)(e) = C_L^l(u \cdot e)$.

The algorithm LlBCA is basically the same: as long as the table is not l-bounded closed or l-bounded consistent, it keeps adding elements to S or E, and extends the table accordingly. When both conditions are satisfied, the learner constructs its hypothesis and presents it to the teacher. If it is the correct one, the algorithm halts by outputting the conjectured automaton. Otherwise, it continues by processing the information received from the teacher (the counterexample).

An observation table (S, E, C^l) is called *l-bounded closed* if for every u in $S\Sigma$ there exists a v in S such that $\mathit{row}^l(u) = \mathit{row}^l(v)$, and *$l$-bounded consistent* if for any u_1, u_2 in S such that $\mathit{row}^l(u_1) = \mathit{row}^l(u_2)$, we have $\mathit{row}^l(u_1 \cdot a) = \mathit{row}^l(u_2 \cdot a)$ for all a in Σ.

For any l-bounded closed, l-bounded consistent observation table (S, E, C^l), we define the automaton $\mathcal{A}(S, E, C^l) = (Q, \Sigma, \delta, q_0, F)$ as follows:

- $Q = \{\mathit{row}^l(u) \mid u \in S\}$,
- $q_0 = \mathit{row}^l(\lambda)$,
- $F = \{\mathit{row}^l(u) \mid u \in S \text{ and } \lambda \in C_L^l(u)\}$,
- $\delta(\mathit{row}^l(u), a) = \mathit{row}^l(u \cdot a)$.

It is easy to see that $\mathcal{A}(S, E, C^l)$ is a well-defined automaton.

The algorithm is described in what follows.

Algorithm 11 LlBCA: Learning DFAs with lBCQs and EQs

```
 1: Initialize $S$ and $E$ with $\{\lambda\}$
 2: Ask $l$BCQs for $\lambda$ and each $a \in \Sigma$
 3: Construct the initial observation table $(S, E, C^l)$
 4: repeat
 5:   repeat
 6:     while $(S, E, C^l)$ is not $l$-bounded closed do
 7:       find $u$ in $S$ and $a$ in $\Sigma$ such that $row^l(u \cdot a) \notin \{row^l(v) \mid v \in S\}$
 8:       add $u \cdot a$ to $S$
 9:       extend $C_L^l$ to $(S \cup S\Sigma)E$ using $l$BCQs
10:     end while
11:     while $(S, E, C^l)$ is not $l$-bounded consistent do
12:       find $u_1$,$u_2 \in S$ and $a \in \Sigma$, $e \in E$ such that $row^l(u_1) = row^l(u_2)$ and
          $C_L^l(u_1 \cdot a \cdot e) \neq C_L^l(u_2 \cdot a \cdot e)$
13:       add $a \cdot e$ to $E$
14:       extend $C_L^l$ to $(S \cup S\Sigma)E$ using $l$BCQs
15:     end while
16:   until $(S, E, C^l)$ is $l$-bounded closed and $l$-bounded consistent
17:   Construct the conjecture $\mathcal{A}(S, E, C^l)$
18:   if the teacher replies with a counterexample $u$ then
19:     add $u$ and all its prefixes to $S$
20:     extend $C_L^l$ to $(S \cup S\Sigma)E$ using $l$BCQs
21:   end if
22: until the teacher replies Yes to the conjecture
23: Halt and output $\mathcal{A}(S, E, C^l)$
```

5.2.2 Correctness, Termination and Running Time

In order to prove that the algorithm terminates by outputting the correct DFA, we basically follow the same steps we did for LCA. First, we need to specify what conditions have to be met by an automaton to be consistent with the values of a given observation table.

Assume that (S, E, C^l) is an l-bounded closed and l-bounded consistent observation table. We say that the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is *l-bounded consistent with the function C_L^l with respect to* (S, E, C^l) if for every u in $S \cup S\Sigma$ and e in E,

$$C_L^l(u \cdot e) = \{v \in \Sigma^* \mid |v| \leq l, \delta(q_0, u \cdot e \cdot v) \in F\}.$$

The following theorem is the key to proving that regular languages can be learned in polynomial time using lBCQs and EQs.

Theorem 18. *If (S, E, C^l) is an l-bounded closed and l-bounded consistent observation table, then the automaton $\mathcal{A}(S, E, C^l)$ is l-bounded consistent with the finite function C^l_L. Any other automaton l-bounded consistent with C^l_L but not equivalent to $\mathcal{A}(S, E, C^l)$ must have more states.*

The theorem is proved by the following lemmas.

Lemma 27. *Let (S, E, C^l) be an l-bounded closed and l-bounded consistent observation table. For the automaton $\mathcal{A}(S, E, C^l)$ and every u in $S \cup S\Sigma$, $\delta(q_0, u) = row^l(u)$.*

Proof. The proof is by induction on the length of u. □

Lemma 28. *If (S, E, C^l) is an l-bounded closed and l-bounded consistent observation table and $\mathcal{A}(S, E, C^l) = (Q, \Sigma, \delta, q_0, F)$, then for each u in $S \cup S\Sigma$ and all $e \in E$, there exists v in S such that $\delta(q_0, u \cdot e) = \delta(q_0, v)$ and $C^l_L(u \cdot e) = C^l_L(v)$.*

Proof. Let u be a string from $S \cup S\Sigma$. Because (S, E, C^l) is l-bounded closed, there exists $v \in S$ such that $row^l(v) = row^l(u)$, and hence $\delta(q_0, v) = \delta(q_0, u)$ (by Lemma 28) and $C^l_L(v \cdot e) = C^l_L(u \cdot e)$ for all e in E.

The proof is by induction on the length of e.

- If $e = \lambda$, it is clear that $\delta(q_0, u \cdot \lambda) = \delta(q_0, v)$ and $C^l_L(u \cdot \lambda) = C^l_L(v)$.
- Now suppose the result holds for all e in E of length at most k, and let e be an element of E of length $k+1$. Since E is suffix-closed, $e = a \cdot e'$ for some a in Σ and e' in E. From $\delta(q_0, u) = \delta(q_0, v)$ we get $\delta(q_0, u \cdot e) = \delta(q_0, v \cdot a \cdot e')$.

  Applying the induction hypothesis on e' for the string va in $S \cup S\Sigma$ we obtain that there exist v' in S such that $\delta(q_0, va \cdot e') = \delta(q_0, v')$ and $C^l_L(va \cdot e') = C^l_L(v')$. Thus, $\delta(q_0, u \cdot e) = \delta(q_0, v')$ and $C^l_L(u \cdot e) = C^l_L(v')$.

We have shown that for any u in S and any e in E there exists v' in S such that $\delta(q_0, u \cdot e) = \delta(q_0, v')$ and $C^l_L(u \cdot e) = C^l_L(v')$ which concludes our proof. □

Lemma 29. *Let (S, E, C^l) be an l-bounded closed and l-bounded consistent observation table. Then $\mathcal{A}(S, E, C^l)$ is l-bounded consistent with the function C^l_L with respect to (S, E, C^l).*

Proof. Let $\mathcal{A}(S, E, C^l) = (Q, \Sigma, \delta, q_0, F)$. We have to show that for any u in $S \cup S\Sigma$ and e in E, $C^l_L(u \cdot e) = \{v \in \Sigma^* \mid |v| \leq l, \delta(q_0, u \cdot e \cdot v) \in F\}$. From Lemma 28 we know that there exists u_0 in S such that $\delta(q_0, u \cdot e) = \delta(q_0, u_0)$ and $C^l_L(u \cdot e) = C^l_L(u_0)$. Hence, it is enough to prove that given an arbitrary u_0 in S, the following equality holds: $C^l_L(u_0) = \{v \in \Sigma^* \mid |v| \leq l, \delta(q_0, u_0 \cdot v) \in F\}$.

Because the table is l-bounded closed, for any symbol a_i in Σ^*, $i \in \{1, \ldots, n\}$ we can inductively find the strings $u_i \in S$ such that $row^l(u_{i-1} \cdot a_i) = row^l(u_i)$

(by Lemma 28), and hence $\delta(q_0, u_{i-1} \cdot a_i) = \delta(q_0, u_i)$ and $C_L^l(u_{i-1} \cdot a_i) = C_L^l(u_i)$. Clearly, $\delta(q_0, u_0 a_1 a_2 \cdots a_n) = \delta(q_0, u_n)$. It is straightforward to show that the string $a_1 a_2 \cdots a_n$ is in $C_L^l(u_0)$ if and only if λ is in $C_L^l(u_n)$ and $n \leq l$.

Now, if we denote the string $a_1 a_2 \cdots a_n$ by w, what we obtained can be summarized as follows: $w \in C_L^l(u_0) \Leftrightarrow (\lambda \in C_L^l(u_n)$ and $|w| \leq l) \Leftrightarrow (row^l(u_n) \in F$ and $|w| \leq l) \Leftrightarrow (\delta(q_0, u_n) \in F$ and $|w| \leq l) \Leftrightarrow (\delta(q_0, u_0 \cdot w) \in F$ and $|w| \leq l) \Leftrightarrow w \in \{v \in \Sigma^* \mid |v| \leq l, \delta(q_0, u_0 \cdot v) \in F\}$. If we take the beginning and the end of this series of equivalences we get $C_L^l(u_0) = \{v \in \Sigma^* \mid |v| \leq l, \delta(q_0, u \cdot e \cdot v) \in F\}$, which concludes our proof. □

Lemma 30. *Let (S, E, C^l) be an l-bounded closed and l-bounded consistent observation table. Suppose that $\mathcal{A}(S, E, C^l)$ has n states. If $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ is any automaton l-bounded consistent with C_L^l with respect to (S, E, C^l) that has n or fewer states, then $\mathcal{A}'$ is isomorphic with $\mathcal{A}(S, E, C^l)$.*

Proof. As in Lemma 22 from Chapter 5 we define the relation $\phi \subseteq Q \times Q'$ such that $row^l(u)\ \phi\ q'$ if and only if $q' = \delta'(q_0', u)$. We will first prove that ϕ is an injection. Indeed, for any $u_1, u_2 \in S$ such that $row^l(u_1)\ \phi\ q'$, $row^l(u_2)\ \phi\ q'$, we have $\delta'(q_0', u_1) = q' = \delta'(q_0', u_2)$, and hence $\delta'(q_0', u_1 \cdot e \cdot v) = \delta'(q_0', u_2 \cdot e \cdot v)$ for all e in E and v in Σ^*. Because $\mathcal{A}'$ is l-bound consistent with C_L^l, it follows immediately that $C_L^l(u_1 \cdot e) = C_L^l(u_2 \cdot e)$ for all e in E, which implies $row^l(u_1) = row^l(u_2)$.

One may show that ϕ is an automata isomorphism in exactly the same way we did for the above mentioned Lemma 22, so we will omit this proof here. □

We have shown that as soon as the table (S, E, C^l) is l-bounded closed and l-bounded consistent, $\mathcal{A}(S, E, C^l)$ is the minimal automaton consistent with C_L^l with respect to the given observation table. Combining this with the fact that the number of distinct rows in S (and hence the size of $\mathcal{A}(S, E, C^l)$) is necessarily increasing until it reaches the size of the target automaton (for the same reasons discussed in Section 5.1.2), we obtain that the algorithm LlBCA terminates in finite steps by outputting the target automaton.

The total running time of LlBCA is bounded by a polynomial in n and m where n is the size of the target language and m the length of the longest counterexample returned by the teacher. The proof for this fact is very similar to the one used for LCA, so we will omit it here.

One important difference though is that the observation table in this setting contains sets instead of strings (0's and 1's for L*, strings in $\Sigma \cup \{\Theta\}$ for LCA). So a natural question is how it affects the total running time of the algorithm.

Let us first notice that the set $C_L^l(w)$ has at most $c = (|\Sigma|^{l+1} - 1)/(|\Sigma| - 1)$ strings (one of length zero, Σ of length 1,..., $|\Sigma|^l$ strings of length l) for any w in

Σ^* and any regular language L over the alphabet Σ. So, the size of $C_L^l(w)$ might be exponential in l for some w in Σ^*. But since l is fixed for our algorithm, having to deal with sets instead of strings might increase the total running time of the algorithm by a constant $c = 1 + |\Sigma| + \cdots + l|\Sigma|^l$, which does not change the polynomiality of LlBCA. Of course this argument works when the target class contains languages of size arbitrarily big. If we apply LlBCA for, let us say, all DFAs of size l, then l is no longer a constant here and the algorithm is definitely not polynomial in l.

5.2.3 Running Example

In order to explain how this algorithms works, we will track its run on the same example we used for LCA. So, assume the language we want to learn is $L = (a + bba)^+$ and let us say take $l = 3$.

The algorithm starts with $S = \{\lambda\}, E = \{\lambda\}$, and the following observation table.

Table 5.16: $S = \{\lambda\}, E = \{\lambda\}$

T_1	λ
λ	$\{a, aa, aaa, bba\}$
a	$\{\lambda, a, aa, aaa, bba\}$
b	$\{ba, baa\}$

This table is clearly not l-bounded closed, since both $row^3(a)$ and $row^3(b)$ are different from $row^3(\lambda)$. The algorithm proceeds by adding a and b to S, and updating the observation table (see Table 5.17).

Table 5.17: $S = \{\lambda, a, b\}, E = \{\lambda\}$

T_2	λ
λ	$\{a, aa, aaa, bba\}$
a	$\{\lambda, a, aa, aaa, bba\}$
b	$\{ba, baa\}$
aa	$\{\lambda, a, aa, aaa, bba\}$
ab	$\{ba, baa\}$
ba	$\emptyset$
bb	$\{a, aa, aaa\}$

Since the table is still not l-bounded closed (note that $row^3(ba)$ and $row^3(bb)$ are not in $row^3(S)$), two more string are added to S (see Table 5.18).

Table 5.18: $S = \{\lambda, a, b, ba, bb\}, E = \{\lambda\}$

T_3	λ	State
λ	$\{a, aa, aaa, bba\}$	q_0
a	$\{\lambda, a, aa, aaa, bba\}$	$q_1 \in F$
b	$\{ba, baa\}$	q_2
ba	$\emptyset$	q_3
bb	$\{a, aa, aaa\}$	q_4
aa	$\{\lambda, a, aa, aaa, bba\}$	$q_1 \in F$
ab	$\{ba, baa\}$	q_2
baa	$\emptyset$	q_3
bab	$\emptyset$	q_3
bba	$\{\lambda, a, aa, aaa, bba\}$	$q_1 \in F$
bbb	$\emptyset$	q_3

The observation table is now l-bounded closed and l-bounded consistent, and one may notice that the conjectured automaton (represented in Figure 5.8) is isomorphic to $\mathcal{A}_{L_w}$.

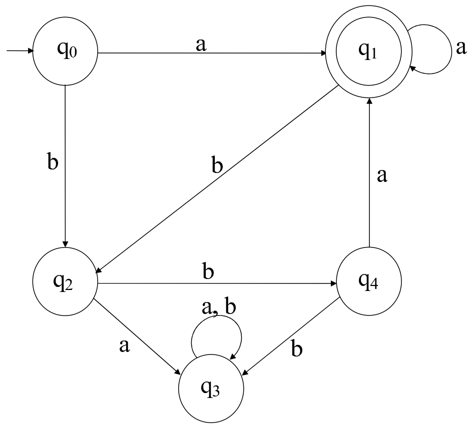


Figure 5.8: The automaton associated with Table 5.18

It is easy to check that the algorithm asks a total number of eleven 3BCQs and only one EQ. Of course, choosing a smaller value for l would increase the number of queries asked, as one can see in the following table.

Algorithm	CQs	EQs
L2BCA	11	1
L1BCA	40	3
L0BCA = L*	44	3
LCA	8	2

5.2.4 Comparison between LlBCA, LCA and L*

Recall that the time complexity of the algorithm LCA records a serious improvement when compared to L* for classes of languages with small injectivity degree. In this section we define a similar measure for lBCQs, namely the *l-injectivity degree*, and perform a similar investigation for classes of languages having the *l-injectivity property*.

For any regular language L over the alphabet Σ, we denote by $lInjDeg(L)$ the *l-injectivity degree* of the language L, that is:

$$lInjDeg(L) = index(L) - |\{C^l_L(u) \mid u \in \Sigma^*\}|$$

Note that $\{C^l_L(u) \mid u \in \Sigma^*\}$ is a set of sets, so what we count in $|\{C^l_L(u) \mid u \in \Sigma^*\}|$ is the number of distinct sets. Moreover, we say that the language L has the *l-injectivity property* or that it is *l-injective* if $lInjDeg(L) = 0$. Let us denote by $l\mathcal{I}nj$ the class of all l-injective regular languages.

Remark 8. *Let L be a regular language. The following properties are satisfied.*

1. *L is l-injective if and only if $C^l_L(u_1) = C^l_L(u_2)$ implies $u_1 \equiv_L u_2$ for all u_1, u_2 in Σ^*.*
2. *$(l-1)\mathcal{I}nj \subsetneq l\mathcal{I}nj$ for all $l \geq 1$.*
3. *If L is in $0\mathcal{I}nj$ then $index(L) \leq 2$.*

Theorem 19. *For any language L in $l\mathcal{I}nj$, the number of lBCQs needed by LlBCA in order to learn L is linear in the size of $\mathcal{A}_L$ and the number of EQs is one.*

Proof. For the two trivial languages, $L = \emptyset$ and $L = \Sigma^*$ the proof is immediate. Let $L \neq \Sigma^*$ be a nonempty arbitrary language in $l\mathcal{I}nj$ of index $n > 1$, and $\mathcal{A}_L = (Q, \Sigma, q_0, \delta, F)$ the minimal complete DFA accepting L.

We show by induction that at any step $k < n$ of the algorithm, the table (S, E, C^l) has the following properties:
- $E = \{\lambda\}$
- $|S| = k$
- (S, E, C^l) is l-bounded consistent
- (S, E, C^l) is not l-bounded closed

For $k = 1$ it is clear that $E = \{\lambda\}$, $S = \{\lambda\}$ (and hence $|S| = k$) and the table is l-bounded consistent (since it has only one element). To prove that it is not l-bounded closed, we assume the contrary. (S, E, C^l) is l-bounded closed implies that $C^l_L(a) = C^l_L(\lambda)$, $\forall a \in \Sigma$ which corresponds to $a \equiv_L \lambda$, $\forall a \in \Sigma$ and hence $\delta(q_0, a) = \delta(q_0, \lambda) = q_0$, $\forall a \in \Sigma$. This means that the target automaton has only one state, which contradicts the non triviality of L.

Suppose that the result holds for all steps strictly smaller than k, and we want to prove that the four conditions hold at the step k ($k < n$). Since at step $k-1$ the set S had $k-1$ strings and the table was l-bounded consistent and not l-bounded closed, it means that at step k the set S has one more element (and hence k elements), E continues to be $\{\lambda\}$ and the table remains l-bounded consistent (since the algorithm proceeded by adding to S a distinct line). The only thing that needs to be shown is that (S, E, C^l) is not l-bounded closed.

Assume by contradiction that (S, E, C^l) is l-bounded closed. Then LlBCA constructs the conjectured automaton $\mathcal{A}(S, E, C^l) = (Q', \Sigma, q'_0, \delta', F')$ as described in Section 5.2.1. Let us define the function $\phi : Q' \to Q$, by $\phi(row^l(u)) = \delta(q_0, u)$. We show that ϕ is well defined, injective, that $\phi(q'_0) = q_0$, $\phi(F') \in F$ and $\phi(\delta'(row^l(u), a)) = \delta(\phi(row^l(u)), a)$, for all u in S and a in Σ.

1. ϕ is clearly well defined, since there are no two strings $u_1 \neq u_2$ in S such that $row^l(u_1) = row^l(u_2)$.

2. To see that ϕ is injective, lets take two distinct states in Q', $row^l(u_1)$ and $row^l(u_2)$. Because $E = \{\lambda\}$, $row^l(u_1) \neq row^l(u_2) \Leftrightarrow C^l_L(u_1) \neq C^l_L(u_2)$ and since L has the injectivity property, this is equivalent to $u_1 \not\equiv_L u_2 \Leftrightarrow \delta(q_0, u_1) \neq \delta(q_0, u_2) \Leftrightarrow \phi(row^l(u_1)) \neq \phi(row^l(u_2))$, which implies that ϕ is injective.

3. $\phi(q'_0) = \phi(row^l(\lambda)) = \delta(q_0, \lambda) = q_0$.

4. Let us take q in $\phi(F')$. This means that there exists u in S such that $row^l(u) \in F'$ and $\phi(row^l(u)) = q$, which is equivalent to $\lambda \in C^l_L(u)$ and $\delta(q_0, u) = q$. Hence, q is in F.

5. $\delta'(row^l(u), a) = row^l(u \cdot a) = row^l(v)$, where $v \in S$ and hence $\phi(\ \delta'(\ row^l(u),\ a)) = \phi(row^l(v)) = \delta(q_0, v)$. But $\delta(q_0, v) = \delta(q_0, u \cdot a)$ because $u \cdot a \equiv_L v$, since $C^l_L(u \cdot a) = C^l_L(v)$ and the language L is l-injective. So, $\phi(\delta'(row^l(u), a)) = \delta(\delta(q_0, u), a) = \delta(\phi(row^l(u)), a)$.

Clearly, $\mathcal{A}(S, E, C^l)$ is a complete automaton. Now it is enough to see that we have constructed an injective morphism between two complete automata such that $|Q'| = k < n = |Q|$, which leads to a contradiction. $\square$

Note that there are only a finite number of l-injective languages (any language in $l\mathcal{I}nj$ has at most $2^{1+|\Sigma|+\cdots+|\Sigma|^l}$ states), so the total running time of the algorithm might be exponential in the size of the target language as in the following example.

Example 13. Let us consider the class $\bar{\mathcal{S}}_l = (\bar{L}_w)_{w \in \Sigma^l}$ with $\bar{L}_w := \Sigma^* \backslash \{w\}$. For any language L in $\bar{\mathcal{S}}_l$ we have: $L \in l\mathcal{I}nj \backslash (l-1)\mathcal{I}nj$, $index(L) = l+2$, and for each u in Σ^*, the number of elements of the set $C^l_{\bar{L}}(u)$ is at least $|\Sigma^{\leq l}| - 1$, that is $|\Sigma|(|\Sigma|^l - 1)/(|\Sigma| - 1)$. So just to compare two rows in the observation table we need exponential time in the size of $\mathcal{A}_L$.

Corollary 11. *The class $l\mathcal{I}nj$ is learnable with lBCQs.*

Proof. The same argument as for the case of injective languages holds: if we need only one EQs (which confirms that the conjectured automaton is the target one), then we do not have to ask this question at all. □

An algorithm similar to LCA$^{\text{inj}}$ can be constructed for l-injective languages. This algorithm will use only lBCQs.

Algorithm 12 The algorithm LlBCA$^{\text{inj}}$ for l-injective languages

```
1: Initialize $S$ and $E$ with $\{\lambda\}$
2: Ask $l$BCQs for $\lambda$ and each $a \in \Sigma$
3: Construct the initial observation table $(S, E, C^l)$
4: while $(S, E, C^l)$ is not $l$-bounded closed do
5:    find $u$ in $S$ and $a$ in $\Sigma$ such that $row^l(u \cdot a) \notin row^l(S)$
6:    add $u \cdot a$ to $S$
7:    extend $C^l_L$ to $(S \cup S\Sigma)E$ using $l$BCQs
8: end while
9: Construct the conjecture $\mathcal{A}(S, E, C^l)$
10: Halt and output $\mathcal{A}(S, E, C^l)$
```

Please note that although LlBCA$^{\text{inj}}$ uses only a linear number of queries, its total running time is not polynomial in general (see the language class from Example 13). But this is not at all surprising if we take a closer look at our previous results:

- $PolMemQ = PolLBCorQ$,
- $\bar{\mathcal{S}}_l$ is not polynomial time learnable with MQs (hence $\bar{\mathcal{S}}_l \notin PolLBCorQ$),
- $\bar{\mathcal{S}}_l \subset l\mathcal{I}nj$ (hence $l\mathcal{I}nj \notin PolLBCorQ$).

To conclude, by replacing MQs with LBCQs in L* we only get an improvement in the number of queries since the total running time of the new algorithm LlBCA is heavily dependent on the size of the correcting sets.

So far we introduced two language classes for which the algorithms LCA and LlBCA need only a linear number of PCQs and LBCQs, respectively, and only one EQ. In the sequel we show how are these two language classes related.

Proposition 8. *For any L in $\mathcal{I}nj$, there exists l such that $L \in l\mathcal{I}nj$.*

Proof. Let L be an arbitrary language in $\mathcal{I}nj$. If l is the size of $\mathcal{A}_L$, we show that $L \in l\mathcal{I}nj$, that is $u_1 \equiv_L u_2 \Leftrightarrow C^l_L(u_1) = C^l_L(u_2)$. Since one implication always holds, we have to prove only that $C^l_L(u_1) = C^l_L(u_2)$ implies $u_1 \equiv_L u_2$. But this is clear if we notice that from $C^l_L(u_1) = C^l_L(u_2)$ we get $C_L(u_1) = C_L(u_2)$ (the correction of any string cannot be longer than the size of the language). □

So, $\mathcal{I}nj \subseteq \bigcup_{l\geq 1} l\mathcal{I}nj$. To see that the inclusion is strict, consider the language $L_{inj} = [(a+b+bb)a(a+b)]^*[(a+b+bb)a + (a+bb)b(a+b)^*]$ with $\mathcal{A}_{L_{inj}}$ represented in Figure 5.9.

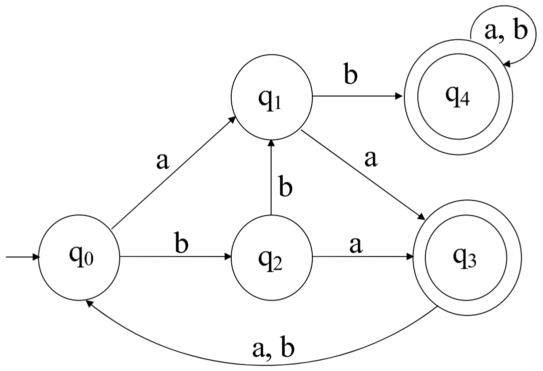


Figure 5.9: The minimal complete DFA for the language L_{inj}

Clearly, L_{inj} is not an injective language since the strings a and b have the same prefix correcting string a although they are not in the same equivalence class). Moreover, for all $l \geq 1$, $L_{inj} \in l\mathcal{I}nj$. Hence, $L_{inj} \in l\mathcal{I}nj \backslash \mathcal{I}nj$ for all $l \geq 1$.

On the other hand, we show that for any $l \geq 1$, the sets $\mathcal{I}nj$ and $l\mathcal{I}nj$ are incomparable.

Proposition 9. *For any $l \geq 0$, there exists L in $\mathcal{I}nj$ such that $L \notin l\mathcal{I}nj$.*

Proof. Let us take $l \geq 0$ arbitrary, and consider the language $L^l_{inj} = \Sigma^{\geq l+2}$ over the alphabet $\Sigma = \{a, b\}$ with $\mathcal{A}_{L^l_{inj}}$ represented in Figure 5.10.

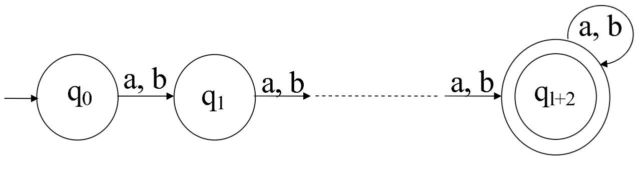


Figure 5.10: The minimal complete DFA for the language L^l_{inj}

Notice that this language has as many prefix correcting strings as states in its minimal DFA: $C_{L^l_{inj}}(a^i) = a^{l+2-i}$ for any $i \in \{0, 1, \ldots, l+2\}$. Hence, L^l_{inj} is injective. Moreover, $C^l_{L^l_{inj}}(\lambda) = C^l_{L^l_{inj}}(a) = \emptyset$ although $\lambda \not\equiv_{L^l_{inj}} a$. So, L^l_{inj} is not l-injective. □

An intuitive picture displaying the relation between the two language classes is presented in Figure 5.11.

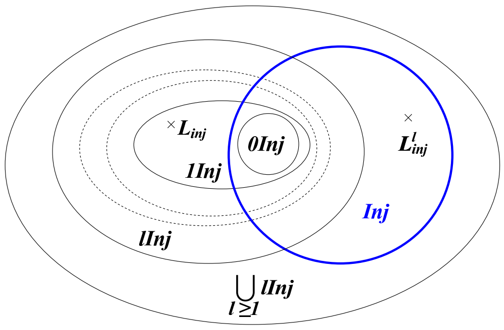


Figure 5.11: Comparison between $\mathcal{I}nj$ and $l\mathcal{I}nj$

Getting back to the comparison LlBCA versus L*, there is a clear trade-off between the number of queries asked by LlBCA and the size of the answers returned by the teacher, that is, for bigger values of l, LlBCA asks less queries than L*, but with the price of increasing observation table entries.

On the other hand, when comparing LCA with LlBCA, there is no unique answer for the question of which one is the best, or when is one of them performing better than the other. What we can say though is that for *big* values of l (a relative notion, heavily depending on the target language), LlBCA asks less queries than LCA. But, the size of the entries of the observation table might be exponentially bigger (or, they can as well be small enough).

5.3 Remarks and Further Research

This chapter was dedicated to the question "how can correction queries help in the identification of DFAs?". The reader might have noticed that, among the three types of CQs previously discussed, we left out from our study the edit distance based type of queries. The reason we did so is that, although they seem to be the most adequate for learning natural languages, for the case of DFAs they do not help too much, and we are going to detail the whys and wherefores in the sequel. Of course they can be used, to the same extent as MQs, to determine whether or not a string belongs to the target language, but the extra information provided by the teacher does not help otherwise: note that learning DFAs basically means learning a finite number of equivalence classes. And, whereas in the case of PCQs and LBCQs, the possible corrections form a finite set, the set of all possible edit distance correcting strings might be infinite. Moreover, in many cases, two strings belonging to the same equivalence class have different corrections as in the following example.

Example 14. Take $\Sigma = \{a\}$ and $L = (aa)^*$. For any string $w = a^{2k+1}$, $\textsc{Edc}_L(w)$ is either a^{2k} or a^{2k+2}, which means that any string in L can be a possible correction. Furthermore, the distance between $\textsc{Edc}_L(a^3)$ and $\textsc{Edc}_L(a^{205})$ is at least 200 although a^3 and a^{205} are equivalent with respect to L.

Therefore, the only helpful information we get from an EDCQ oracle when the target is a DFA is whether or not the given string is in the language, and for this purpose an MQ oracle suffices.

On the other hand, LBCQs are more useful than MQs only if our unique goal is to reduce the number of interactions between the teacher and the learner, and we are not worried about space or time limitations. We have seen though that the teacher's burden might be quite high: recall that if the class to be learned is $\bar{\mathcal{S}} = (\bar{L}_w)_{w\in\Sigma^*}$ with $\bar{L}_w = \Sigma^*\backslash\{w\}$, then any queried string would receive as answer a set with at least $|\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^l$ elements. Moreover, unless we are lucky to "guess" a prefix of w, we would get a lot of useless answers: $C^l_{\bar{L}_w}(v) = \Sigma^{\leq l}$ for any $v \notin \mathit{Pref}(\{w\})$.

So, among the three types of correction queries proposed, the only ones that make a difference and can be successfully used for DFA identification, together with EQs, are the PCQs. With very few exceptions (which are rather extreme cases, like the one presented in Figure 5.4), PCQs outperforms his "rivals". And even in those extreme cases mentioned earlier, the difference between the number of MQs and PCQs is negligible.

There are various related research topics that deserve further investigation. For example, instead of restricting the concept to be learned to the class of DFAs, one may approach a broader perspective and study the conditions that have to be met by an arbitrary language class in order to be learnable with a polynomial number of CQs and EQs. This has already been done for the combination of MQs and EQs[5]. Or, one could follow Angluin [Ang04] and establish bounds for the number of CQs, or CQs and (proper) EQs needed to learn a class of concepts. Moreover, it would be interesting to check whether the classes of languages learnable with a finite number of CQs and a bounded number of EQs form an infinite hierarchy, as suggested by the result of Theorem 17 in Section 5.1.4.

[5]Hellerstein et al. show that a target class is polynomial query learnable with MQs and EQs if and only if it has *polynomial certificates* (see [HPRW96] for details).

Chapter 6

Learning Regular Tree Languages with Structural Correction Queries

and Equivalence Queries In the Chomsky hierarchy, context-free grammars (CFGs) are the ones that best approximate the structure of natural languages. But context-free languages (CFLs) seem to be hard to learn (see [dlHO06], for example). Back in the 90's, Sakakibara has the idea to use the structure of the parse trees as supplementary information, showing that CFLs are learnable with structured membership queries and EQs. Twenty years later, Drewes and Högberg [DH03, DH07] improve Sakakibara's algorithm, generalizing L^* to regular tree languages (RTLs). The interest in learning regular tree languages is justified by the nice relationship that exists between these languages and CFLs: the yield of any regular tree language is a CFL. On the other hand, trees have proved to be very useful in other fields of natural language processing as well.

In this chapter, we introduce the notion of *structural correction queries* (SCQs), thus adapting correction queries to trees, and we show that RTLs are learnable with SCQs and EQs.

Let us fix Δ to be an arbitrary ranked alphabet, and $T \subseteq \mathbb{T}_\Delta$ the regular tree language to be learned. The EQs are defined as usual: given a tree recognizer $\mathcal{R}$, the teacher will check whether $\mathcal{R}$ is equivalent to $\mathcal{R}_T$. If the answer is `No`, then a tree *counterexample* $t \in (\mathcal{T}(\mathcal{R})\backslash T) \cup (T\backslash \mathcal{T}(\mathcal{R}))$ (in the symmetric difference of $T(\mathcal{R})$ and T) is returned. The answer to a SCQ is a *correcting context*. The *correcting context* of a tree $t \in \mathbb{T}_\Delta$ with respect to the tree language T and the Knuth-Bendix order $<_{kbo}$ on $\mathbb{C}_\Delta$, denoted $Cor_T(t)$, is the minimal context of the set $Fron_T(t)$. In case that no such context exists (i.e., $Fron_T(t) = \emptyset$) we say

$Cor_T(t) = \Theta$, where Θ is a symbol which does not belong to $\Delta \cup \{\xi\}$. Hence, Cor_T is a function from $\mathbb{T}_\Delta$ to $\mathbb{C}_\Delta \cup \{\Theta\}$. Note that $Cor_T(t) = \xi$ if and only if t is in T, and for all $t_1, t_2 \in \mathbb{T}_\Delta$, $t_1 \cong_T t_2$ implies $Cor_T(t_1) = Cor_T(t_2)$, but the converse does not hold.

Remark 9. *If $Cor_T(t) = c_1 \cdot c_2$, then $Cor_T(t \cdot c_1) = c_2$ for any $t \in \mathbb{T}_\Delta$ and $c_1, c_2 \in \mathbb{C}_\Delta$.*

Proof. If $Cor_T(t) = c_1 \cdot c_2$, then $c_1 \cdot c_2$ is the smallest context in $\mathbb{C}_\Delta$ such that $t \cdot c_1 \cdot c_2 \in T$, and hence $c_2 \in Fron_T(t \cdot c_1)$. But c_2 must be the smallest context with this property since otherwise we reach a contradiction with the minimality of $c_1 \cdot c_2$ (by Remark 1). We conclude that $Cor_T(t \cdot c_1) = c_2$. □

Remark 10. *If t is a tree in $\mathbb{T}_\Delta$ such that $Fron_T(t) = \emptyset$, then $Fron_T(t \cdot c) = \emptyset$ for all contexts c in $\mathbb{C}_\Delta$.*

Proof. Suppose by contrary that there exists a context $c \in \mathbb{C}_\Delta$ such that $Fron_T(t \cdot c) \neq \emptyset$. Then for any $c' \in Fron_T(t \cdot c)$, we have $t \cdot c \cdot c' \in T$, and hence $c \cdot c' \in Fron_T(t)$. We reach a contradiction with $Fron_T(t) = \emptyset$. □

Remark 11. *For any tree t in $\mathbb{T}_\Delta$, the following statements hold:*

1. *If $Cor_T(t) \neq \Theta$, then $Cor_T(t \cdot Cor_T(t)) = \xi$.*
2. *If $Cor_T(t) = \Theta$, then $Cor_T(t \cdot c) = \Theta$ for all $c \in \mathbb{C}_\Delta$, but the existence of a context $c \in \mathbb{C}_\Delta$ with $Cor_T(t \cdot c) = \Theta$ does not imply that $Cor_T(t) = \Theta$.*

Proof. Let t be an arbitrary tree in $\mathbb{T}_\Delta$.

1. If $Cor_T(t) = c \neq \Theta$, then $t \cdot c \in T$ which implies $t \cdot c \cdot \xi \in T$, and hence $\xi \in Fron_T(t \cdot c)$. Because ξ is the smallest possible context, we obtain immediately that $Cor_T(t \cdot Cor_T(t)) = \xi$.
2. If $Cor_T(t) = \Theta$, then $Fron_T(t) = \emptyset$ which implies, using Remark 10, $Fron_T(t \cdot c) = \emptyset$ for all $c \in \mathbb{C}_\Delta$, and hence $Cor_T(t \cdot c) = \Theta$. Let us now consider the recognizer $\mathcal{R} = (Q, \Delta, \delta, F)$ from Example 6. If we take $t = f(a, g(a, b), b)$ and $c = g(\xi, b)$, it is clear that $Cor_T(t \cdot c) = \Theta$, but $Cor_T(t) = \xi \neq \Theta$.

□

6.1 The Algorithm LSCA

Let S be a set of trees in $\mathbb{T}_\Delta$ and E a set of contexts in $\mathbb{C}_\Delta$. Then, S is called *subtree-closed* if $t \in S$ implies that all subtrees of t are elements of S.

The set E is called *ξ-prefix closed* with respect to S if $c \in E\backslash\{\xi\}$ implies that there exists c' in E such that $c = f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot c'$ for $m > 0$, $f \in \Delta_m$ and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m \in S$. Let $Composed(S)$ be the set $\{f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \mid m > 0, f \in \Delta_m, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m \in S\}$.

For the tree language T, an *observation table* (S, E, Cor) consists of a nonempty finite subtree-closed set S of trees, a nonempty finite set E of contexts which is ξ-prefix closed with respect to S, and the restriction Cor of the mapping Cor_T to the set $\{t \cdot e \mid t \in S \cup S\Delta, e \in E\}$, denoted by $(S \cup S\Delta)E$. The interpretation of Cor is that, for any $t \in S \cup S\Delta$ and $e \in E$, $Cor(t \cdot e) = c$ if and only if $c \in \mathbb{C}_\Delta$ is the minimal context such that $t \cdot e \cdot c$ is accepted by the target tree recognizer $\mathcal{R}_T$.

As usual, the observation table can be given as a two-dimensional array with rows labeled by elements of $S \cup S\Delta$, columns labeled by elements of E, and the entry for row s and column e equal to $Cor(s \cdot e)$. The elements in E help us distinguish between two different states that have the same minimal context. For example, if there are two trees $s_1, s_2 \in S$ belonging to different equivalence classes for which $Cor_T(s_1) = Cor_T(s_2)$, E should, at some point, contain a context e such that $Cor_T(s_1 \cdot e) \neq Cor_T(s_2 \cdot e)$.

The algorithm LSCA will use the observation table to build a tree recognizer. Rows labeled by the elements of S are candidates for states of the recognizer being constructed, and columns labeled by the elements of E correspond to distinguishing experiments for these states. Rows labeled by elements of $S\Delta$ are used to construct the transition function. An intuitive image is presented in Table 6.1.

Observation table		E $\ldots$ e $\ldots$
S	$\vdots$ s $\vdots$	$\vdots$ $\ldots$ $Cor(s \cdot e)$ $\ldots$ $\vdots$
$S\Delta\backslash S$	$\vdots$ $f(s_1, s_2, \ldots, s_m)$ $\vdots$	$\vdots$ $\ldots$ $Cor(f(s_1, s_2, \ldots, s_m) \cdot e)$ $\ldots$ $\vdots$

Table 6.1: The structure of an observation table (S, E, Cor)

If s is an element of $S \cup S\Delta$, Row_s denotes the finite function from E to $\mathbb{C}_\Delta \cup \{\Theta\}$ defined by $Row_s(e) = Cor(s \cdot e)$ and represents the observed behavior of the tree s. By Row_S we understand $\{Row_s \mid s \in S\}$.

An observation table (S, E, Cor) is called:

- *tree-closed* if for all s in $S\Delta$ there exists t in S such that $Row_s = Row_t$;
- *tree-consistent* if for any s_1 and s_2 in S such that $Row_{s_1} = Row_{s_2}$, we have $Row_{f(t_1,\ldots,t_{i-1},s_1,t_{i+1},\ldots,t_m)} = Row_{f(t_1,\ldots,t_{i-1},s_2,t_{i+1},\ldots,t_m)}$ for all $m > 0$, f in Δ_m, $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m$ in S and $1 \leq i \leq m$;
- *tree-complete* if for any p in $\{Cor(s \cdot e) \mid s \in S, e \in E\}$, $trees(p) \subseteq S$.

In other words a table is tree-closed if the observed behavior of every element of $S\Delta$ can already be seen from the behaviors of elements of S. It is tree-consistent if, for every two trees in S which have the same observed behavior, the corresponding trees in $S\Delta$ (the ones obtained by applying a depth one context with subtrees in S) will also have the same behaviors. A table is tree-complete if all the implicit information which arises from the use of the correcting contexts is used.

If (S, E, Cor) is tree-closed and tree-consistent, one can define the corresponding tree recognizer $\mathcal{R}(S, E, Cor) = (Q, \Delta, \delta, F)$ as follows:

- $Q = \{Row_s \mid s \in S\}$,
- $F = \{Row_s \mid s \in S \text{ and } Cor(s) = \xi\}$,
- $\delta_0(f) = Row_f$ for every $f \in \Delta_0$, and
- $\delta_m(Row_{s_1}, \ldots, Row_{s_m}, f) = Row_{f(s_1,\ldots,s_m)}$ for every $m > 0$, $f \in \Delta_m$ and $s_1, \ldots, s_m \in S$.

It is clear that $\mathcal{R}(S, E, Cor)$ has at most one sink state $q_\Theta = Row_s$, where $Cor(s) = \Theta$. Note that if $Cor(s) = \Theta$, by Remark 11 and the above construction of the recognizer, $Cor(s \cdot e) = \Theta$ for all $e \in E$.

We show that $\mathcal{R}(S, E, Cor)$ is a well-defined (deterministic) tree recognizer. Indeed, if s_1, s_2 are elements of S such that Row_{s_1}=Row_{s_2}, then $Cor(s_1) = Cor(s_1 \cdot \xi)$ and $Cor(s_2) = Cor(s_2 \cdot \xi)$ are defined and equal to each other since E contains ξ. Hence, F is well defined. Since the observation table (S, E, Cor) is tree-consistent, $Row_{f(t_1,\ldots,t_{i-1},s_1,t_{i+1},\ldots,t_m)} = Row_{f(t_1,\ldots,t_{i-1},s_2,t_{i+1},\ldots,t_m)}$ for f in Δ_m, $m > 0$, $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m \in S$, and because it is tree-closed, this common value is equal to Row_s for some s in S. Thus δ is well defined.

6.1.1 The Learner

The algorithm can be seen as an interaction between two actors: the *learner* who must identify the target language being allowed to use specific kinds of questions (SCQs and EQs), and the *teacher* who knows this language and is assumed to answer correctly to the questions.

Algorithm 13 LSCA: Learning regular tree languages with SCQs and EQs

```
1:  Initialize $S$ as $\{a\}$ for an arbitrarily fixed $a \in \Delta_0$, and $E$ as $\{\xi\}$
2:  UPDATE($Tab$)
3:  repeat
4:    repeat
5:      while $Tab$ is not tree-closed do
6:        $Tab$:=CLOSURE($Tab$)
7:      end while
8:      while $Tab$ is not tree-consistent do
9:        $Tab$:=CONSISTENCY($Tab$)
10:     end while
11:     if $Tab$ is not tree-complete then
12:       $Tab$:=COMPLETENESS($Tab$)
13:     end if
14:   until $Tab$ is tree-closed and tree-consistent
15:   eq:=EQUIV($Tab$)
16: until eq=Yes
17: Return $\mathcal{R}(S, E, Cor)$
```

In what follows we explain the steps performed by the learner in order to identify the target language. The learner algorithm uses as its main data structure the observation table that we described in the previous section. We denote an arbitrary observation table (S, E, Cor) by Tab.

The learner starts with an initial observation table $Tab = (S, E, Cor)$, where for an arbitrarily fixed a in Δ_0, $S = \{a\}$, $E = \{\xi\}$, and the value $Cor(a \cdot \xi)$ is obtained by asking a SCQ. The procedure UPDATE receives as a parameter an observation table and asks SCQs for all the entries in the table where there is no information available. The goal of the inner loop is to construct a tree-closed, tree-consistent and tree-complete observation table.

The procedure CLOSURE is very simple. It just searches for a tree $s \in S\Delta$ such that $Row_s \notin Row_S$ and adds it to S. After that it updates the table received as a parameter.

```
Procedure CLOSURE($Tab$)
    find $s$ in $S\Delta$ such that $Row_s \notin Row_S$;
    $S := S \cup \{s\}$;
    UPDATE($Tab$);
    return $Tab$;
```

The procedure CONSISTENCY searches for trees in S which have the same row values, and hence seem to represent the same state of the automaton, but which have a different behavior once we apply a context c in $Composed(S)$. The procedure adds a new experiment to E in order to distinguish between these two states.

```
Procedure CONSISTENCY($Tab$)
    find $s_1, s_2 \in S$ and $e \in E$ such that $Row_{s_1} = Row_{s_2}$ and
     $Cor(f(t_1, \ldots, t_{i-1}, s_1, t_{i+1}, \ldots, t_m) \cdot e) \neq Cor(f(t_1, \ldots, s_2, \ldots, t_m) \cdot e)$
     for some $f \in \Delta_m$ and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m \in S$;
    $E := E \cup \{f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot e\}$;
    UPDATE($Tab$);
    return $Tab$;
```

The procedure COMPLETENESS is recursive, but terminating. It is enough to see that the contexts returned by the teacher have a special feature. Each subtree of those contexts is a minimal tree in its equivalence class. Because T is a regular tree language, there are a finite number of equivalence classes, and hence the table can be found not tree-complete at most n times, where n represents the number of states of the minimal recognizer $\mathcal{R}_T$.

```
Procedure COMPLETENESS($Tab$)
    while there exist $s \in S, e \in E$ such that $trees(Cor(s \cdot e)) \not\subseteq S$ do
       for all $t \in trees(Cor(s \cdot e)) \backslash S$
          $S := S \cup \{t\}$;
       end for
       UPDATE($Tab$);
       COMPLETENESS($Tab$);
    end while
    return $Tab$;
```

The procedure EQUIV just asks the teacher if the conjectured recognizer is equivalent to the target tree recognizer. If the answer is No, it takes the counterexample returned by the teacher and adds it to S, along with all its subtrees. After that it updates the observation table.

```
Procedure EQUIV($Tab$)
    construct $\mathcal{R} = \mathcal{R}(S, E, Cor)$;
    if $\mathcal{R}$ and $\mathcal{R}_T$ are equivalent then return Yes;
    else get a counterexample $t$;
          for all $t' \in sub(t)$
             $S := S \cup \{t'\}$;
          end for
          UPDATE($Tab$);
          return No;
    end if
```

We have seen an algorithm that learns RTLs with SCQs and EQS if a teacher who can answer these two types of queries is available. Answering EQs for RTLs can be done in polynomial time. In the sequel we show that SCQs can also be answered in polynomial time.

6.1.2 The Teacher

Actually, the teacher can give the answer in linear time if some precomputation is done (see Algorithm 14, lines 1-6). We just have to compute for each state q its minimal context c^q (this computation is polynomial in the size of the target recognizer). The algorithm presented below first determines for each q the minimal tree t^q such that $\delta(t^q) = q$, and then computes the minimal context c^q such that $\delta(t \cdot c^q) \in F$ for all $t \in \mathbb{T}_\Delta$ with $\delta(t) = q$, where $\mathcal{R}_T = (Q, \Delta, \delta, F)$ is the minimal tree recognizer for the target tree language T. When the teacher receives a tree t as an input, it is enough to compute $\delta(t)$ (which can be done in linear time) and return $c^{\delta(t)}$.

Algorithm 14 An algorithm for computing minimal contexts

```
 1: for all $q \in Q$ do
 2:     $t^q$:=MinT($q, \emptyset$);
 3: end for
 4: for all $q \in Q$ do
 5:     $c^q$:=MinC($q, \emptyset$);
 6: end for
 7: for any input tree $t$ submitted by the learner do
 8:     $q := \delta(t)$;
 9:     return $c^q$;
10: end for
```

In the process of computing the minimal trees we have to consider only the nonrecursive rules, that is, rules of the form $\delta_m(q_1, \ldots, q_m, f) = q$ such that $q \notin \{q_1, \ldots, q_m\}$. We show that recursive rules do not help finding the minimal tree.

Indeed, let us suppose we know that the minimal trees for the states q_1,..., $q_{i-1}, q_{i+1}, \ldots, q_m$ are $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m$, respectively, and we want to compute the minimal tree for q using the rule $\delta_m(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_m, f) = q$. It is clear that the tree $f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_m)$ is always greater than the tree t, no matter what are the weights of $f, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m, t$ (this is true even for the case in which $m = 1$ and $\omega(f) = 0$).

Also, we have to eliminate the loops from our computation. For example, if we want to compute the minimal tree for a state q using $\delta_m(q_1, \ldots, q_m, f) = q$, then in the recursive process of computing minimal trees for all states q_i we have to eliminate all rules which contain the state q. Otherwise, we will find ourselves in a similar situation to the one described above. That is why we need to introduce the set N of "not allowed" states. We construct the set R_t^N by eliminating from the set of all rules first the recursive ones and after that all the rules of the form $\delta_m(q_1,\ldots,q_m,f)=q$, where at least one q_i is in N.

```
Procedure MinT($q, P$)
    MinTree:=$\Theta$;
    for all rules $\delta_m(q_1,\ldots,q_m,f)=q$ in $R_t^N$ do
       if MinT($q_i, P\cup\{q\}$) $\neq \Theta$ for all $i \in \{1,\ldots,m\}$ then
          $t := f($MinT($q_1, P\cup\{q\}$)$,\ldots,$ MinT($q_m, P\cup\{q\}$)$)$;
          if MinTree = $\Theta$ then MinTree:=$t$
          else
             if $t <_{kbo}$ MinTree then MinTree:=$t$
             end if
          end if
       end if
    end for
    return MinTree;
```

The existence of a minimal tree for each state is guaranteed since the target recognizer is minimal, and hence all its states are reachable. The symbol Θ is used to indicate that the minimal tree has not been found yet or that there is no minimal tree for that state in the given circumstances (we can think of the set N as a supplementary restriction imposed on the search space).

In order to construct the minimal context for each state we also need to remove from the search space the rules which would generate cycles. More precisely, suppose we know that the state q appears in the left-hand side only in the rule $\delta_m(q_1,\ldots,q_{i-1},q,q_{i+1},\ldots,q_m,f)=q'$ and q' appears in the left-hand side of two rules and one of them is $\delta_l(q'_1,\ldots,q'_{i-1},q',q'_{i+1},\ldots,q'_l,g)=q$. It is clear that we should not use this rule to compute the minimal context, first of all because we get into a loop, and secondly because the context c will always be smaller then the context $g(t^{q'_1},\ldots,f(t^{q_1},\ldots,t^{q_{i-1}},\xi,t^{q_{i+1}},\ldots,t^{q_{m-1}}),\ldots,t^{q'_l})\cdot c$ for any $c\in\mathbb{C}_\Delta$. In order to avoid this kind of loops we introduce the set R_c^N which contains only rules of the form $\delta_m(q_1,\ldots,q_m,f)=q$ in which $q\notin N$.

It is also obvious that choosing to replace each state q by the minimal tree t^q is going to give us the smallest correcting context because if we take any other tree t, then $t^q\cdot c <_{kbo} t\cdot c$ for any context $c\in\mathbb{C}_\Delta$. Choosing to stop the search once we found a final state is justified by the properties of the Knuth-Bendix order. Continuing the search on the same path would give us only bigger and bigger contexts.

The existence of a minimal context for all states except for the sink state is also guaranteed since the target recognizer is minimal, and hence there is at most one state which is not co-reachable (the sink state).

```
Procedure MinC($q, N$)
    if $q \in F$ then MinContext:=$\xi$;
    else
       MinContext:=$\Theta$;
       for all rules $\delta_m(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_m, f) = q'$ in $R_c^{N \cup \{q\}}$ do
          if MinC($q', N \cup \{q\}) \neq \Theta$ then
             $c := f(t^{q_1}, \ldots, t^{q_{i-1}}, \xi, t^{q_{i+1}}, \ldots, t^{q_m})$·MinC($q', N \cup \{q\}$);
             if MinContext= $\Theta$ then MinContext:=$c$
             else
                if $c <_{kbo}$ MinContext then MinContext:=$c$;
                end if
             end if
          end if
       end for
    end if
    return MinContext;
```

The symbol Θ is used here for similar reasons: to indicate that either the minimal context has not been found yet, or that given the set of rules which can be applied, there is no minimal context for that state.

6.2 Correctness, Termination, Running Time

In order to prove that the algorithm terminates in finite steps we need further definitions and results.

Assume that (S, E, Cor) is a tree-closed and tree-consistent observation table. We say that the recognizer $\mathcal{R} = (Q, \Delta, \delta, F)$ is *tree-consistent* with the function Cor with respect to the observation table (S, E, Cor) if for every s in $S \cup S\Delta$ and e in E, the following statements hold:

1. $Cor(s \cdot e) = \Theta \Leftrightarrow \delta(s \cdot e)$ is a sink state.
2. $Cor(s \cdot e) = c \in \mathbb{C}_\Delta \Leftrightarrow (\delta(s \cdot e \cdot c) \in F$ and for all $c' \in \mathbb{C}_\Delta$, $\delta(s \cdot e \cdot c') \in F$ implies $c <_{kbo} c')$.

The important fact about the recognizer $\mathcal{R}(S, E, Cor)$ is the following.

Theorem 20. *If (S, E, Cor) is a tree-closed, tree-consistent and tree-complete observation table, then $\mathcal{R}(S, E, Cor)$ is tree-consistent with Cor with respect to (S, E, Cor). Any other tree recognizer tree-consistent with Cor with respect to (S, E, Cor) but not isomorphic with $\mathcal{R}(S, E, Cor)$ must have more states.*

The theorem is proved by a sequence of straightforward lemmas.

Lemma 31. *Assume that (S, E, Cor) is a tree-closed, tree-consistent and tree-complete observation table. For the recognizer $\mathcal{R}(S, E, Cor)$ and for every s in $S \cup S\Delta$, $\delta(s) = Row_s$.*

Proof. It is clear from the definition of $\mathcal{R}(S, E, Cor)$. $\square$

Lemma 32. *Assume that (S, E, Cor) is a tree-closed, tree-consistent and tree-complete observation table. For each s in $S \cup S\Delta$ and e in E, there exists s' in S such that $\delta(s \cdot e) = \delta(s')$ and $Cor(s \cdot e) = Cor(s')$.*

Proof. Let s be an element in $S \cup S\Delta$. We prove our lemma by induction on the depth of e. When e is ξ, by Lemma 31 we have $\delta(s) = Row_s$. Since (S, E, Cor) is a tree-closed table, there exists $s' \in S$ such that $Row_{s'} = Row_s$ which implies $Cor(s') = Cor(s)$ and $\delta(s') = \delta(s)$.

Next, suppose that the result holds for all contexts in E of depth at most k, and let e be an element of E which has the depth $k + 1$. Since E is ξ-prefix closed with respect to S, there exists $e' \in E$ of depth k such that $e = f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot e'$ for some $m > 0$, $f \in \Delta_m$ and $t_1, \ldots, t_{i-1}$, t_{i+1}, $\ldots$, t_m in S. Because (S, E, Cor) is tree-closed, we can take s' in S such that Row_s=$Row_{s'}$ (hence, $Cor(s \cdot e) = Cor(s' \cdot e)$ and $\delta(s) = \delta(s')$).

Then, $\delta(s \cdot e) = \delta(s \cdot f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot e')$
$= \delta(s' \cdot f(t_1, \ldots, t_{i-1}, \xi, t_{i+1}, \ldots, t_m) \cdot e')$ (replacement lemma)
$= \delta(f(t_1, \ldots, t_{i-1}, s', t_{i+1}, \ldots, t_m) \cdot e')$.

By the induction hypothesis on e', there exists s'' in S such that $\delta(s'') = \delta(f(t_1, \ldots, t_{i-1}, s', t_{i+1}, \ldots, t_m) \cdot e')$ and $Cor(f(t_1, ..., t_{i-1}, s', t_{i+1}, ..., t_m) \cdot e') = Cor(s'')$ which implies $\delta(s \cdot e) = \delta(s'')$ and $Cor(s \cdot e) = Cor(s'')$. $\square$

Lemma 33. *Assume that (S, E, Cor) is a tree-closed, tree-consistent and tree-complete observation table. Then the recognizer $\mathcal{R}(S, E, Cor)$ is tree-consistent with the function Cor with respect to (S, E, Cor).*

Proof. Let s be in $S \cup S\Delta$ and e in E. From Lemma 32 we know that there exists s' in S such that $\delta(s \cdot e) = \delta(s')$ and $Cor(s \cdot e) = Cor(s')$. So, it is enough to prove that for all s' in S, we have:

1. $Cor(s') = \Theta \Leftrightarrow \delta(s')$ is a sink state.
2. $Cor(s') = c \in \mathbb{C}_\Delta \Leftrightarrow (\delta(s' \cdot c) \in F$ and c is the smallest context with this property).

For the first statement, it is enough to see that $Cor(s') = \Theta$ if and only if $Row_{s'}$ is a sink state, and by Lemma 31, if and only if $\delta(s')$ is a sink state. For the second one, we will first show that $Cor(s') = c \in \mathbb{C}_\Delta$ implies $\delta(s' \cdot c) \in F$. If $c = \xi$, the result is immediate. Suppose c equals $c_1 \cdot c_2 \cdot \ldots \cdot c_n$ with $c_i \in \mathbb{C}_\Delta$ having the node labeled ξ at depth exactly 1 for all $i \in \{1, 2, \ldots, n\}$ and $n \geq 1$.

From the definition of $trees(c)$ it is clear that $trees(c_i) \subseteq trees(c)$ for every i in $\{1, 2, \ldots, n\}$. But (S, E, Cor) is tree-complete, so $trees(c) \subseteq S$, and hence

$trees(c_i) \subseteq S$. Using the fact that $depth(c_i) = 1$, we obtain that c_i belongs to $Composed(S)$. Then, because the table (S, E, Cor) is tree-closed, we can inductively find the trees $s_1, \ldots, s_n$ in S, as it is shown in the sequel:

$s' \cdot c_1 \in S\Delta \Rightarrow \exists\ s_1 \in S$: $\delta(s' \cdot c_1) = \delta(s_1)$, $Cor(s' \cdot c_1) = Cor(s_1)$

$s_1 \cdot c_2 \in S\Delta \Rightarrow \exists\ s_2 \in S$: $\delta(s_1 \cdot c_2) = \delta(s_2)$, $Cor(s_1 \cdot c_2) = Cor(s_2)$

...

$s_{n-1} \cdot c_n \in S\Delta \Rightarrow \exists\ s_n \in S$: $\delta(s_{n-1} \cdot c_n) = \delta(s_n)$, $Cor(s_{n-1} \cdot c_n) = Cor(s_n)$.

Clearly, $\delta(s' \cdot c) = \delta(s_n)$ (using the replacement lemma). Starting from $Cor(s') = c$ and applying several times Remark 9, we obtain $Cor(s_n) = \xi$ which is equivalent to $\delta(s_n) \in F$, and furthermore with $\delta(s' \cdot c) \in F$.

Next, we show that if we take $c = c_1 \cdot c_2 \cdot \ldots \cdot c_n$ to be the smallest context in Knuth-Bendix order such that $\delta(s' \cdot c) \in F$, then $Cor(s') = c$. Note that the set $\{c \mid \delta(s' \cdot c) \in F\}$ is not empty because we proved that it contains the context $Cor(s')$.

We know that $\delta(s' \cdot c) = \delta(s_n)$, and $\delta(s_n) \in F$ implies $Cor(s_n) = \xi$. But $Cor(s_n) = Cor(s_{n-1} \cdot c_n)$, so $Cor(s_{n-1} \cdot c_n) = \xi$, and hence $c_n \in Fron_T(s_{n-1})$. We prove that this implies $Cor(s_{n-1}) = c_n$. Indeed, assume that there exists a context $c'_n <_{kbo} c_n$ such that $Cor(s_{n-1}) = c'_n$. Then, we exhibit a context c' strictly smaller than c such that $\delta(s' \cdot c') \in F$ which contradicts the choice we have made for c. We can take c' to be $c_1 \cdot c_2 \cdot \ldots \cdot c_{n-1} \cdot c'_n$. Therefore, $\delta(s' \cdot c')$ $= \delta(s' \cdot c_1 \cdot \ldots \cdot c_{n-1} \cdot c'_n) = \delta(s_{n-1} \cdot c'_n)$, and because $Cor(s_{n-1}) = c'_n$, it follows that $\delta(s_{n-1} \cdot c'_n) \in F$.

But $Cor(s_{n-2} \cdot c_{n-1}) = Cor(s_{n-1}) = c_n$. Reasoning in the same manner we obtain $Cor(s') = c_1 \cdot c_2 \cdot \ldots \cdot c_n$ which implies $Cor(s') = c$.

We showed that:

1. $Cor(s') = c \in \mathbb{C}_\Delta$ implies $\delta(s' \cdot c) \in F$.

2. If c is the smallest context such that $\delta(s' \cdot c) \in F$, then $Cor(s') = c$.

So, if $Cor(s') = c \in \mathbb{C}_\Delta$, then $\delta(s' \cdot c) \in F$. Now, assume c is not the smallest context such that $\delta(s' \cdot c) \in F$. Let us take $c' <_{kbo} c$ to be the smallest context with this property. Then, $Cor(s') = c'$, and hence $c = c'$. This concludes the proof of the lemma. □

Lemma 34. *Assume that (S, E, Cor) is a tree-closed, tree-consistent and tree-complete observation table and that $\mathcal{R}(S, E, Cor) = (Q, \Delta, \delta, F)$ has n states. If $\mathcal{R}' = (Q', \Delta, \delta', F')$ is any recognizer tree-consistent with Cor with respect to (S, E, Cor) that has n or fewer states, then $\mathcal{R}'$ is isomorphic with $\mathcal{R}(S, E, Cor)$.*

Proof. We define the relation $\phi \subseteq Q \times Q'$ as follows: for each $s \in S$, $Row_s\ \phi\ q' \Leftrightarrow \delta'(s) = q'$.

Let us take $s_1, s_2 \in S$ such that there exists $q' \in Q'$, $Row_{s_1} \ \phi \ q'$ and $Row_{s_2} \ \phi \ q'$ (clearly, $\delta'(s_1) = q' = \delta'(s_2)$). We will show that this implies $Row_{s_1} = Row_{s_2}$. Suppose by contrary that $Row_{s_1} \neq Row_{s_2}$. Then, there exists $e \in E$ such that $Row_{s_1}(e) \neq Row_{s_2}(e)$, and so $Cor(s_1 \cdot e) \neq Cor(s_2 \cdot e)$. We distinguish two cases: $Cor(s_1 \cdot e) = \Theta$, $Cor(s_2 \cdot e) \neq \Theta$ (the case $Cor(s_1 \cdot e) \neq \Theta$, $Cor(s_2 \cdot e) = \Theta$ is symmetric), and $Cor(s_1 \cdot e), Cor(s_2 \cdot e) \neq \Theta$.

I) $Cor(s_1 \cdot e) = \Theta$, $Cor(s_2 \cdot e) = c \neq \Theta$. Because $\mathcal{R}'$ is tree-consistent with Cor, $\delta'(s_1 \cdot e)$ is a sink state and $\delta'(s_2 \cdot e \cdot c) \in F'$. But $\delta'(s_1) = \delta'(s_2)$, and so $\delta'(s_1 \cdot e) = \delta'(s_2 \cdot e)$. Hence, $\delta'(s_2 \cdot e)$ is a sink state. This means that $\delta'(s_2 \cdot e \cdot c)$ is a sink state which contradicts $\delta'(s_2 \cdot e \cdot c) \in F'$.

II) $Cor(s_1 \cdot e) = c_1$, $Cor(s_2 \cdot e) = c_2$, $c_1 \neq c_2$ and $c_1, c_2 \neq \Theta$. Because $\mathcal{R}'$ is tree-consistent with Cor, we have $\delta'(s_1 \cdot e \cdot c_1) \in F'$, $\delta'(s_2 \cdot e \cdot c_2) \in F'$ and c_1, c_2 are the smallest contexts with this property. But $\delta'(s_1) = \delta'(s_2)$, and so $\delta'(s_1 \cdot e \cdot c_1) = \delta'(s_2 \cdot e \cdot c_1)$. Hence, $\delta'(s_2 \cdot e \cdot c_1) \in F'$ which implies $c_2 \leq_{kbo} c_1$. In a similar way it can be shown that $c_1 \leq_{kbo} c_2$. We draw the conclusion that $c_1 = c_2$ which leads to a contradiction.

We have shown that the relation ϕ is an injection. This implies that $|Q| \leq |\phi(Q)|$. 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 isomorphism from $\mathcal{R}(S, E, Cor)$ to $\mathcal{R}'$, that is, it preserves the transition function and $\phi(F) = F'$:

1. We have $q' \in \phi(F) \Leftrightarrow \exists s \in S$ such that $Row_s \in F$ and $\phi(Row_s) = q' \Leftrightarrow \exists s \in S$ such that $Cor(s) = \xi$ and $\delta'(s) = q'$. Because $\mathcal{R}'$ is tree-consistent with Cor with respect to (S, E, Cor), this is equivalent to $\exists s \in S$ such that $\delta'(s) \in F'$ and $\delta'(s) = q'$, and hence to $q' \in F'$.

2. Let $f \in \Delta_0 \cap S$. We have $\phi(\delta_0(f))$=$\phi(Row_f)$=$\delta'_0(f)$.

3. Let us take $m > 0$, $f \in \Delta_m$ and $t_1, \ldots, t_m \in S$. We want to show that $\phi(\delta_m(Row_{t_1}, \ldots, Row_{t_m}, f)) = \delta'_m(\phi(Row_{t_1}), \ldots, \phi(Row_{t_m}), f)$. First of all, $\phi(\delta_m(Row_{t_1}, \ldots, Row_{t_m}, f)) = \phi(Row_{f(t_1,\ldots,t_m)}) = \phi(Row_{s'}) = \delta'(s')$ for some s' in S such that $Row_{s'} = Row_{f(t_1,\ldots,t_m)}$. On the other hand, $\delta'_m(\phi(Row_{t_1}), \ldots, \phi(Row_{t_m}), f) = \delta'_m(\delta'(t_1), \ldots, \delta'(t_m), f)$ which furthermore equals $\delta'(f(t_1, ..., t_m))$. Since $\delta'(s')$ and $\delta'(f(t_1, \ldots, t_m))$ have identical row values, namely $Row_{s'}$ and $Row_{f(t_1,\ldots,t_m)}$, they must be the same

state of $\mathcal{R}'$, and hence we obtain that $\phi(\delta_m(Row_{t_1}, ..., Row_{t_m}, f))$ is equal to $\delta'_m(\phi(Row_{t_1}), \ldots, \phi(Row_{t_m}), f)$.

This concludes the proof. □

Now the proof of Theorem 20 follows since from Lemma 33 we know that $\mathcal{R}(S, E, Cor)$ is tree-consistent with Cor and Lemma 34 shows that any other recognizer tree-consistent with Cor is either isomorphic to $\mathcal{R}(S, E, Cor)$ or contains at least one more state. Thus, $\mathcal{R}(S, E, Cor)$ is the unique minimal tree recognizer tree-consistent with Cor.

If the algorithm terminates, it obviously returns the correct recognizer. It is also clear that LSCA halts in finitely many steps since:

- whenever the table is found not tree-closed or not tree-consistent the number of distinct rows in S increases by at least one;
- the procedure COMPLETENESS is performed at most n times;
- for any tree-closed, tree-consistent and tree-complete observation table (S, E, Cor), if n denotes the number of different values of Row_s for s in S, then any recognizer tree-consistent with Cor must have at least n states (from the injectivity of function ϕ defined in Lemma 34);
- LSCA can make at most $n - 1$ incorrect conjectures since the size of the conjectured recognizer is initially at least one and may not exceed n (whenever the teacher answers by a counterexample the number of distinct values of Row_s for s in S increases by at least 1).

Hence, LSCA always eventually finds a tree-closed, tree-consistent and tree-complete observation table (S, E, Cor) and makes a conjecture $\mathcal{R}(S, E, Cor)$. Since LSCA has to make another conjecture as long as it is running, it must terminate by making a correct conjecture.

Let us now discuss the time complexity of the algorithm LSCA. The total running time of LSCA could be bounded by a polynomial in n (n is the number of states in $\mathcal{R}_T$) if the teacher always returned the minimal possible counterexample. However, there is no restriction on the size of the counterexample. Hence, we have to use both n and the maximum size of the counterexamples (denoted by m in the sequel) as parameters when describing the complexity of the algorithm.

We will first show that all the procedures run in time polynomial in the size of the observation table and that the final observation table is polynomial in n and m (although it is exponential in the maximum rank of symbols).

One may notice that CLOSURE(Tab) and CONSISTENCY(Tab) cannot be called more than $n - 1$ times since each of them increases the number of distinct rows from S, and this number is initially one and cannot exceed n. The same statement also holds for the procedure COMPLETENESS(*Tab*): it cannot be called more than n times since the total number of minimal tree representatives for the states coincides with the number of states. When the observation table is tree-closed, tree-consistent and tree-complete, the algorithm runs the procedure EQUIV(*Tab*), and this can happen no more than n times since any counterexample adds at least one distinct row to the set S.

To compute CLOSURE(*Tab*), the algorithm considers each pair in $S \times (S\Delta \backslash S)$ and compares the observed behaviors of the two trees (in the worst case). This task can be done using $\mathcal{O}(|S| \cdot |S\Delta\backslash S| \cdot |E|)$ comparisons.

To compute CONSISTENCY(*Tab*), the algorithm finds the trees s, s' in S such that $Row_s = Row_{s'}$. This requires, in the worst case, $\mathcal{O}(|S|^2 \cdot |E|)$ operations. Then, it tries to find $m > 0$, $f \in \Delta_m$ and $t_1, \ldots, t_m \in S$ such that $Row_{f(t_1,\ldots,t_{i-1},s,t_{i+1},\ldots,t_m)} \neq Row_{f(t_1,\ldots,t_{i-1},s',t_{i+1},\ldots,t_m)}$. In the worst case the algorithm would have to perform $\mathcal{O}(|\Delta| \cdot |S|^{h-1} \cdot |E|)$ comparisons, where h is the maximum arity of the symbols in Δ.

To compute COMPLETENESS(*Tab*), for all contexts in $\{Cor(s \cdot e) \mid s \in S, e \in E\}$, the algorithm checks if their direct subtrees are also in S. This can be done in $\mathcal{O}(n(h-1) \cdot |S|^2 \cdot |E|)$ steps.

To compute EQUIV(*Tab*), LSCA first constructs $\mathcal{R}(S, E, Cor)$ in time polynomial in the size of the observation table. A counterexample requires the addition of at most m trees of size at most m to S, and this can happen at most $n-1$ times.

For the procedure UPDATE(*Tab*) it is clear that the number of SCQs asked coincides with the size of the final observation table. Regarding the dimensions of S, $S\Delta\backslash S$ and E, it is easy to see that the number of trees in S cannot exceed $2n + m(n-1)$ (it starts with one tree, the procedures CLOSURE and CONSISTENCY together can add at most $n-1$ trees, EQUIV at most $m(n-1)$ trees, and COMPLETENESS at most n trees), $S\Delta\backslash S$ has at most $|\Delta||S|^h$ elements, and the number of contexts in E cannot be greater than n.

Hence, the total running time of LSCA can be bounded by a polynomial function of m and n.

6.3 Running Example

In what follows we show how our algorithm works on an example. Let $\Delta = \{f/3, g/2, a/0, b/0\}$ be a ranked alphabet and assume the following order among the symbols of the alphabet: $a < b < g < f$. The weight function $\omega : \Delta \cup \{\xi\} \to \mathbb{R}_0^+$ is defined by: $\omega(\xi) = \omega(a) = \omega(b) = 1$, $\omega(g) = 2$ and $\omega(f) = 3$.

Assume that the target language is

$$T = \{g(a,b) \cdot [f(a,\xi,b)]^n \mid n \geq 0\} \cup \{g(b,b) \cdot [f(a,\xi,b)]^n \mid n \geq 0\},$$

where $[f(a,\xi,b)]^0 = \xi$ and $[f(a,\xi,b)]^n = [f(a,\xi,b)]^{n-1} \cdot f(a,\xi,b)$, $n \geq 1$. Clearly, T is a regular tree language and $\mathcal{R}_T = (Q, \Delta, \delta, F)$ the minimal tree recognizer for T, where $Q = \{q_a, q_b, q_s, q_f\}$, $F = \{q_f\}$, and the transition function δ is defined as follows:

- $\delta_0(a) = q_a$, $\delta_0(b) = q_b$,
- $\delta_2(q_a, q_b, g) = \delta_2(q_b, q_b, g) = q_f$, $\delta_2(q_a, q_a, g) = \delta_2(q_b, q_a, g) = q_s$,
- $\delta_3(q_a, q_f, q_b, f) = q_f$, and $\delta_3(q_x, q_y, q_z, f) = q_s$ for all remaining cases.

The algorithm starts by constructing the following observation table, where $S = \{a\}$ and $E = \{\xi\}$ (actually S can contain any of the two symbols with arity 0, i.e., a or b).

Tab_1		E
		ξ
S	a	$g(\xi,b)$
$S\Delta\backslash S$	b	$g(\xi,b)$
	$g(a,a)$	Θ
	$f(a,a,a)$	Θ

Table 6.2: The observation table for $S = \{a\}$ and $E = \{\xi\}$

The observation table from Table 6.2 is not tree-closed because $Row_{g(a,a)}$ is not in Row_S, and the algorithm proceeds by adding the tree $g(a,a)$ to S. Updating the current table, we get the observation table from Table 6.3.

Tab_2		E
		ξ
S	a	$g(\xi,b)$
	$g(a,a)$	Θ
$S\Delta\backslash S$	b	$g(\xi,b)$
	$g(a,g(a,a))$	Θ
	$\vdots$	$\vdots$
	$f(g(a,a),g(a,a),g(a,a))$	Θ

Table 6.3: The observation table for $S = \{a, g(a,a)\}$ and $E = \{\xi\}$

This table is tree-closed and tree-consistent but not tree-complete (we have $g(\xi,b) \in Cor(S \cdot E)$, and $trees(g(\xi,b)) = \{b\} \not\subseteq S$). Hence, LSCA adds the tree b to S and updates the table (see Table 6.4).

Tab_3		E: ξ
S	a	$g(\xi,b)$
	b	$g(\xi,b)$
	$g(a,a)$	Θ
$S\Delta\backslash S$	$g(a,b)$	ξ
	$g(b,a)$	Θ
	$g(b,b)$	ξ
	$\vdots$	$\vdots$
	$f(g(a,a),g(a,a),g(a,a))$	Θ

Table 6.4: The observation table for $S = \{a, b, g(a,a)\}$ and $E = \{\xi\}$

This table is not tree-closed because $Row_{g(a,b)} \notin Row_S$ nor tree-consistent because $Row_a = Row_b$ and $Row_{g(a,a)} = \Theta \neq \xi = Row_{g(a,b)}$. The algorithm continues by adding the tree $g(a,b)$ to S and the context $g(a,\xi)$ to E. We get the following observation table (Table 6.5).

Tab_4		E: ξ	E: $g(a,\xi)$	States
S	a	$g(\xi,b)$	Θ	q_a
	b	$g(\xi,b)$	ξ	q_b
	$g(a,a)$	Θ	Θ	q_s
	$g(a,b)$	ξ	Θ	q_f
$S\Delta\backslash S$	$g(b,a)$	Θ	Θ	q_s
	$g(b,b)$	ξ	Θ	q_f
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$f(a,g(a,a),b)$	Θ	Θ	q_s
	$f(a,g(a,b),b)$	ξ	Θ	q_f
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$f(g(a,b),g(a,b),g(a,b))$	Θ	Θ	q_s

Table 6.5: The observation table for $S = \{a, b, g(a,a), g(a,b)\}$, $E = \{\xi, g(a,\xi)\}$

The above observation table is tree-closed, tree-consistent and tree-complete, and the conjectured recognizer $\mathcal{R}(S,E,Cor)$ is isomorphic to $\mathcal{R}_T$. Hence, the teacher's answer to the EQ will be `Yes`. The algorithm outputs $\mathcal{R}(S,E,Cor)$ and halts.

6.4 Regular Tree Languages Learnable with Structural Correction Queries

Recall that for the string case, the more "injective" a language is, the better our algorithm - that uses CQs instead of MQs - works, to the extent that for pure injective languages one does not need EQs anymore. In this section we show that these results can be extended to trees, and hence that there exists an infinite class of tree languages learnable with structural correction queries alone. Moreover, we present a restricted version of LSCA for injective tree languages, namely LSCA$^{\text{inj}}$.

A tree language T is *injective* (or T has the *injectivity property*) if for any two trees t_1, t_2 in $\mathbb{T}_\Delta$, $t_1 \cong_T t_2 \Leftrightarrow Cor_T(t_1) = Cor_T(t_2)$. The subclass of regular tree languages which have the injectivity property is denoted by $Tree\mathcal{I}nj$.

Note that one implication always holds: $t_1 \cong_T t_2 \Rightarrow Cor_T(t_1) = Cor_T(t_2)$. It is an easy exercise to see that the tree language from Example 6 belongs to the class $Tree\mathcal{I}nj$ and the one from Section 6.3 does not.

Theorem 21. *For any tree language T in $Tree\mathcal{I}nj$, the algorithm LSCA asks only one EQ when learning T.*

Proof. For the two trivial languages $T = \emptyset$ and $T = \mathbb{T}_\Delta$ the proof is immediate. Hence, let T be a non-trivial tree language in $Tree\mathcal{I}nj$ and $\mathcal{R}_T = (Q', \Delta, \delta', F')$ the minimal tree recognizer accepting T with $|Q'| = n > 1$.

We show by induction that, at any step $k < n$ of the algorithm, the table (S, E, Cor) has the following properties: $E = \{\xi\}$, $|S| = k$, and (S, E, Cor) is tree-consistent but not tree-closed.

For $k = 1$, $E = \{\xi\}$ and $S = \{a\}$, where $a \in \Delta_0$. It is clear that the table is not tree-closed since this would imply that the target recognizer has only one state which contradicts the non-triviality of T.

Suppose that the result holds for all steps strictly smaller than k, and we want to prove that the above three conditions are satisfied at step k ($k < n$). Since at step $k-1$ the set S had $k-1$ trees and the table was not tree-closed, it means that at step k after running procedure CLOSURE the set S has one more element (and hence k elements), E continues to be $\{\xi\}$, and clearly the table is tree-consistent. The only fact that needs to be shown is that (S, E, Cor) is not tree-closed.

Assume by contradiction that (S, E, Cor) is tree-closed. Then, we can construct a recognizer $\mathcal{R}(S, E, Cor) = (Q, \Delta, \delta, F)$ with $|Q| = k$, as in Section 6.1 (note that the observation table does not need to be tree-complete).

We define $\phi : Q \to Q'$ by $\phi(Row_s) = \delta'(s)$. We prove that the following statements hold: ϕ is well defined and injective, $\phi(\delta_m(Row_{s_1}, \ldots, Row_{s_m}, f))$ $= \delta'_m(\phi(Row_{s_1}), \ldots, \phi(Row_{s_m}), f)$ for $m \geq 0$, $f \in \Delta_m$, $s_1, \ldots, s_m \in S$, and $\phi(F) \subseteq F'$.

Clearly, ϕ is well defined since there are no two trees $s_1 \neq s_2$ in S such that $Row_{s_1} = Row_{s_2}$. To see that ϕ is injective, let us take two distinct states in Q, namely Row_{s_1} and Row_{s_2}. Because $E = \{\xi\}$, $Row_{s_1} \neq Row_{s_2}$ is equivalent to

$Cor(s_1) \neq Cor(s_2)$, and since T has the injectivity property, this is equivalent to $s_1 \ncong_T s_2 \Leftrightarrow \delta'(s_1) \neq \delta'(s_2) \Leftrightarrow \phi(Row_{s_1}) \neq \phi(Row_{s_2})$. Thus, ϕ is injective.

We have $\delta_m(Row_{s_1}, \ldots, Row_{s_m}, f) = Row_{f(s_1,\ldots,s_m)} = Row_s$ for some s in S. Hence $\phi(\delta_m(Row_{s_1}, \ldots, Row_{s_m}, f)) = \phi(Row_s) = \delta'(s)$. On the other hand, we know that $Cor(s) = Cor(f(s_1, \ldots, s_m))$ and the tree language T is injective. We get that $s \cong_T f(s_1, \ldots, s_m)$, so $\delta'(s) = \delta'(f(s_1, \ldots, s_m))$. Hence, $\phi(\delta_m(Row_{s_1}, \ldots, Row_{s_m}, f)) = \delta'(f(s_1, \ldots, s_m)) = \delta'_m(\delta'(s_1), \ldots, \delta'(s_m), f) = \delta'_m(\phi(Row_{s_1}), \ldots, \phi(Row_{s_m}), f)$.

For any q' in $\phi(F)$, there exists s in S such that $Row_s \in F$ and $\phi(Row_s) = q'$ which is equivalent to $Cor(s) = \xi$ and $\delta'(s) = q'$, and furthermore to $q' \in F'$. Hence, $\phi(F) \subseteq F'$.

Clearly, $\mathcal{R}(S, E, Cor)$ is a complete recognizer (i.e., for all $m \geq 0$, δ_m is a total function). Now it is enough to see that we have constructed an injective morphism from $\mathcal{R}(S, E, Cor)$ to $\mathcal{R}_T$ such that $|Q| = k < n = |Q'|$ which leads us to a contradiction. Hence, (S, E, Cor) is not tree-closed.

We proved that the procedure CLOSURE(Tab) is called at least $n-1$ times. Obviously, this means that by that time S already has n different rows, and since it cannot have more (the total number of states in the target recognizer is n), in step n the table will be tree-closed and tree-consistent. After running the procedure COMPLETNESS (which cannot add any new row because the table already contains n different rows), the output recognizer will be the target one, and the answer to the EQ will be `Yes`. □

Corollary 12. *The class $Tree\mathcal{I}nj$ is learnable with SCQs only.*

Proof. One can modify LSCA to output the automaton $\mathcal{R}(S, E, Cor)$ and halt when the table is tree-closed, tree-consistent and tree-complete. From the previous theorem it is clear that the algorithm will return the target automaton. □

The restricted version of LSCA for the subclass of injective languages is presented in Algorithm 15.

Algorithm 15 The algorithm LSCA$^{\text{inj}}$ for tree injective languages

```
1: Initialize $S$ as $\{a\}$ for an arbitrarily fixed $a \in \Delta_0$, and $E$ as $\{\xi\}$;
2: UPDATE($Tab$);
3: while $Tab$ is not tree-closed do
4:     $Tab$:=CLOSURE($Tab$);
5: end while
6: Return $\mathcal{R}(S, E, Cor)$;
```

6.5 Remarks and Further Research

We have seen that the results obtained for learning DFAs with PCQs and EQs could be smoothly extended to cover their tree counterpart. In this chapter we learned that RTLs are polynomial time learnable with SCQs and EQs, and that EQs are not needed when the class to be identified is a subset of $Tree\mathcal{I}nj$ (injective regular tree languages). In order to do that, correcting strings became correcting contexts, and LCA has been adapted to work with trees (a few modifications were of course required). Obviously, the answer to CQs corresponding to trees are no longer suffixes, but contexts that "embrace" the trees received as input. Therefore, answering CQs for trees is a bit more intricate, but polynomial nevertheless.

One may notice as well that because of the enhanced information carried over by contexts (along with a correcting context we also get some details about the structure of derivations), it is less common to have tree languages with high injectivity degrees.

Regarding the complexity of our algorithm, we mentioned at some point that although LSCA is polynomial in m and n (m being the size of the biggest counterexample returned by the teacher and n the number of states of the target recognizer), it is exponential in the rank of the symbols. That is because in order to define a total transition function for a deterministic bottom-up tree recognizers $\mathcal{R} = (Q, \Delta, \delta, F)$, one needs $|Q|^m$ entries for each symbol f of arity m. This drawback is actually inherited from the previous adaptations of L^* to trees. There exists though a generalization of L^* for tree languages in which dead states and unnecessary long counterexamples are avoided (saving a lot of time and space resources) by Drewes and Högberg [DH07]. A similar approach could be implemented for LSCA as well, but since it is rather straightforward, we leave it to the reader as an exercise.

Finally, we should not forget the role of the weight function in determining the correcting context. By changing the weights or the order between symbols of the alphabet one can affect the answer to SCQs. On the other hand, the order we have used in our algorithm, i.e., a Knuth-Bendix order, was only for illustrative purposes: all the results are preserved if trees are compared using any simplification order.

Further research topics on learning RTLs include, but is not limited to, the study of their learnability with other types of SCQs and EQs, or the extension of the results on l-injective languages obtained in the string case to trees.

Bibliography

[AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? (extended abstract). In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 444–454, New York, NY, USA, 1991. ACM Press.

[AK94] Dana Angluin and Mārtiņš Kriķis. Learning with malicious membership queries and exceptions (extended abstract). In *Proc. 7th Annual Conference on Computational Learning Theory (COLT '94)*, pages 57–66, New York, NY, USA, 1994. ACM Press.

[Ang78] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

[Ang79] Dana Angluin. Finding patterns common to a set of strings (extended abstract). In *Proc. 11th Annual ACM Symposium on Theory of Computing (STOC '79)*, pages 130–141, New York, NY, USA, 1979. ACM Press.

[Ang80a] Dana Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21(1):46–62, 1980.

[Ang80b] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[Ang81] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51(1):76–87, 1981.

[Ang82] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

[Ang87a] Dana Angluin. Learning k-bounded context-free grammars. Techn. Rep. TR-557, Yale University, New Haven, Conn., 1987.

[Ang87b] Dana Angluin. Learning k-term DNF formulas using queries and counter-examples. Techn. Rep. TR-559, Yale University, New Haven, Conn., 1987.

[Ang87c] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Ang89] Dana Angluin. Equivalence queries and approximate fingerprints. In X, editor, *Proc.* 2nd *Annual Workshop on Computational Learning Theory (COLT '89)*, pages 134–145, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990.

[Ang01] Dana Angluin. Queries revisited. In Naoki Abe, Roni Khardon, and Thomas Zeugmann, editors, *Proc.* 12th *International Conference on Algorithmic Learning Theory (ALT'01)*, volume 2225 of *Lecture Notes in Artificial Intelligence*, pages 12–31, Berlin, Heidelberg, 2001. Springer.

[Ang04] Dana Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Computing Surveys*, 15(3):237–269, 1983.

[BB06] Leonor Becerra-Bonache. *On the Learnability of Mildly Context-Sensitive Languages using Positive Data and Correction Queries.* PhD thesis, University of Tarragona, 2006.

[BeBiDe05] Leonor Becerra-Bonache, Cristina Bibire, and Adrian Horia Dediu. Learning DFA from corrections. In Henning Fernau, editor, *TAGI*, WSI-2005-14, pages 1–11. Technical Report, University of Tubingen, 2005.

[BBdlHJT07] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings with correction queries. In Joost N. Kok, Jacek Koronacki, Ramon López de Mántaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *Proc.* 18th *European Conference on Machine Learning (ECML '07)*, volume 4701 of *Lecture Notes in Computer Science*, pages 18–29, Berlin, Heidelberg, 2007. Springer-Verlag.

[BeDeTî06] Leonor Becerra-Bonache, Adrian Horia Dediu, and Cristina Tîrnăucă. Learning DFA from correction and equivalence queries. In Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, and E. Tomita, editors, *Proc.* 8th *International Colloquium on Grammatical Inference (ICGI '06)*, volume 4201 of *Lecture Notes in Artificial Intelligence*, pages 281–292, Berlin, Heidelberg, 2006. Springer-Verlag.

[BBY04] Leonor Becerra-Bonache and Takashi Yokomori. Learning mild context-sensitiveness: Toward understanding children's language learning. In Georgios Paliouras and Yasubumi Sakakibara, editors, *ICGI*, volume 3264 of *Lecture Notes in Computer Science*, pages 53–64, Berlin, Heidelberg, 2004. Springer-Verlag.

[BDGW94] José Luis Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. An optimal parallel algorithm for learning DFA. In *Proc. 7th Annual Conference on Computational Learning Theory (COLT '94)*, pages 208–217, New York, NY, USA, 1994. ACM Press.

[BDGW97] José Luis Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. Algorithms for learning finite automata from queries: A unified view. In Ding-Zhu Du and Ker-I Ko, editors, *Advances in Algorithms, Languages, and Complexity - In Honor of Ronald V. Book*, pages 53–72. Kluwer Academic Publishers, 1997.

[BH70] Roger Brown and Camille Hanlon. Derivational complexity and the order of acquisition in child speech. In J.R. Hayes, editor, *Cognition and the Development of Language*, pages 155–175. Wiley, New York,, 1970.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

[BR87] Piotr Berman and Robert Roos. Learning one-counter languages in polynomial time (extended abstract). In *Proc. of the 28th Annual Symposium on Foundations of Computer Science (FOCS '87)*, pages 61–67. IEEE, 1987.

[BS89] Avrim Blum and Mona Singh. Learning functions of k terms. In M. Fulk and John Case, editors, *Proc. 3rd Annual Workshop on Computational Learning Theory (COLT '90)*, pages 144–153, San Mateo, 1989. Morgan Kaufmann Publishers Inc.

[CCM08] Alexander Clark, Francois Coste, and Laurent Miclet, editors. *Proc. 9th International Colloquium on Grammatical Inference (ICGI '08)*, volume 5278 of *Lecture Notes in Artificial Intelligence*, Berlin, Heidelberg, 2008. Springer-Verlag.

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications (TATA). Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. Release October, 12th 2007.

[Cha] Sam Chapman. Sam's string metrics. `http://www.dcs.shef.ac.uk/~sam/stringmetrics.html`.

[Cho56] Noam Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124, 1956.

[CJR+03] John Case, Sanjay Jain, Rüdiger Reischuk, Frank Stephan, and Thomas Zeugmann. Learning a subclass of regular patterns in polynomial time. In Gavaldà et al. [GJT03], pages 234–246.

[CJR+06] John Case, Sanjay Jain, Rüdiger Reischuk, Frank Stephan, and Thomas Zeugmann. Learning a subclass of regular patterns in polynomial time. *Theoretical Computer Science*, 364(1):115–131, 2006.

[CO94] Rafael Carrasco and José Oncina, editors. *Proc.* 2nd *International Colloquium on Grammatical Inference (ICGI '94)*, volume 862 of *Lecture Notes in Artificial Intelligence*, Berlin, Heidelberg, 1994. Springer-Verlag.

[COCR98] Rafael Carrasco, José Oncina, and Jorge Calera-Rubio. Stochastic inference of regular tree languages. In Honavar and Slutski [HS98], pages 187–198.

[CS83] John Case and Carl Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 2(25):193–220, 1983.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[DH03] Frank Drewes and Johanna Högberg. Learning a regular tree language from a teacher. In Zoltán Ésik and Zoltán Fülöp, editors, *Proc.* 7th *International Conference on Developments in Language Theory*, volume 2710 of *Lecture Notes in Computer Science*, pages 279–291, Berlin, Heidelberg, 2003. Springer-Verlag.

[DH07] Frank Drewes and Johanna Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems*, 40(2):163–185, 2007.

[dlH97] Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

[dlHO06] Colin de la Higuera and Jose Oncina. Learning context-free languages. *Artificial Intelligence Review*, 2006.

[DPS86] Marty Demetras, Kathryn Post, and Catherine Snow. Feedback to first language learners: The role of repetitions and clarification questions. *Journal of Child Language*, 13:275–292, 1986.

[DS86] Robert P. Daley and Carl H. Smith. On the complexity of inductive inference. *Information and Control*, 1-3(69):12–40, 1986.

[Fer00] Henning Fernau. Identification of function distinguishable languages. In Hiroki Arimura, Sanjay Jain, and Arun Sharma, editors, *Proc.* 11th *International Conference on Algorithmic Learning Theory (ALT '00)*, volume 1968 of *Lecture Notes in Artificial Intelligence*, pages 116–130, Berlin, Heidelberg, 2000. Springer.

[FK84] H. Fukuda and Kazuo Kamata. Inference of tree automata from sample set of trees. *International Journal of Computer and Information Sciences*, 13(3):177–196, 1984.

[Fu74] King-Sun Fu. *Syntactic Methods in Pattern Recognition.* Academic Press, New York, 1974.

[FZ96] Rusins Freivalds and Thomas Zeugmann. Co-learning of recursive languages from positive data. In *Proceedings of the Second International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 122–133, London, UK, 1996. Springer-Verlag.

[GJT03] Ricard Gavaldà, Klaus P. Jantke, and Eiji Takimoto, editors. *Proc.* 14th *International Conference on Algorithmic Learning Theory (ALT '03)*, volume 2842 of *Lecture Notes in Artificial Intelligence*, Berlin, Heidelberg, 2003. Springer-Verlag.

[GK91] Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. In M. Fulk and John Case, editors, *Proc.* 4th *Annual Workshop on Computational Learning Theory (COLT '91)*, pages 303–314, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol72] E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[Gol78] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS92] William I. Gasarch and Carl H. Smith. Learning via queries. *Journal of ACM*, 39(3):649–674, 1992.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[Hor69] James Jay Horning. A study of grammatical inference. Techn. Rep. 139, Univ. of Stanford, Dept. of Computer Science, 1969.

[HPRW96] Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, 1996.

[HPTS84] Kathryn Hirsh-Pasek, Rebecca Treiman, and Maita H. Schneiderman. Brown & Hanlon revisited: mothers' sensitivity to ungrammatical forms. *Journal of Child Language*, 11:81–89, 1984.

[HRPW95] Lisa Hellerstein, Vijay Raghavan, Krishnan Pillaipakkamnatt, and Dawn Wilkins. How many queries are needed to learn? In *Proc.* 27th *Annual ACM Symposium on Theory of Computing*, pages 190–199, New York, NY, USA, 1995. ACM Press.

[HS98] Vasant Honavar and Giora Slutski, editors. *Proc.* 4th *International Colloquium on Grammatical Inference (ICGI '98)*, volume 1433 of *Lecture Notes in Artificial Intelligence*, Berlin, Heidelberg, 1998. Springer-Verlag.

[HST07] Marcus Hutter, Rocco Servedio, and Eiji Takimoto, editors. *Proc.* 18th *International Conference on Algorithmic Learning Theory (ALT '07)*, volume 4754 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2007. Springer.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[HU90] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Jan91] Klaus P. Jantke. Monotonic and non-monotonic inductive inference of functions and patterns. In J. Dix, P.H. Schmitt, and K.P. Jantke, editors, *Proc.* 1st *1st International Workshop on Nonmonotonic and Inductive Logic*, volume 543, pages 161–177, Berlin, Heidelberg, 1991. Springer-Verlag.

[JK07] Sanjay Jain and Efim B. Kinber. One-shot learners using negative counterexamples and nearest positive examples. In Hutter et al. [HST07], pages 257–271.

[JK08] Sanjay Jain and Efim B. Kinber. Learning languages from positive data and negative counterexamples. *Journal of Computer and System Sciences*, 74(4):431–456, 2008.

[JL95] Klaus P. Jantke and Steffen Lange, editors. *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*, volume 961 of *Lecture Notes in Computer Science*, London, UK, 1995. Springer.

[JLZ05] Sanjay Jain, Steffen Lange, and Sandra Zilles. Gold-style and query learning under various constraints on the target class. In Sanjay Jain, Hans-Ulrich Simon, and Etsuji Tomita, editors, *Proc.* 16th *International Conference on Algorithmic Learning Theory (ALT '05)*, volume 3734 of *Lecture Notes in Computer Science*, pages 226–240, Berlin, Heidelberg, 2005. Springer.

[JLZ07] Sanjay Jain, Steffen Lange, and Sandra Zilles. A general comparison of language learning from examples and from queries. *Theoretical Computer Science*, 387(1):51–66, 2007.

[JP98] Hugues Juillé and Jordan B. Pollack. A stochastic search approach to grammar induction. In Honavar and Slutski [HS98], pages 126–137.

[Kin08] Efim Kinber. On learning regular expressions and patterns via membership and correction queries. In Clark et al. [CCM08], pages 125–138.

[KP89] Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proc.* 2nd *annual workshop on Computational learning theory (COLT '89)*, pages 57–71, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1994.

[KY97] Satoshi Kobayashi and Takashi Yokomori. Learning approximately regular languages with reversible languages. *Theoretical Computer Science*, 174(1–2):251–257, 1997.

[Lan90] Steffen Lange. A note on polynominal-time inference of k-variable pattern languages. In Jürgen Dix, Klaus P. Jantke, and Peter H. Schmitt, editors, *Nonmonotonic and Inductive Logic*, volume 543 of *Lecture Notes in Computer Science*, pages 178–183. Springer, 1990.

[Lan92] Kevin J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proc.* 5th *Annual Conference on Computational Learning Theory (COLT '92)*, pages 45–52, New York, NY, USA, 1992. ACM Press.

[Lan94] Steffen Lange. The representation of recursive languages and its impact on the efficiency of learning. In *Proc.* 7th *Annual Conference on Computational Learning Theory (COLT '94)*, pages 256–267, New York, NY, USA, 1994. ACM Press.

[Lan00] Steffen Lange. Algorithmic learning of recursive languages. Mensch und Buch Verlag, 2000.

[Lee96] Lillian Lee. Learning of context-free languages: a survey of the literature. Techn. Rep. TR-12-96, Harvard University, 1996.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[LG02] Steffen Lange and Gunter Grieser. On the power of incremental learning. *Theoretical Computer Science*, 288(2):277–307, 2002.

[LJ78] Leon S. Levy and Aravind K. Joshi. Skeletal structural descriptions. *Information and Control*, 39:192–211, 1978.

[LPP98] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In Honavar and Slutski [HS98], pages 1–12.

[LW90] Steffen Lange and Rolf Wiehagen. Polynomial-time inference of pattern languages. In *ALT*, pages 289–301, 1990.

[LW91] Steffen Lange and Rolf Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Comput.*, 8(4):361–370, 1991.

[LZ92] Steffen Lange and Thomas Zeugmann. Types of monotonic language learning and their characterization. In *Proc.* 5th *Annual Conference on Computational Learning Theory (COLT '92)*, pages 377–390, New York, NY, USA, 1992. ACM Press.

[LZ93a] Steffen Lange and Thomas Zeugmann. Language learning in dependence on the space of hypotheses. In *Proc.* 6th *Annual Conference on Computational Learning Theory (COLT '93)*, pages 127–136, New York, NY, USA, 1993. ACM Press.

[LZ93b] Steffen Lange and Thomas Zeugmann. Language learning with a bounded number of mind changes. In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *Proc.* 10th *Annual Symposium on Theoretical Aspects of Computer Science (STACS '93)*, volume 665 of *Lecture Notes in Computer Science*, pages 682–691, Berlin, Heidelberg, 1993. Springer-Verlag.

[LZ93c] Steffen Lange and Thomas Zeugmann. Learning recursive languages with bounded mind changes. *Int. J. Found. Comput. Sci.*, 4(2):157–178, 1993.

[LZ03] Steffen Lange and Sandra Zilles. On the learnability of erasing pattern languages in the query model. In Gavaldà et al. [GJT03], pages 129–143.

[LZ04a] Steffen Lange and Sandra Zilles. Comparison of query learning and gold-style learning in dependence of the hypothesis space. In Shai Ben-David, John Case, and Akira Maruoka, editors, *Proc.* 15th *International Conference on Algorithmic Learning Theory (ALT '04)*, volume 3244 of *Lecture Notes in Computer Science*, pages 99–113, Berlin, Heidelberg, 2004. Springer-Verlag.

[LZ04b] Steffen Lange and Sandra Zilles. Formal language identification: query learning vs. Gold-style learning. *Information Processing Letters*, 91(6):285–292, 2004.

[LZ04c] Steffen Lange and Sandra Zilles. Replacing limit learners with equally powerful one-shot query learners. In John Shawe-Taylor and Yoram Singer, editors, *Proc.* 17th *Annual Conference on Computational Learning Theory (COLT '04)*, volume 3120 of *Lecture Notes in Computer Science*, pages 155–169, Berlin, Heidelberg, 2004. Springer-Verlag.

[LZ05] Steffen Lange and Sandra Zilles. Relations between gold-style learning and query learning. *Information and Computation*, 203(2):211–237, 2005.

[Mar88] Assaf Marron. Learning pattern languages from a single initial example and from queries. In *COLT*, pages 345–358, 1988.

[Mas] Timothy Mason. The evidence from acquisition in extreme conditions. Available on: `www.timothyjpmason.com/WebPages/LangTeach/Licence/CM/OldLectures/L3_ExtremeCircs.htm`.

[Mit99] Victor Mitrana. Patterns and languages: An overview. *Grammars*, 2(2):149–173, 1999.

[MK87] Assaf Marron and Ker-I Ko. Identification of pattern languages from examples and queries. *Information and Computation*, 74(2):91–112, 1987.

[MS97] Satoshi Matsumoto and Ayumi Shinohara. Learning pattern languages using queries. In *EuroCOLT '97: Proceedings of the Third European Conference on Computational Learning Theory*, pages 185–197, London, UK, 1997. Springer-Verlag.

[MiTî08] Victor Mitrana and Cristina Tîrnăucă. New bounds for the query complexity of an algorithm that learns DFAs with correction and equivalence queries. Submitted to Information Processing Letters, 2008.

[Muk92a] Yasuhito Mukouchi. Characterization of finite identification. In Klaus P. Jantke, editor, *Proc. 3rd International Workshop on Analogical and Inductive Inference (AII '92)*, volume 642 of *Lecture Notes in Artificial Intelligence*, pages 260–267, London, UK, 1992. Springer.

[Muk92b] Yasuhito Mukouchi. Inductive inference with bounded mind changes. In Klaus P. Jantke S. Doshita, K. Furukawa and T. Nishida, editors, *Proc. 3rd Workshop on Algorithmic Learning Theory (ALT '92)*, volume 743 of *Lecture Notes in Artificial Intelligence*, pages 125–134, Berlin, Heidelberg, 1992. Springer.

[MVMP04] Carlos Martín-Vide, Victor Mitrana, and George Păun, editors. *Formal Languages and Applications.* Studies in Fuzzyness and Soft Computing 148. Springer-Verlag, Berlin, Heidelberg, 2004.

[Myh57] John Myhill. Finite automata and the representation of events. Technical Report TR-57-624, WADD, Wright Patterson AFB, Ohio, 1957.

[Ner58] Anil Nerode. Linear automaton transformations. In *Proceedings of the American Mathematical Society*, volume 9, pages 541–544, 1958.

[NL05] Jochen Nessel and Steffen Lange. Learning erasing pattern languages with queries. *Theoretical Computer Science*, 348(1):41–57, 2005.

[OG91] José Oncina and Pedro García. Inferring regular languages in polynomial update time. In Pérez de la Blanca, Sanfeliú, and Vidal, editors, *Pattern Recognition and Image Analysis*, pages 49–61. World Scientific Publishing, 1991.

[OG92] José Oncina and Pedro García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5, pages 99–108. World Scientific Publishing, 1992. volume 5 of Series in Machine Perception and Artificial Intelligence.

[OGV93] José Oncina, Pedro García, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

[Okh05] Alexander Okhotin. On language equations with symmetric difference. Techn. Rep. 734, TUCS, Turku University, 2005.

[PH97] Rajesh Parekh and Vasant G. Honavar. Learning DFA from simple examples. In Ming Li and Akira Maruoka, editors, *Proc.* 8th *International Conference on Algorithmic Learning Theory (ALT '97)*, volume 1316 of *Lecture Notes in Artificial Intelligence*, pages 116–131, London, UK, 1997. Springer-Verlag.

[Pit89] Leonard Pitt. Inductive inference, DFAs, and computational complexity. In *AII '89: Proceedings of the International Workshop on Analogical and Inductive Inference*, pages 18–44, London, UK, 1989. Springer-Verlag.

[PV88] Leonard Pitt and L.G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

[Ron95] Dana Ron. *Automata Learning and its Applications.* PhD thesis, Hebrew University, 1995.

[RP99] Douglas L.T. Rohde and David C. Plaut. Language acquisition in the absence of explicit negative evidence: How important is starting small. *Cognition*, 72:67–109, 1999.

[RS87] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata (extended abstract). In *Proc. of the* 28th *Annual Symposium on Foundations of Computer Science (FOCS '87)*, pages 78–87. IEEE, 1987.

[RS93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

[SA95] Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In Jantke and Lange [JL95], pages 259–291.

[Sak87a] Yasubumi Sakakibara. Inductive reference of logic programs based on algebraic semantics. Technical Report ICOT, TR-260 79, Fujitsu International Institute for Advanced Study of Social Information Science, 1987.

[Sak87b] Yasubumi Sakakibara. Inferring parsers of context-free languages from structural examples. Technical Report ICOT, TR-330 81, Fujitsu International Institute for Advanced Study of Social Information Science, 1987.

[Sak90] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76:223–242, 1990.

[Sak92] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.

[Sak95] Hiroshi Sakamoto. Language learning from membership queries and characteristic examples. In Klaus P Jantke, Takeshi Shinohara, and Thomas Zeugmann, editors, *Proc. 6th International Conference on Algorithmic Learning Theory (ALT '95)*, volume 997 of *Lecture Notes in Artificial Intelligence*, pages 55–65, Berlin, Heidelberg, 1995. Springer.

[Sal94] Arto Salomaa. Patterns. *Bulletin of the EATCS*, 54:194–206, 1994.

[Sal95] Arto Salomaa. Return to patterns. *Bulletin of the EATCS*, 55:144–157, 1995.

[SG94] José M. Sempere and Pedro García. A characterization of even linear languages and its application to the learning problem. In Carrasco and Oncina [CO94], pages 38–44.

[Sha83] Ehud Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA, 1983.

[Shi83] Takeshi Shinohara. Polynomial time inference of extended regular pattern languages. In *Proceedings of RIMS Symposium on Software Science and Engineering*, pages 115–127, London, UK, 1983. Springer-Verlag.

[Tîr08b] Cristina Tîrnăucă. Learning reversible languages from correction queries only. Available on: `http://grlmc-dfilrom.urv.cat/grlmc/PersonalPages/cristina/publications.htm`, 2008.

[Tîr08a] Cristina Tîrnăucă. A note on the relationship between different types of correction queries. In Clark et al. [CCM08], pages 213–223.

[Tak88] Yuji Takada. Grammatical interface for even linear languages based on control sets. *Information Processing Letters*, 28(4):193–199, 1988.

[Tak94] Yuji Takada. A hierarchy of language families learnable by regular language learners. In Carrasco and Oncina [CO94], pages 16–24.

[Tak95] Yuji Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123(1):138–145, 1995.

[TB73] Boris A. Trakhtenbrot and Janis M. Barzdin. *Finite Automata: behavior and synthesis*. North-Holland Publishing company, Amsterdam, 1973.

[TîKn07a] Cristina Tîrnăucă and Timo Knuutila. Polynomial time algorithms for learning k-reversible languages and pattern languages with correction queries. In Hutter et al. [HST07], pages 264–276.

[TîKn07b] Cristina Tîrnăucă and Timo Knuutila. Efficient language learning with correction queries. Technical Report 822, Turku Center for Computer Science, May 2007.

[TîKo07] Cristina Tîrnăucă and Satoshi Kobayashi. A characterization of the language classes learnable with correction queries. In J. Cai, S. Barry Cooper, and H. Zhu, editors, *Proc.* 4rd *International Conference on Theory and Applications of Models of Computation (TAMC '07)*, volume 4484 of *Lecture Notes in Computer Science*, pages 398–407, Berlin, Heidelberg, 2007. Springer-Verlag.

[TîKo09] Cristina Tîrnăucă and Satoshi Kobayashi. Necessary and sufficient conditions for learning with correction queries. To appear in Theoretical Computer Science, 2009.

[TîTî07] Cătălin Ionuţ Tîrnăucă and Cristina Tîrnăucă. Learning regular tree languages from correction and equivalence queries. *Journal of Automata, Languages and Combinatorics, Special Issue for WATA 2006*, 12(4), 2007.

[TTWT04] Yasuhiro Tajima, Etsuji Tomita, Mitsuo Wakatsuki, and Matsuaki Terada. Polynomial time learning of simple deterministic languages via queries and a representative sample. *Theoretical Computer Sciece*, 329(1-3):203–221, 2004.

[Tze89] Wen-Guey Tzeng. The equivalence and learning of probabilistic automata (extended abstract). In *Proc. of the* 30th *Annual Symposium on Foundations of Computer Science (FOCS '89)*, pages 268–273. IEEE, 1989.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Wat90] Osamu Watanabe. A formal study of learning via queries. In Mike Paterson, editor, *Proc.* 17th *International Colloquium on Automata, Languages and Programming (ICALP '90)*, volume 443 of *Lecture Notes in Computer Science*, pages 139–152, Berlin, Heidelberg, 1990. Springer-Verlag.

[Wat94] Osamu Watanabe. A framework for polynomial time query learnability. *Mathematical Systems Theory*, 27:211–229, 1994.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of ACM*, 21(1):168–173, 1974.

[WG94] Osamu Watanabe and Ricard Gavaldà. Structural analysis of polynomial-time query learnability. *Mathematical Systems Theory*, 27(3):231–256, 1994.

[Yok94] Takashi Yokomori. Learning non-deterministic finite automata from queries and counterexamples. *Machine Intelligence*, 13:169–189, 1994.

[Yok96] Takashi Yokomori. Learning two-tape automata from queries and counterexamples. *Mathematical Systems Theory*, 29(3):259–270, 1996.

[Zeu06] Thomas Zeugmann. Inductive inference and language learning. In J. Cai, S. Barry Cooper, and A. Li, editors, *Proc.* 3rd *International Conference on Theory and Applications of Models of Computation (TAMC '06)*, volume 3959 of *Lecture Notes in Computer Science*, pages 464–473, Berlin, Heidelberg, 2006. Springer-Verlag.

[ZL95] Thomas Zeugmann and Steffen Lange. A guided tour across the boundaries of learning recursive languages. In Jantke and Lange [JL95], pages 190–258.

[ZLK95] Thomas Zeugmann, Steffen Lange, and Shyam Kapur. Characterizations of monotonic and dual monotonic language learning. *Information and Computation*, 120(2):155–173, 1995.

Index