

Dissertation

Supervised Learning of Wrappers from Structured Data Sources

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung

von

O. Univ.-Prof. Dipl.-Ing. Dr. Georg Gottlob

E 184

Institut für Informationssysteme

eingereicht an der Technischen Universität Wien

Fakultät für technische Naturwissenschaften und Informatik

von

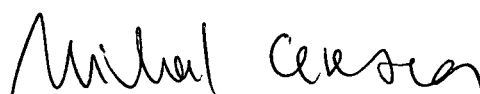
Mgr. Michal Ceresna

Zelena 12

91501 Nove Mesto nad Vahom

Slowakei

Wien, am 19. Mai 2005



Kurzfassung

HTML Wrapping ist eine häufig gebrauchte Strategie für den Zugriff und die Extraktion von Daten, die sich im World Wide Web befinden. HTML Wrapper lokalisieren die relevanten Daten in den Webseiten und transformieren sie in Formate, die für die weitere maschinelle Verarbeitung geeignet sind, wie zum Beispiel XML.

In den letzten Dekaden wurden verschiedene Methoden und Sprachen zur Generierung von HTML Wrappern vorgeschlagen und analysiert. Das reicht von hand-codierten Perl Programmen, Induktion von Stringautomaten und semi-automatischen visuellen Systemen mit speziell konstruierten Extraktionssprachen bis hin zu Data-Mining Methoden wie Support Vektor Maschinen, Bayesianischen Sortiermaschinen und Markov Modellen.

In der vorliegenden Dissertation konzentrieren wir uns auf das interaktive Lernen von Wrapper Programmen. Diese Systeme erstellen Wrapper basierend auf der visuellen Interaktion mit einem menschlichen Designer. Der Wrapperdesigner interagiert direkt mit der dargestellten Webseite, insbesondere durch Markierung von positiven und negativen Beispielinstanzen. Diese Beispielinstanzen dienen als Basis für die Generierung eines Wrappers. Die Semistruktur von Webseiten, die sich als DOM Baum darstellen läßt, erweist sich als sehr nützliches Hilfsmittel für die Datenextraktion. Deshalb interessieren wir uns insbesondere für Methoden des Lernens von Baumstrukturen, wo wir auf bessere Ergebnisse im Bereich der Erlernbarkeit solcher Strukturen hoffen als in vergleichbaren Lerntheorien auf flachen Strings.

Der erste Teil der Dissertation beschäftigt sich mit den existierenden Modellen des aktiven Lernens, die auf die interaktive Wrapper Generierung anwendbar sind. Wir studieren die theoretischen Grenzen der aktiven Erlernbarkeit, wir untersuchen die existierenden Algorithmen und vergleichen die relevanten Lernenmodelle.

Der zweite Teil der Dissertation beschäftigt sich mit dem aktiven Lernen von HTML Wrappern. Wir wählen die Sprache XPath als Formalismus für das Ausdrücken von HTML Wrappern und studieren die Erlernbarkeit von verschiedenen XPath Fragmenten. Danach präsentieren wir eine Methode

für Behandlung von HTML Attributen die auf dem Konzept der Entropie basiert. Diese Methode ermöglicht es, die vorgeschlagenen Algorithmen für das Lernen von Baumstrukturen mittels Abfragen mit anderen Methoden aus Data-Mining und Semantic Web, wie zum Beispiel Ontologien, Entscheidungsbäume oder der Bayesianischen Klassifikation, besser zu kombinieren.

Contents

Introduction	5
I Active Learning	9
1 Learning Models	10
1.1 Basic Concepts	10
1.2 Overview of Learning Models	11
1.3 Identification in the Limit	12
1.4 Identification from the Given Data (Characteristic Sets)	14
1.5 Teaching	18
1.6 Query Based Learning	20
1.6.1 Consistency Problem and Equivalence Queries	22
2 Bounds on Number of Required Queries	27
2.1 Bounds on Membership Queries	28
2.2 Bounds on Equivalence Queries	29
2.2.1 Halving-XEQ algorithm	31
2.2.2 Standard Optimal Algorithm	32
2.3 Bounds on Membership and Equivalence Queries	36
3 Measures for Learning Complexity	39
3.1 Teaching and Exclusion Dimension	39
3.1.1 Teaching Dimension	39
3.1.2 Exclusion Dimension	41
3.1.3 Replacing Equivalence Queries	42
3.2 Approximate Fingerprint Dimension	44
3.3 Vapnik-Chervonenkis Dimension	46
4 Learning of String Automata	51
4.1 Packs and Access Strings	51

4.2	Expanding of Pack/Adding Access Strings	55
4.3	DFA Learning with Membership and Equivalence Queries . . .	58
4.3.1	Complexity	63
4.4	Lower Bounds for Number of Queries	64
II	HTML Wrapping	69
5	Wrapper Induction and Learning	70
5.1	Wrapping Approaches	70
5.2	Interaction with the Wrapping System	74
6	Query Based Learning of XPath	79
6.1	XPath Definitions and Background	79
6.2	Boundaries of Interactive Wrapper Induction	80
6.3	Equivalence Queries	82
6.4	Membership Queries	94
6.5	Tree-prefix Queries	96
7	Classification of DOM Attributes	103
7.1	Entropy	104
7.2	Building Decision Trees	105
7.3	Wrapping with Attribute Classification	106
7.3.1	Clustering	106
7.3.2	Building Features	109
7.3.3	Training	111
	Conclusion	118
	Bibliography	122

Introduction

HTML wrapping is a commonly adopted strategy for accessing and extracting data located on the Web. HTML wrappers locate relevant data in Web pages and transform them into formats suitable for further machine processing such as XML.

In the last decade various approaches to creating HTML wrappers have been researched. These range from hand-coded Perl programs, induction of string based automata and semi-automatic visual systems with specially designed wrapping languages to data mining approaches such as support vector machines, Bayesian classifiers and hidden Markov models.

In this thesis, we focus on the learning process of interactive wrapper generators. These are systems that create wrappers from visual interaction with a human wrapper designer. The wrapper designer interacts directly with the rendered Web page, marking positive and negative example instances. These example instances are then used to generate the wrapper. A tree-structured representation of Web pages (the DOM tree) has proved to be useful for HTML wrappers. We therefore are specifically interested in tree learning techniques, where we hope for better wrapper learnability results in comparison to flat string approaches.

The first part of this thesis is devoted to existing active learning models applicable to interactive wrapper generation. We study the theoretical bounds of active learnability, analyse existing algorithms and compare the existing learning models. Through this part we assume familiarity of the reader with basic concepts from the complexity theory, theory of formal languages and boolean functions.

The second part of this thesis focuses on the active learning of HTML wrappers. We choose the XPath language as the formalism for expressing HTML wrappers and study the learnability of various XPath fragments. At the end, we present a method of dealing with HTML attributes. This method allows us to better combine the proposed algorithms for learning of tree shapes with other approaches from data mining and semantic Web research, for example ontologies, decision trees and Bayesian classification. Familiarity

of the reader with basic concepts of XML, XPath and DOM standards is assumed through this part of the thesis.

Chapter 1 introduces elementary concepts from machine learning. We define samples, hypothesis spaces, concepts and their representations. Then we present the learning models related to active learning: identification in the limit, identification from characteristic sets, teaching and query based learning. For each of these models we show their basic properties and their well known results. We start with the model of identification in the limit, which has most of the negative learnability results, because it is too generic. Next we study the model of identification from characteristic sets which, in contrast, has most of the positive polynomial learnability results, because it assumes a very strong condition of receiving a teaching set from the user. Additionally, we prove the non-learnability of NFA and super-classes in this model, and this also implies the non-learnability of NFA for other weaker learning models.

With the query based model we try to remove the assumption of requiring a teaching set. We treat the problem of an adversely constructed example sequence by empowering the learning algorithm with the possibility to ask queries about the target concept.

In the second chapter, we investigate generic query based learning algorithms. We abstract from computational time and space and study the lower and upper bounds for the number of queries that are required if using membership only, equivalence only, or equivalence and membership queries. For each of these cases we give algorithms or examples where the lower and upper bounds are reached.

We also present the halving-XEQ algorithm capable of discovering any target concept with at most logarithmic complexity in number of extended equivalence queries with respect to the size of the hypothesis space.

Additionally, we investigate the standard optimal algorithm. This algorithm establishes a min-max theorem and thus provides us with non-obvious results about relations between the number of required extended equivalence queries and membership queries.

In Chapter 3 we continue with studying the properties of hypothesis spaces. We explore the combinatorial properties and measures that allow us to express how many queries are needed to learn in these hypothesis spaces.

We start with using only membership queries and define the teaching dimension; the number of examples required to prove that a concept belongs to the hypothesis space. Next we define the exclusion dimension; the number of examples required to prove that a concept does not belong to the hypothesis space. Afterwards we discuss the problem of replacing a proper or extended equivalence query with a sequence of membership queries. We in-

investigate how many queries are required for the replacement. This replacing technique, together with the teaching and exclusion dimensions, allows us to establish tight lower and upper bounds from the number of required membership only, or membership and equivalence queries. The next measure that we study is the approximate fingerprint dimension that establishes the lower and upper bounds when using only proper equivalence queries. This measure we later reuse in the context of HTML wrapper induction to prove the non-learnability of several XPath fragments using only equivalence queries.

At the end of the chapter we present the Vapnik-Chervonenkis dimension which is useful for establishing the lower bounds for the number of required membership and equivalence queries, and also for establishing the lower bounds for the number of examples required to be seen by the learning algorithm.

Chapter 4 is devoted to the L^* algorithm, the well known result of query based learning. This algorithm is capable of polynomial learning of deterministic string automata using membership and equivalence queries.

We gradually build the necessary background required for explanation of the algorithm. Therefore, we define the notions of observation packs, access strings, is-like strings and escaping. Then we show the isomorphism between the minimal deterministic automaton accepting the target regular language being learned and the group of access strings, and therefore prove correctness of the algorithm. Afterwards we present the algorithm itself, demonstrating its operation on an example. We complete the chapter discussing the lower bounds for the number of required membership and equivalence queries.

In the second part of this thesis we focus on HTML wrapping. We start with Chapter 5, where the various approaches to wrapping are described. We discuss their theoretical foundation and review the existing implementations.

We decide to focus on interactive HTML wrapper generators operating on DOM trees. We give examples for different kinds of instances being extracted in DOM trees and formalise the interaction between the user and the wrapping system.

Chapter 6 is dedicated to the learnability of the XPath language. Because the XPath language is nowadays present in many XML related applications, we choose it as the hypothesis space for wrapper induction. Moreover, the XPath language also has many sub-fragments that scale well with respect to the polynomially-to-exponentially enlarging translation into the finite tree automata. This was found to be useful when investigating the borders between learnability and non-learnability, because the deterministic tree automata are known to be identifiable from characteristic sets, but the nondeterministic tree automata are not. We show that using only equivalence queries, even surprisingly simple XPath fragments are not equivalence

query learnable. Next, we continue with the analysis of learning using only membership queries, and prove by reduction to the equivalence problem of the XPath fragments, that these queries are not sufficient. Summarising usage of the classical type of queries, we discover negative results if using only equivalence queries, and find difficulties with the visualisation of membership queries needed by the minimal adequate teacher. Therefore we propose the tree-prefix queries that can be naturally visualised on DOM trees (Web pages) and, together with the equivalence queries, still form a polynomial learning algorithm in reasonable XPath fragments.

In the last chapter we investigate handling of the DOM attributes. While it is still possible to view the attributes as internal DOM nodes and apply the algorithms from the previous chapter, we prefer to analyse an approach using decision tree classifiers. In practice, this approach seems to better deal with the semantics of HTML and the various features of the text content. We start the description with defining the notion of entropy and explain how the entropy-based classifiers are built. Then we describe the clustering of the instances, construction of the datasets from DOM attributes and the training of the attribute classifiers. Finally, we describe how to use the constructed clustering of the instances and the trained attribute classifiers for performing the information extraction.

Part I

Active Learning

Chapter 1

Learning Models

1.1 Basic Concepts

A domain, denoted as X , is a set of elements. A concept, denoted as c , is subset of the domain X and a hypothesis space C is set of concepts. For example, a domain is a set of all words over some finite alphabet, a concept is a regular language and a hypothesis space is the class of all regular languages.

A trivial example of a concept c is an element x from the domain X . It is either labelled positively if $x \in c$ or negatively if $x \notin c$.

The learning problem is the problem of a learner to identify the unknown concept chosen from the hypothesis space C . The concept which should be the result of learning is also called target concept and we will denote it as $c^* \in C$. Passive learning methods do not give a possibility for the learner to control which examples it receives. In active learning methods, the learner can influence the received examples, for example by asking queries and receiving counterexamples. It is assumed that the queries are answered by an oracle, called teacher.

A hypothesis space is considered to be learnable, if there exists a tractable learning algorithm that is able to discover any chosen target concept c^* from the hypothesis space C . A learning algorithm that is polynomial is usually considered as tractable. That means, the algorithm runs in polynomial time, sees at most polynomial number of examples, asks at most polynomial number of queries etc. Not all inputs are known to the learning algorithm at the beginning of the run. Therefore, the polynomial bound is computed with respect to the size of target concept c^* , size of the largest example seen so far or the sum of size for all examples seen so far.

The same concept can have several representations. For example, a concept being some regular language can be represented with a deterministic fi-

nite automaton, nondeterministic finite automaton or logic formula. Different representations have influence on polynomial learnability of the hypothesis spaces, because not all representation spaces are polynomially transformable between each other. For example, the hypothesis space of regular languages is polynomially learnable in the query based model if the representation space is DFA, but it is not learnable if the representation space is NFA. Therefore, if running time of a learning algorithm is computed with respect to the size of the target concept, it means the size of the smallest representation of the target concept c in the representation space R .

1.2 Overview of Learning Models

In the literature, various learning models have been already proposed and studied. The most widely known are:

- query-based learning [Ang88, Ang04]
- mistake bounded online learning [Lit88]
- identification from the given data [Gol67, Gol78]
- PAC (probably approximately correct) learning [KV94, Ant94]

One of the most studied concept classes are finite automata, because often they draw a border line between existence and non-existence of tractable learning algorithms in those models. Besides the generic learning frameworks listed above, other more specialised approaches, especially for the finite automata domain have been proposed:

- Modelling external environment using inference from homing sequences. That is, inference from (limited) information about the sequence of states that were passed by the automaton modelling the environment. These states are discovered while interacting with the environment that is executing some actions inside of it and interpreting the results.
- Stochastic models - make use of the known probabilities of transitions in the δ function of the automaton.

In this chapter, we will study some of these learning models, explore their properties and relations among them. Let us note that the most extensive research has been done for the PAC learning model of Valiant and query based learning model of Angluin.

As we will see, there is a huge amount of results about learnability of various concept classes such as finite automata, context free grammars, boolean functions in the above listed learning models. Nevertheless, it is interesting to briefly view these learning models also from the cognitive point of view. That is, how well do these models capture the character of learning by human beings? There are some properties of the learning by humans, nicely captured already by the current formal models:

- usage of oracles (asking queries and teachers) helps with learning;
- frequency (probabilistic distribution) of examples influences the results of learning;
- simple examples (shorter or for example in sense of Kolmogorov complexity) are better;
- there are examples (characteristic and teaching sets) that are especially well describing the target concept being learned.

One of the unnatural properties of the learning algorithms is that they are hard-coded to particular representation classes. So, most of them learn finite automata or boolean functions, but not concept classes with arbitrary representations. Despite of this drawback, there still do not exist learning algorithms capable of learning natural languages, even though young children can easily do this. Clearly, one asks a question: What makes it so difficult? Is the hardness hidden in the knowledge representation or in the incapable learning algorithms? Or is the problem caused by missing effective combination among them?

1.3 Identification in the Limit

The first learning model for identification of the languages has been proposed in the work of Gold [Gol67]. In his model, a learning algorithm reads a sequence of examples labelled with $+$ in case of a positive example and with $-$ in case of a negative example. The learned concept is either outputted after reading each example - the so called online learning mode or only after reading the whole sequence - the so called batch mode.

Definition 1.3.1. A complete presentation of the language L is an ordered (possibly infinite) sequence of labelled samples such that every example appears at least once.

An algorithm A identifies the language L in the limit, if on every complete presentation of the language L , the automaton A converges to the correct target concept.

There are two results from Gold binding the tractable cases of the language identification in the limit.

Theorem 1.3.2. *[Gol67] No super-finite class of languages is identifiable in the limit from positive data only.*

Informally, this theorem states that receiving infinitely many, but only positive examples is not sufficient. Therefore, any successful learning algorithm has to either deal also with the negative examples or besides observing the examples, it also has to make additional usage of other channels for getting information about the target concept. Two of such possible channels with information about the target concept are the example based queries, discussed in the following Section 1.6 and the assumptions about the probabilistic distribution of examples made in the PAC model. Moreover, the next negative results show us closer boundaries for existence of the tractable (polynomial) learning algorithm.

Theorem 1.3.3. *[Gol78, Ang78] Learning the smallest deterministic finite automaton consistent with an arbitrary set of positive and negative examples of a regular language L is NP-hard.*

Because finding the minimal DFA is intractable, there were attempts to find at least approximations of the minimal automaton, however also this leads to an NP-hard problem:

Theorem 1.3.4. *[PW89] If the minimal automaton M_L for the language L has n states, it is NP-hard for fixed $k \in \mathbb{N}$ to find a deterministic finite automaton with at most n^k states that is consistent with an arbitrary set of positive and negative examples of the language L .*

This complexity result applies also to all super-classes of DFA. As a consequence we obtain that just observing the positive and negative examples is for most of the interesting concept classes not sufficient.

The problem lies in the distribution of the received examples. In the model of identification in the limit we learn from arbitrary sequences of examples, this includes also such sequences, where many uninformative examples occur at the beginning and the good, informative examples are placed only very far - in the limit.

Various learning models treat the problem of sample distribution differently. This is also the reason why we obtain for them different, sometimes incomparable learnability results:

- identification from characteristic sets - relies on the environment (for example the teacher) that good examples are provided early to the learning algorithm;
- PAC learning - assumes that the examples appear according to some probabilistic distribution; the successfulness of the learning algorithm is measured according to this distribution;
- query based learning - does not assume anything about the target distribution, but empowers the learning algorithm with stronger weapons (queries) than just the observing of examples.

1.4 Identification from the Given Data (Characteristic Sets)

There is a problem with the sound definition of the polynomial learnability in Gold's model of learning in the limit, discussed in the above section. Therefore, a new framework for passive learning, called identification from the given data, has been proposed.

In the unbounded (non-polynomial) case, it is equivalent to the identification in the limit, but additionally it also consistently defines the polynomial learnability with respect to the number of seen examples, size of the largest example received so far and polynomial running time of the learning algorithm. In the following section, let $\|S\|$ denote the sum of lengths of all elements in the set S .

The model for identification from the given data implicitly assumes that good examples are provided in the input sequence read by the learning algorithm. Thanks to this restriction, there are weaker assumptions expected from the learning algorithm.

First, the learning algorithm is required, for each given set (S^+, S^-) of positive and negative examples of some concept $c \in C$, to return in polynomial time a consistent hypothesis $c' \in C$. Let us note that c' may not be necessarily equivalent to c . It is only required that c' is consistent with the given examples of c .

Second, it is required that for each target concept c exists a special set (S_c^+, S_c^-) , called characteristic set such that for this characteristic set and all its super-sets returns the learning algorithm the same correct target concept. Let us explicitly note, that the characteristic sets may vary for different target concepts. Formally defined:

Definition 1.4.1. A concept class C is identifiable in the limit from the polynomial time and data if and only if there exist a learning algorithm A and two polynomials $p()$ and $q()$ such that

1. given any set (S^+, S^-) of labelled examples of a concept $c \in C$, the learning algorithm A returns in time $p(\|S^+ \cup S^-\|)$ a concept $c' \in C$ consistent with the set (S^+, S^-) ;
2. for each concept $c \in C$, there exists a characteristic set (S_c^+, S_c^-) of size less than $q(|c|)$ such that for every superset (S^+, S^-) , $S^+ \supseteq S_c^+$, $S^- \subseteq S_c^-$ of the characteristic set returns the learning algorithm A a concept c' equivalent to the concept c .

Of course, in the practical implementation we do not know the characteristic set of the target concept being learned. But, due to the monotone Property 2, in the Definition 1.4.1, we assume that after providing enough examples we reach a superset of the characteristic set and therefore, the correct target concept will be returned by the learning algorithm A .

The Algorithm 1 outlines the protocol of the communication between a teacher T and a learner L in the framework of identification from the given data.

Algorithm 1 Protocol of identification from the given data

$S^+ = \emptyset, S^- = \emptyset$

do

T : add more examples into S^+ or S^-

L : generate hypothesis H consistent with (S^+, S^-)

$L \rightarrow T$: return hypothesis H to the teacher

until teacher decides whether H is the correct target concept

One of the negative properties of the identification from the given data is that the learning algorithm does not know whether the outputted result is really equivalent to the target concept (that is the learning can be successfully ended). This happens, because it is assumed that the characteristic set will appear once in the input sequence, but we can not recognise that situation.

There are positive results for learning of regular languages in the framework of polynomial identification from the given data, when using deterministic representation of the concepts such as DFA or deterministic tree automata. This is thanks to the algorithm called RPNI (regular positive negative inference), proposed by Oncina and Garcia [GO92, GO93]. Given a superset of a characteristic set, the RPNI algorithm infers in polynomial

time the minimal deterministic finite automaton consistent with the given samples.

To outline briefly, the tree version of this algorithm (denoted as tRPNI) is based on state merging in a prefix-tree bottom-up tree automaton built from the given positive examples. The negative examples are used as guidelines to prevent merging of certain states. The tRPNI algorithm converges in the limit to the minimal bottom-up deterministic tree automaton accepting the language being learned.

Let us note that Niehren et al. [CLN04] show one of the possible ways for applying of the learning using the tRPNI algorithm to wrapper induction.

On the other side, the super-classes of the deterministic finite automata are known to be not identifiable with polynomial time and data. In particular, this is interesting in connection with the relation to other learning models such as the query based learning and online prediction, because it implies non-learnability also in these models.

Theorem 1.4.2. [Hig97] *Assuming $P \neq coNP$, the class of NFA is not identifiable from polynomial time and data.*

Proof. Let A be a learning algorithm for identification of the NFA from polynomial time and data. Then, let $p()$ and $q()$ be the polynomials corresponding to the learning algorithm A that exists according to Definition 1.4.1.

The equivalence problem for NFA is:

- Input: two nondeterministic finite automata A_1 and A_2
 Goal: return the answer “yes”, if the languages defined by these automata are equal ($L(A_1) = L(A_2)$).

This problem is known to be coNP-complete [GJ90], even for the case of an input alphabet with a single letter. Let us denote this letter as ‘a’.

In the following part, we show, how this NFA equivalence problem can be reduced to the problem of identification from polynomial time and data.

Let A_1 and A_2 be two arbitrary nondeterministic finite automata (two concepts). Definition 1.4.1 implies that for the automata A_1 and A_2 there exist characteristic sets:

- $(S_{A_1}^+, S_{A_1}^-)$ of size $\mathcal{O}(q(|A_1|))$ such that the learning algorithm A outputs an automaton equivalent to A_1
 $(S_{A_2}^+, S_{A_2}^-)$ of size $\mathcal{O}(q(|A_2|))$ such that the learning algorithm A outputs an automaton equivalent to A_2

Let us analyse the following sets:

$$S_1^+ = \{a^k \mid k \leq q(|A_1|), a^k \in L(A_1)\}, \quad S_1^- = \{a^k \mid k \leq q(|A_1|), a^k \notin L(A_1)\}$$

$$S_2^+ = \{a^k \mid k \leq q(|A_2|), a^k \in L(A_2)\}, \quad S_2^- = \{a^k \mid k \leq q(|A_2|), a^k \notin L(A_2)\}$$

The sets (S_1^+, S_1^-) and (S_2^+, S_2^-) are consistent supersets of the characteristic sets $(S_{A_1}^+, S_{A_1}^-)$ respectively $(S_{A_2}^+, S_{A_2}^-)$, defined above. Clearly, these sets can be constructed in polynomial time.

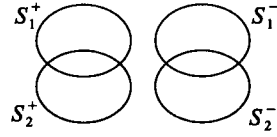
Now, two cases are possible:

1. $S_1^+ \cap S_2^- \neq \emptyset$ or $S_1^- \cap S_2^+ \neq \emptyset$

Then, there exists a sample in $S_1 \cap S_2$ proving that $L(A_1) \neq L(A_2)$ and therefore, the answer to the equivalence problem is “no”.

2. $S_1^+ \cap S_2^- = \emptyset$ and $S_1^- \cap S_2^+ = \emptyset$

The situation is depicted in the following figure:



We have:

$(S_{A_1}^+, S_{A_1}^-) \subseteq (S_1^+, S_1^-) \subseteq (S_1^+ \cup S_2^+, S_1^- \cup S_2^-)$ and therefore, the definition 1.4.1 implies that the learning algorithm outputs an automaton A'_1 equivalent to the automaton A_1 .

Similarly,

$(S_{A_2}^+, S_{A_2}^-) \subseteq (S_2^+, S_2^-) \subseteq (S_1^+ \cup S_2^+, S_1^- \cup S_2^-)$ and therefore, the definition 1.4.1 implies that the learning algorithm outputs an automaton A'_2 equivalent to the automaton A_2 .

Because the learning algorithm is deterministic, on the input set $(S_1^+ \cup S_2^+, S_1^- \cup S_2^-)$ outputs always the same concept and therefore $A'_1 = A'_2$. Combining with the above equivalences, we have that A_1 is equivalent to A_2 ($A_1 \equiv A'_1 = A'_2 \equiv A_2$). Therefore, the answer to the equivalence problem is “yes”.

It can be decided in polynomial time, which of the above two cases occurred and therefore, the equivalence problem can be solved in polynomial time. \square

Corollary 1.4.3. *The following concept classes are not identifiable from polynomial time and data: NFA, CFG, nondeterministic tree automata, linear context grammars (LIN).*

Proof. Combining the technique in Theorem 1.4.2 and (co)NP-hard or harder equivalence problem of these classes. \square

The following table summarises results of identification from polynomial time and data:

concept class	polynomial learnability
DFA	yes [GO92]
deterministic tree automata	yes [GO93]
NFA	no (Corollary 1.4.3)
CFG	no (Corollary 1.4.3)
LIN	no (Corollary 1.4.3)

1.5 Teaching

The characteristic sets in the previous section have not been just arbitrary sets of examples. They were special, because of minimising the effort of the learner to discover the target concept by providing good examples.

So, there naturally a question is raised: Given a target concept $c^? \in C$, what is the minimal number of examples needed to be provided so that there is a learner capable of discovering the target concept $c^?$?

These questions have been studied in the work of Mathias and Goldman [GM96]. They define a teaching model resembling this behaviour. The biggest obstacle for providing a good definition of teaching is to avoid the so-called collusion. Collusion means the cooperation mode between the teacher T and learner L , where examples are used by them as an information channel to transfer an encoded representation of the target concept. Mathias and Goldman choose the conception of an adversary that introduces additional correctly labelled examples into the teaching set, following the goal of making the learner to fail with discovering the target concept. Formally their model is defined as:

Definition 1.5.1. A concept class C is $f()/g()$ T/L-learnable if and only if there exist a teacher T and a learner L such that for any adversary ADV the following teaching protocol succeeds:

1. Adversary ADV chooses a target concept $c^? \in C$.

2. Teacher T computes the set of examples S_T (called teaching set) of size at most $f(\|S_T\|)$.
3. Adversary ADV optionally adds correctly labelled examples to this set. That is, she constructs an augmented set $S_{ADV} = S_T \cup \{\text{added examples}\}$.
4. Learner L outputs in time $g(\|S_{ADV}\|)$ a concept c equivalent to the target concept $c^?$.

Definition 1.5.2. A concept class C is semi/poly T/L-learnable if and only if it fulfils the conditions of Definition 1.5.1 and additionally:

1. The size of the set S_T is polynomially bounded ¹.
2. The learner outputs a concept equivalent to the target concept $c^?$ in polynomial time. (The function g is a polynomial).

In the next theorem we show that giving the best possible teaching set is equivalent to providing the characteristic set, thus we show equivalence of identification from characteristic sets and semi/poly teachability.

A hypothesis space C is consistent-easy class of concepts, if there exists a polynomial algorithm that for any given set of labelled examples returns a consistent concept (not necessary minimal). All usual language classes such as DFA, NFA, CFG etc. are consistent-easy.

Theorem 1.5.3. [Hig97] *A consistent-easy concept class C is identifiable from polynomial time and data if and only if it is semi/poly T/L teachable.*

Proof. \Rightarrow Let $c^? \in C$ be any target concept chosen by the adversary ADV . Because the hypothesis space C is polynomially identifiable, there exist a polynomial learning algorithm A and a characteristic set $(S_{c^?}^+, S_{c^?}^-)$ of the target concept $c^?$ such that the algorithm outputs a concept c' equivalent to $c^?$ on any input that is a superset of the characteristic set $(S_{c^?}^+, S_{c^?}^-)$. Therefore, if the teacher returns the characteristic set $(S_{c^?}^+, S_{c^?}^-)$ as the teaching set and the learner uses the algorithm A , the hypothesis space C is semi/poly teachable.

\Leftarrow Because the hypothesis space C is consistent-easy, the condition 1 of Definition 1.4.1 is immediately fulfilled.

Let $c^? \in C$ be an arbitrary target concept. Because the hypothesis space C is semi/poly teachable, there exists a learning algorithm A and teaching set $S_T = (S_T^+, S_T^-)$ such that the algorithm A learns the concept $c^?$ with this teaching set. Due to the adversary behaviour of ADV in the definition

¹but its computation may take arbitrary long

1.5.1, the learner returns a concept c' equivalent to the target concept $c^?$ for any superset of (S_T^+, S_T^-) . Therefore, the hypothesis space C is polynomially identifiable with the learning algorithm A and using the characteristic set (S_T^+, S_T^-) for the concept $c^?$. \square

1.6 Query Based Learning

The active form of learning is a configuration with class of concepts C , teacher T and learner L . The goal is for the learner to discover the target concept $c^? \in C$ chosen by the teacher. To achieve this goal, the learner receives examples from the teacher and is additionally allowed to pose various type of questions (queries). We can formalise the learning protocol into the following scheme:

Algorithm 2 Protocol of query based learning

```

while learning is running
  if  $L$  decided to pose a query
     $L \rightarrow T$ : ask query
     $T \rightarrow L$ : answer to query
                (usually for negative answer
                return also a counterexample)
  else
     $T \rightarrow L$ : provide next example

```

return the learned concept c

In general, example based queries are questions of the form:

$$\forall x_1, x_2 \dots x_n \in X : \Phi(x_1, x_2, \dots, x_n)$$

Answer to an example based query is either yes or no with a counterexample $(z_1, z \dots z_k) \in X^k$ such that $\neg \Phi(z_1, z_2, \dots, z_n)$.

Representatives of those queries are equivalence, membership, subset, superset or disjointness queries. The most common types of the queries are formalised in the following list:

- membership query is a question of the form
 - input: example x
 - query: does the example x belong to the target concept $c^?$
 $x \in c^?$
 - answer: yes / no

- equivalence query is a question of the form

input: concept c
 symbolic: $c \equiv c^?$
 query: is the concept c equivalent to the target concept $c^?$
 $\forall x : x \in c \iff x \in c^?$
 answer: yes
 no + counterexample x such that $x \in c \iff x \notin c^?$

- subset or superset query is a question of the form

input: concept c
 symbolic: $c \subseteq c^?$ ($c \supseteq c^?$)
 query: is the concept c subset (superset) of the target concept $c^?$
 $\forall x : x \in c \Rightarrow x \in c^?$ ($\forall x : x \in c^? \Rightarrow x \in c$)
 answer: yes
 no + counterexample x such that $x \in c \wedge x \notin c^?$
 ($x \notin c \wedge x \in c^?$)

Many hypothesis spaces are known to be polynomially learnable if the learning algorithm is allowed to ask both membership and equivalence queries. This setting is called minimally adequate teacher [Ang87] and examples of the learnable classes are deterministic string automata [Ang87, BDGW97], deterministic tree automata [Sak90] or geometric concepts [BGM99]. Interestingly, the hypothesis spaces of DNF are known to be learnable using the subset and superset queries, but not learnable using only equivalence queries and it is an open problem whether membership and equivalence queries are sufficient.

Definition 1.6.1. A concept class C is learnable using example based queries if there exists an algorithm A such that for any target concept $c^? \in C$ the algorithm A returns a concept $c' \in C$ such that the concepts c' and $c^?$ are equivalent.

Definition 1.6.2. A concept class C is polynomially learnable using example based queries if there exist polynomials $p_1()$ and $p_2()$ such that for any target concept $c^? \in C$ at any point holds:

1. the number of the queries posed by the learning algorithm A is bounded by $p_1(|c^?|)$
2. the running time of the learning algorithm A is bounded by $p_2(|c^?|, l)$, where l is size of the largest counterexample received so far.

1.6.1 Consistency Problem and Equivalence Queries

The reasons for nonexistence of a polynomial query based learning algorithm for some hypothesis space waggles between two boundaries:

1. complexity theoretic barrier - the problem of finding a hypothesis consistent with the set of given examples in polynomial time
2. information theoretic barrier - the problem of discovering the target concept with polynomial number of queries, even if unlimited computational time is allowed.

The reasons from the second type barrier are closely studied in Chapter 3. Here we will study an example of the first reason. A necessary condition for the polynomial learnability we are going to present is based on the following observation. If it is hard to find any concept consistent with the given set of samples, then it is even more harder to discover the correct target concept among all the concepts consistent with the seen examples.

Definition 1.6.3. Let $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ such that $x_i \in D$, $y_i \in \{+, -\}$ be a sequence of labelled examples from domain D . A concept c is consistent with the input sequence if $\forall i \ x_i \in c \iff y_i = +$.

Definition 1.6.4. Let C be a class of concepts. The consistency problem for C is a task for the given input sequence $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ to find a consistent concept $c \in C$ or return “no” if no such concept exists in C .

Theorem 1.6.5. [AHHP98] *Let C be a class of concepts which is polynomially size bounded and membership of samples and concepts can be tested in polynomial time. If C is polynomially exactly learnable using only equivalence queries then the consistency problem for C is in P .*

Proof. Let C be polynomial time learnable class of concepts. Then there exists a polynomial learning algorithm A for the class C . Without loss of generality we may assume that A always asks an equivalence query before returning the learned concept $c^?$. Then using the learning algorithm A we can construct an algorithm A' solving the consistency problem for C as outlined in Algorithm 3.

It is straightforward to verify that this algorithm solves the consistency problem and runs in polynomial time because the learning algorithm A is polynomial. Note that we have to test the consistency also for the learned concept c^* because it is not guaranteed which concept the learning algorithm A returns if there is no consistent concept for the given input sequence. \square

Algorithm 3 Solving consistency problem using learnability

```
while simulation of  $A$  is running
    simulate next step of  $A$ 

    if  $A$  asks an equivalence query with concept  $c$ 
        if  $c$  is consistent with input sequence
            # (for all  $i = 1 \dots n$  holds  $c(x_i) = y_i$ )
            return concept  $c$ 
        else
            answer to  $A$  a counter example  $x_i$  such that  $c(x_i) \neq y_i$ 
    else if  $A$  outputs a concept  $c^*$ 
        #  $c^*$  is result of the learning run
        if  $c^*$  is consistent with the input sequence
            # (for all  $i = 1 \dots n$  holds  $c^*(x_i) = y_i$ )
            return concept  $c^*$ 
        else
            # there is no consistent concept
            return "no"
    else if  $A$  outputs "FAIL"
        # there is no consistent concept
        return "no"
```

Corollary 1.6.6. *Assuming $P \neq NP$, if the consistency problem is NP -hard, then C is not polynomial-time learnable using equivalence queries.*

Proof. If the consistency problem for C is NP -hard, then every NP -problem is polynomially reducible to it. Therefore, if $P \neq NP$ also $C \notin P$. Then Theorem 1.6.5 implies that C is not polynomially exactly learnable using only equivalence queries. \square

The above corollary implies that all consistency hard problems, for example the constraint satisfaction problems (CSP) are not polynomially learnable. We have summarised the known query based learnability results in Table 1.1 and Figure 1.1.

As shown in the next theorem, learnability with queries implies identification from polynomial time and data. The reverse implication is not true, an negative example is the class of DFA.

Theorem 1.6.7. *Every hypothesis space C that is learnable using example based queries is identifiable from polynomial time and data.*

Proof. Let A be a polynomial learning algorithm for C using example based queries. Let $c^? \in C$ be any target concept. The characteristic set $(S_{c^?}^+, S_{c^?}^-)$ is the set of all counterexamples returned to the example based queries during simulation of the algorithm A while learning the concept $c^?$.

The algorithm A' for identification from the characteristic set simulates the algorithm A . Each time an example based query $\forall x_1, x_2 \dots x_n \in X : \Phi(x_1, x_2, \dots, x_n)$ is going to be asked by the algorithm A , the algorithm A' computes sequentially the value of Φ for all examples in $S_{c^?}^+ \cup S_{c^?}^-$ and compares it with the classification into $S_{c^?}^+$ and $S_{c^?}^-$. If the classification mismatches the query is answered with “no” and the found counter example is returned. \square

Table 1.1: Learnability with example based queries

C	H	MQ	EQ	$MQ \& EQ$	$MQ \& XEQ$	PAC
DFA	NFA	no [Ang82, AK91]	no [Ang90]	yes [Ang87]		
DFA			no [Ang90]			
NFA			no [Ang90]	no [Hig97]		
CFG			no [Ang90]	no [Hig97]		
deterministic bottom-up tree	DNF		no [Ang90]	yes [Sak90]		
nondeterministic bottom-up tree			no [Ang90]	no [Hig97]		
decision trees			no [Han89]	?	yes [Bsh95]	?
decision trees $\subset DNF$			no [Han89]			
monotone k-term DL		yes [GLR97]				
k-term monotone DNF		yes [CGL97]				
k-CNF			yes [Ang88]			
k-DNF			yes [Ang88]			
DNF			no [Ang90]		yes- [Bsh95]	?
CNF			no [Ang90]			

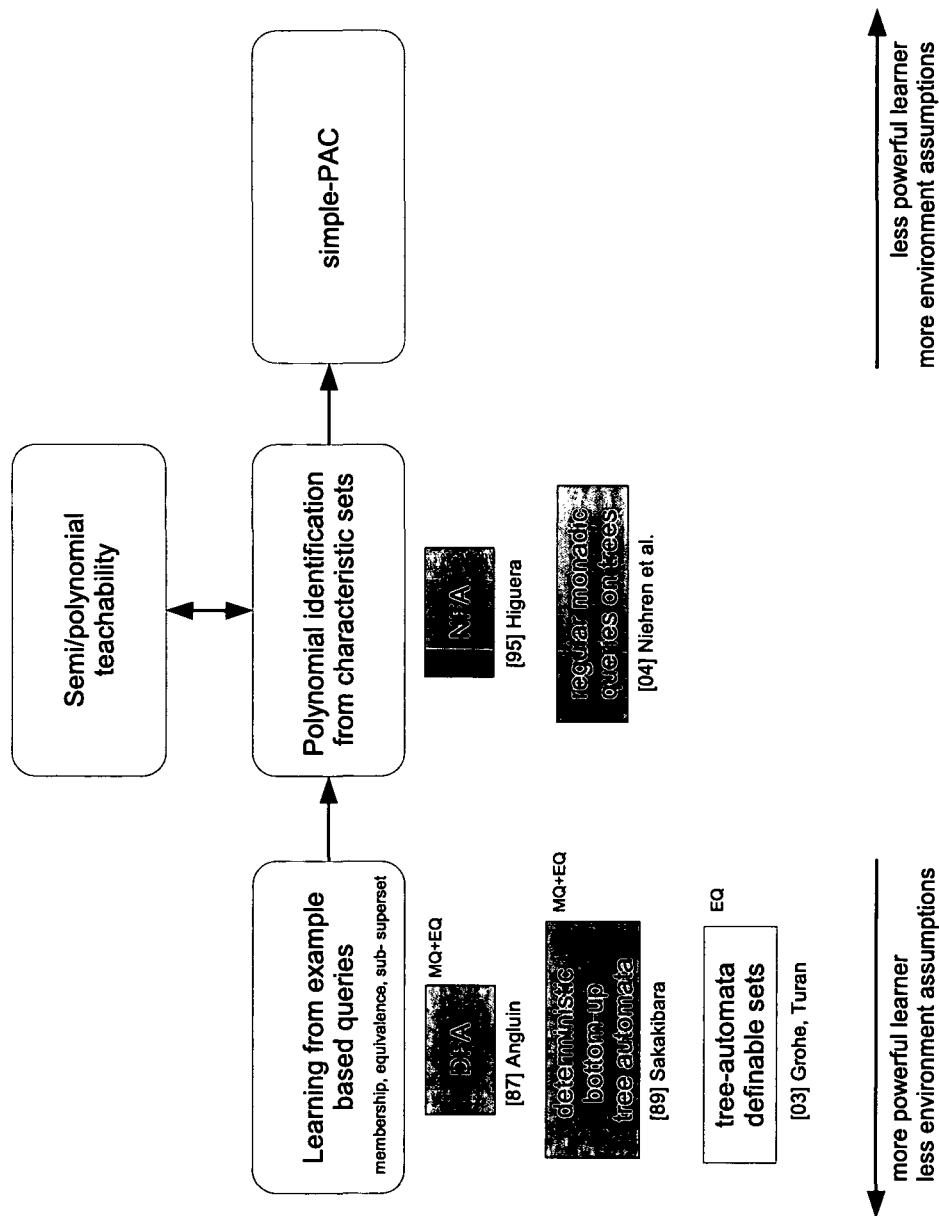


Figure 1.1: Summary of relations between the models

Chapter 2

Bounds on Number of Required Queries

In this chapter, we abstract from the computational time and space complexity and study how many queries are needed to learn concept classes. We will also introduce several generic query-based learning algorithms using different type of queries.

All the learning problems, we are going to analyse, will be defined over a finite value domain of samples X . Concepts will be subset of the domain X and the hypothesis space will be a finite set of concepts.

Because we focus only on combinatorial bounds for the number of required queries, it is possible to view the query-based algorithms as search trees for the hypothesis spaces. Each non-leaf node of that search tree corresponds to a posed query, be it membership, equivalence or any other type of query. Child edges of the node correspond to answers for the asked query. For each node, it is possible to assign a set of the remaining concepts which are consistent with all answers to the queries on the path from this node to the root node of the search tree.

Let $d(T)$ denote a depth of the algorithm (search tree) T . Let $d(c, T)$ denote the maximum depth in T of a leaf node that has singleton set $\{c\}$ as a set of its remaining consistent concepts.

In the following sections, we will give several examples for performance and query complexity of various learning algorithms. To reduce the text volume, we will try to recycle the following learning problem wherever possible:

Example 2.0.8. Shared learning problem

X - finite value domain for samples, $X = \{x_1, x_2, x_3, x_4, x_5\}$

$C_{2.0.8}$ - finite hypothesis space $C = \{c_1, c_2, c_3, c_4, c_5\}$

	x_1	x_2	x_3	x_4
$c_1 = \{x_1, x_2, x_3\}$	+	+	+	-
$c_2 = \{x_3, x_4\}$	-	-	+	+
$c_3 = \{x_1, x_3, x_4\}$	+	-	+	+
$c_4 = \{x_1, x_2, x_3, x_4\}$	+	+	+	+
$c_5 = \{x_1\}$	+	-	-	-

2.1 Bounds on Membership Queries

An MQ-algorithm T is allowed to pose only membership queries while discovering the target concept $c^?$. We denote as $T_{MQ}(C)$ the set of all MQ-algorithms which are successful on C . Let $\#MQ(C)$ be the number of queries required for unambiguous identification of any target concept $c^?$ from the hypothesis space C by the best MQ-algorithm T . Formally,

$$\#MQ(C) = \min_{T \in T_{MQ}(C)} \max_{c \in C} \text{depth}(c, T)$$

Example 2.1.1. For the above defined shared example hypothesis space $C_{2.0.8}$ is $\#MQ(C_{2.0.8}) = 3$. Clearly, there exist optimal and suboptimal MQ-algorithms.

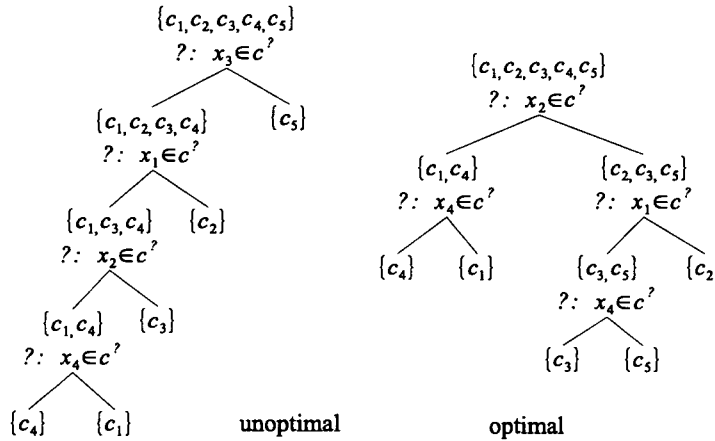


Figure 2.1: Unoptimal and optimal MQ-algorithms

The MQ-algorithms have the so-called partitioning property, that is every concept assigned to at most one leaf of T . Therefore, for any finite hypothesis space we have

$$|C| \leq 2^{\text{depth}(T)} \quad (2.1)$$

Note, that in case of finite hypothesis space there always exists a successful MQ-algorithm. This is the search tree T_{exhaust} which we get with exhaustively querying using all possible samples from the domain X and by subsequent pruning of the redundant membership queries. An example is depicted in the following figure:

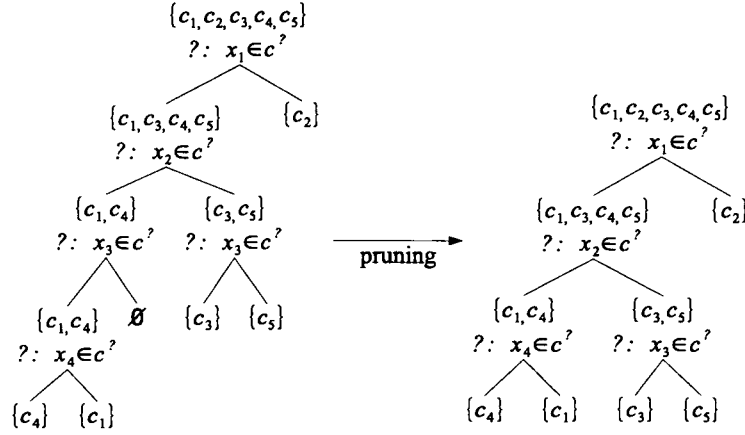


Figure 2.2: Pruning of membership queries

Clearly,

$$\max_{c \in C} \text{depth}(c, T_{\text{exhaust}}) \leq |X| \quad (2.2)$$

Therefore, combining 2.1 and 2.2 we have the next bound. Both, the upper and lower bound are tight if the hypothesis space C contains $2^{|X|}$ concepts.

$$\lg |C| \leq \#MQ(C) \leq |X| \quad (2.3)$$

2.2 Bounds on Equivalence Queries

For discovering of the target concept $c^?$ an EQ-algorithm T is allowed to pose only proper equivalence queries. The subject of a proper equivalence query must be some concept c from the hypothesis space C . An XEQ-algorithm is allowed to ask equivalence queries, where the subject of the query can be an arbitrary concept, not necessarily from the hypothesis space C . Let $\#EQ(C)$ (respectively $\#XEQ(C)$) be the number of queries required for unambiguous identification of any target concept $c^?$ from hypothesis space C by the best EQ-algorithm (XEQ-algorithm) T . Formally,

$$\#EQ(C) = \min_{T \in T_{EQ}(C)} \max_{c \in C} \text{depth}(c, T)$$

$$\#XEQ(C) = \min_{T \in T_{XEQ}(C)} \max_{c \in C} \text{depth}(c, T)$$

Example 2.2.1. For the above defined shared example hypothesis space $C_{2.0.8}$ is $\#EQ(C_{2.0.8}) = \#XEQ(C_{2.0.8}) = 2$. The next figure shows both optimal and unoptimal (X)EQ-algorithms.

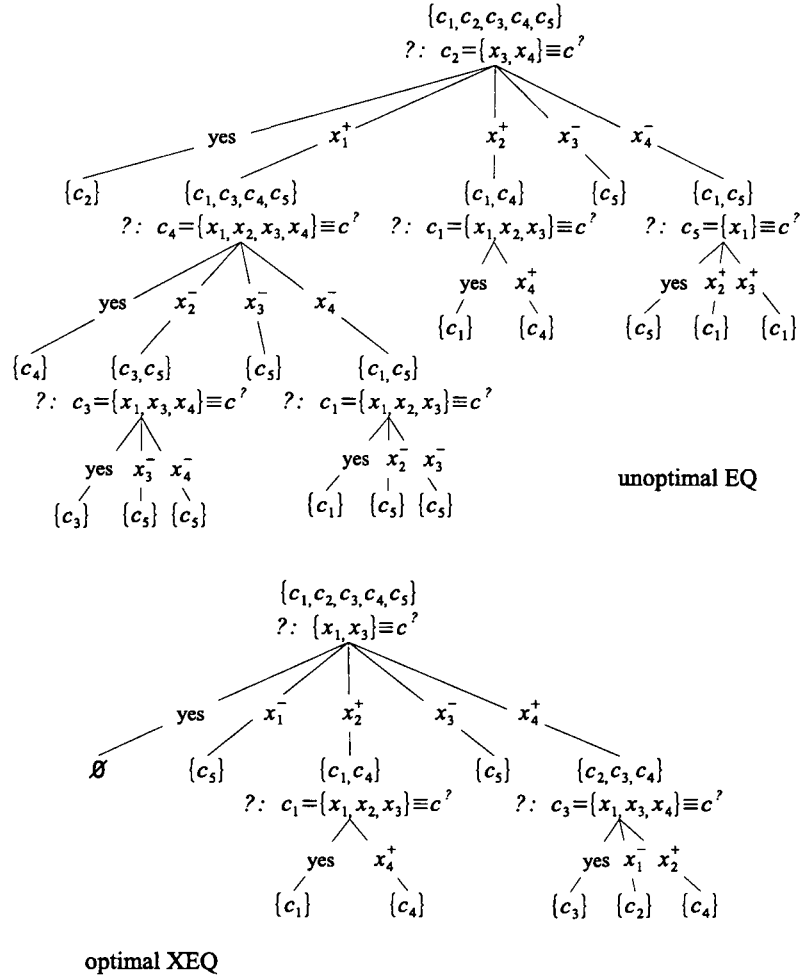


Figure 2.3: Unoptimal and optimal (X)EQ-algorithms

In case of the finite hypothesis space, there exists always a successful EQ-algorithm, because we can exhaustively search through all concepts in the hypothesis space. Therefore,

$$\#EQ(C) \leq |C| - 1 \quad (2.4)$$

Example 2.2.2. There exist concept classes, where this bound is tight. For example, for hypothesis spaces consisting of singletons (e.g. $C_{2.2.2} = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\}$), any EQ-algorithm must ask $|C| - 1$ equivalence queries. Interestingly, only one extended equivalence queries is needed, because we may ask a query $? : c^? \equiv \emptyset$ and the counterexample will be the target concept.

Because each EQ-algorithm is also an XEQ-algorithm, we have

$$\#XEQ(C) \leq \#EQ(C) \quad (2.5)$$

Therefore, combining (2.1) and (2.2) we obtain the following bounds

$$\#XEQ(C) \leq \#EQ(C) \leq |C| - 1 \quad (2.6)$$

2.2.1 Halving-XEQ algorithm

The idea of the binary search can be applied to the problem of learning with extended equivalence queries in finite hypothesis spaces. For dividing the hypothesis space into two halves, we repeatedly as a query use a so-called “majority vote” concept. For construction of this majority vote, the halving algorithm maintains a set of hypotheses C_{consist} , consistent with all so far received answers. An instance x from the domain X is part of the majority vote concept, if it belongs to at least half of all remaining consistent hypothesis. Formally,

$$\text{Majority Vote} = \left\{ x \mid x \in X \wedge |\{c \mid c \in C_{\text{consist}} \wedge x \in c\}| > \frac{|C_{\text{consist}}|}{2} \right\}$$

Due to this construction any counter example received to the query with majority vote disqualifies at least one half of the remaining consistent hypotheses. Therefore, we have the following upper bound for the number of required extended equivalence queries:

$$\#XEQ \leq (\lceil \lg |C| \rceil) \quad (2.7)$$

The following example shows that this bound is not tight, because the halving algorithm does not work well on hypothesis spaces with sparse concepts.

Example 2.2.3. Unoptimal halving algorithm. Let $C_{2.2.3}$ be the hypothesis space defined as

Algorithm 4 Halving-XEQ algorithm

```

 $C_{\text{consist}} = C$ 
while  $|C_{\text{consist}}| \geq 2$  :
    construct majority vote  $c_{\text{query}}$ 
    ask the extended equivalence query  $? : c_{\text{query}} \equiv c?$ 
    if answer to query is "yes"
        return  $c_{\text{query}}$ 
    else
        # answer to query is "no" with counterexample  $x^-$ 
        update  $C_{\text{consist}}$  according to  $x^-$ 
# invariant:  $|C_{\text{consist}}| = 1$ 
return the last consistent hypothesis from  $C_{\text{consist}}$ 
    
```

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
c_1	+	-	-	-	-	-	-	-
c_2	-	+	-	-	-	-	-	-
c_3	-	-	+	-	-	-	-	-
c_4	-	-	-	+	-	-	-	-
c_5	-	-	-	-	+	-	-	-
c_6	-	-	-	-	-	+	-	-
c_7	-	-	-	-	-	+	-	+
c_8	-	-	-	-	-	+	+	-
c_9	-	-	-	-	-	+	+	+

Then $\#XEQ(C_{2.2.3}) = 2$, but the halving algorithm requires 3 queries.

2.2.2 Standard Optimal Algorithm

The standard optimal algorithm (Algorithm 5) was first presented by Littlestone [Lit88] in context of the mistake bounded learning model. Here, we present its modification in the terminology of query based learning.

The standard optimal algorithm, similarly as the halving algorithm, uses the idea of the worst case partitioning for the remaining consistent hypothesis. But here, the splitting criterion is the largest number of answers a teacher may answer to an arbitrary sequence of membership queries, while still having two or more concepts consistent with all those answers. The next min-max theorem formalises this splitting criterion and proves correctness of the standard optimal algorithm.

Algorithm 5 Standard optimal XEQ-algorithm

```

 $C_{\text{consist}} = C$ 
while  $|C_{\text{consist}}| \geq 2$  :
    construct concept  $c_{\text{query}}$ 
     $c_{\text{query}} = \{\}$ 
    for  $x \in X$ 
         $C_x^+ = \{c \mid c \in C_{\text{consist}} \wedge x \in c\}$ 
         $C_x^- = \{c \mid c \in C_{\text{consist}} \wedge x \notin c\}$ 

        if
            
$$\max_{T \in T_{MQ(C_x^+)}} \min_{c \in C_x^+} \text{depth}(c, T) > \max_{T \in T_{MQ(C_x^-)}} \min_{c \in C_x^-} \text{depth}(c, T)$$

        then
            add  $x$  into  $c_{\text{query}}$ 

    ask the extended equivalence query  $? : c_{\text{query}} \equiv c?$ 
    if answer to query is "yes"
        return  $c_{\text{query}}$ 
    else
        # answer to query is "no" with counterexample  $y$ ;
        # update  $C_{\text{consist}}$  according to  $y$ 
        if  $y \in c_{\text{query}}$ 
             $C_{\text{consist}} = C_y^-$ 
        else
             $C_{\text{consist}} = C_y^+$ 

# invariant:  $|C_{\text{consist}}| = 1$ 
return the last consistent hypothesis from  $C_{\text{consist}}$ 

```

Theorem 2.2.4. [Lit88] For every finite hypothesis space C over finite value domain X holds

$$\#XEQ(C) = \min_{T \in T_{XEQ(C)}} \max_{c \in C} \text{depth}(c, T) = \max_{T \in T_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$$

Proof. " \geq " Let C be any finite hypothesis space and let $\#XEQ(C)$ be the number of extended equivalence queries required by any XEQ-algorithm to learn any target concept from the hypothesis space C . We show that there is a strategy for an adversary teacher that forces any XEQ learning algorithm to ask at least

$$d = \max_{T \in T_{MQ}(C)} \min_{c \in C} \text{depth}(c, T)$$

queries. Because the value of $\max_{T \in \mathcal{T}_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$ is always defined, there must exist an MQ-algorithm such that the maximum value is reached for it. Let T_{\max} be that tree with the maximal value, that is

$$\min_{c \in C} \text{depth}(c, T_{\max}) = d$$

Let $\{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$ be an extended equivalence query asked by the learning algorithm. A sufficient strategy for the adversary is to maintain the current node x_i (position) in the tree T_{\max} in the following way: If the current node is a leaf, the adversary answers “yes”. Otherwise, she answers “no” and returns counterexample x_i^+ if $x_i \notin \{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$ and updates the current node to the left child of x_i . Or if $x_i \in \{x_{j_1}, x_{j_2}, \dots, x_{j_k}\}$ she again answers “no”, but returns counterexample x_i^- and updates the current node to the right child of x_i .

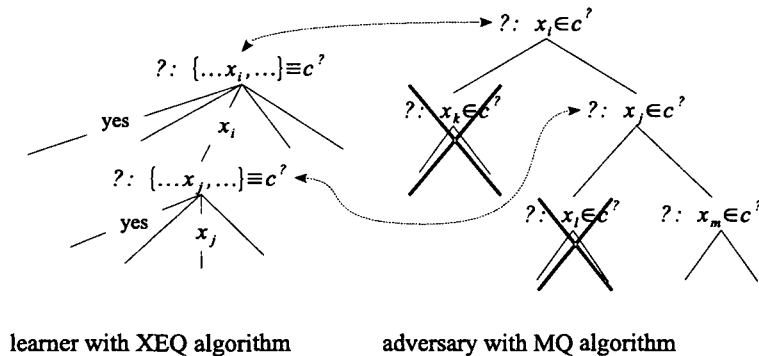


Figure 2.4: Min-max relation of MQ and XEQ algorithms

The definition of d implies that every leaf node in T_{\max} is in depth at least d . Therefore, the above described strategy of the adversary guarantees

that every XEQ-algorithm must ask at least d queries before it discovers any target concept c ?

" \leq " We show that the number of extended equivalence queries asked by the standard optimal algorithm is at most $\max_{T \in T_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$.

Every iteration of the while loop in the standard optimal algorithm (Algorithm 5) asks one extended equivalence query. Therefore, it is sufficient to show that the algorithm runs at most $\max_{T \in T_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$ iterations of the loop.

Let us analyse the value of D_{consist} defined as

$$D_{\text{consist}} := \max_{T \in T_{MQ(C_{\text{consist}})}} \min_{c \in C_{\text{consist}}} \text{depth}(c, T)$$

where C_{consist} is the variable used in the code of Algorithm 5.

Initially, we have

$$D_{\text{consist}} = \max_{T \in T_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$$

Let y be the counter example received during the current loop iteration. Assume that following equation would be true for y :

$$\max_{T \in T_{MQ(C_y^+)}} \min_{c \in C_y^+} \text{depth}(c, T) = \max_{T \in T_{MQ(C_y^-)}} \min_{c \in C_y^-} \text{depth}(c, T) = D_{\text{consist}}$$

Then, we can for the set of remaining consistent hypothesis C_{consist} , construct the following MQ-tree T_y . Root node is y and its subtrees are the maximal MQ-algorithms for C_y^+ and C_y^- . But then, because of the partitioning property of membership queries, we obtain

$$\min_{c \in C_{\text{consist}}} \text{depth}(c, T_y) = D_{\text{consist}} + 1$$

and this is a contradiction with D_{consist} being the maximum.

Because C_y^+ and C_y^- are subsets of C_{consist} , there remains only the case that

$$\min \left\{ \max_{T \in T_{MQ(C^+)}} \min_{c \in C^+} \text{depth}(c, T), \max_{T \in T_{MQ(C^-)}} \min_{c \in C^-} \text{depth}(c, T) \right\} < D_{\text{consist}}$$

We have chosen y to be the c_{query} so that y returned back as the counterexample rules out from C_y^+ , C_y^- the set with larger value. Therefore, the value of D_{consist} will decrease for the next iteration at least by 1.

After reaching $D_{\text{consist}} = 0$ there is only one remaining consistent hypothesis. Therefore, the algorithm will do at most $\max_{T \in T_{MQ(C)}} \min_{c \in C} \text{depth}(c, T)$ iterations. \square

2.3 Bounds on Membership and Equivalence Queries

An MQ&EQ-algorithm T is allowed to pose both membership and equivalence queries. Similarly, an MQ&XEQ-algorithm T is allowed to pose both membership and extended equivalence queries. Let $\#MQ\&EQ(C)$ ($\#MQ\&XEQ(C)$) be the number of queries required for unambiguous identification of any target concept c from hypothesis space C by the best MQ&EQ-algorithm (MQ&XEQ-algorithm) T . Formally,

$$\#MQ\&EQ(C) = \min_{T \in T_{MQ\&EQ}(C)} \max_{c \in C} \text{depth}(c, T)$$

$$\#MQ\&XEQ(C) = \min_{T \in T_{MQ\&XEQ}(C)} \max_{c \in C} \text{depth}(c, T)$$

Because adding new type of queries increases learning power, we have the following trivial bounds

$$\#MQ\&EQ(C) \leq \#MQ(C)$$

$$\#MQ\&EQ(C) \leq \#EQ(C)$$

$$\#MQ\&XEQ(C) \leq \#MQ(C)$$

$$\#MQ\&XEQ(C) \leq \#XEQ(C)$$

$$\#MQ\&XEQ(C) \leq \#MQ\&EQ(C)$$

To my knowledge, there are no stronger generic results valid for all concept classes that bound the number of membership and proper equivalence queries. However, as we will see in Chapter 4, there are results when restricted to particular concept classes such as deterministic finite automata.

Maas and Turan [MT92] showed that adding of the membership queries to the extended equivalence queries yields only logarithmic improvement.

Theorem 2.3.1. *For every finite hypothesis space C over finite value domain X holds*

$$\frac{\#XEQ(C)}{\lg(\#XEQ(C) + 1)} \leq \#MQ\&XEQ(C)$$

Proof. We show, that an adversary teacher has a strategy that forces every MQ&XEQ algorithm to ask at least $\#XEQ(C)/\lg(\#XEQ(C) + 1)$ queries.

From the min-max theorem 2.2.4 we have

$$\#XEQ(C) = \max_{T \in T_{XEQ}(C)} \min_{c \in C} \text{depth}(c, T)$$

Let T_{\max} be such a tree where this value is maximal. Every leaf of T_{\max} is in depth at least $\#XEQ$, because $\min_{c \in C} \text{depth}(c, T_{\max}) = \#XEQ$.

Let $N^{(j)}$ be the set of nodes n from T_{\max} such that n is exactly in depth $\#XEQ$ and its subtree contains at least one leaf node with assigned concept $c \in C$ that is consistent with all answers to the previous queries.

We show that after answering j queries the following invariant remains true:

$$|N^{(j)}| \geq \frac{2^{\#XEQ(C)}}{(\#XEQ(C) + 1)^j} \quad (2.8)$$

Initially, $N^{(0)}$ contains $2^{\#XEQ(C)}$ nodes.

If the j -th query is a membership query, we answer “yes” if and only if

$$|N^{(j+1)}| \geq \frac{1}{2}|N^{(j)}|$$

and the invariant will remain true.

Now let us assume the j -th query will be an extended equivalence query of the form $? : c_{\text{query}} \equiv c^?$. Without loss of generality c_{query} is consistent with all the previous answers, because otherwise we just recycle one of the previous answers.

Thanks to the partitioning property of membership queries and because T_{\max} is a complete binary tree up to the depth $\#XEQ$, there is a node $n \in N^{(j)}$ such that c_{query} is consistent with all the nodes on the path from node n to the root node of the tree T_{\max} . We choose the counterexample for the posed c_{query} using the following strategy.

By Dirichlet's principle, among the nodes on the path from n to the root, exists a node n' such that the subtree of its sibling contains at least $\frac{|N^{(j)}|}{\#XEQ+1}$ nodes from $N^{(j)}$. Let the label of the n' be x . Then we return x as the counterexample and the invariant 2.8 remains true. The situation is depicted in the following figure:

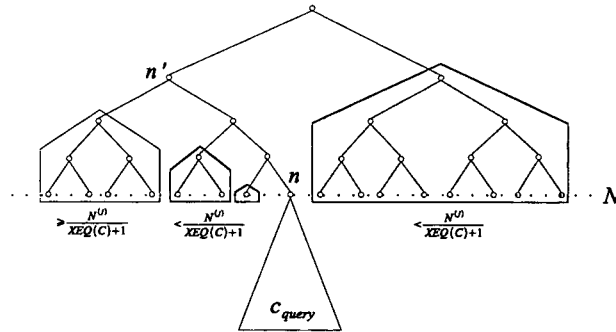


Figure 2.5: Adversary answers to MQ&XEQ algorithm

The MQ&XEQ algorithm must ask at least so many queries that the number of nodes in the set $N^{(j)}$ will be less than 2. Therefore,

$$\#MQ\&XEQ(C) \geq \min_j (|N^{(j)}| \leq 1) = \min_j \left(\frac{2^{\#XEQ(C)}}{(\#XEQ(C) + 1)^j} \leq 1 \right)$$

and so,

$$\frac{2^{\#XEQ(C)}}{(\#XEQ(C) + 1)^{\#MQ\&XEQ(C)}} \leq 1$$

□

Chapter 3

Measures for Learning Complexity

In the previous chapter we have studied the generic learning algorithms that work on arbitrary hypothesis space. In this chapter we extend the hypothesis space to act as a variable.

We analyse structure and combinatorial properties of hypothesis spaces, to find out which types of the hypothesis spaces make the learning harder. We define several measures that express complexity of the learning and establish lower and upper bound for these measures. The bounds are later applied in concrete hypothesis spaces to prove non-learnability results or the existence of learning algorithms.

3.1 Teaching and Exclusion Dimension

3.1.1 Teaching Dimension

Let C be a finite set of concept. Imagine the problem of convincing a sceptical learner that the target concept is some chosen $c \in C$. That is, how many examples do we require to prove that c is really the target concept?

A teaching set of a concept $c \in C$ is the minimal set of examples required to distinguish the concept c from all other concepts in the hypothesis space C .

Let $TD(C)$ denote the size of the largest teaching set required to distinguish a concept $c \in C$ from all the other concepts in C . Formally,

$$TD(C) = \max_{c \in C} \min_{T \in T_{MQ}(C)} \text{depth}(c, T)$$

Example 3.1.1. (Example 2.0.8 continued) The teaching dimension of hypothesis space $C_{2.0.8}$ is 3. Teaching sets for its concepts are the following:

concept	teaching set
c_1	$\{x_3, x_4\}$
c_2	$\{x_1\}$
c_3	$\{x_1, x_2, x_3\}$
c_4	$\{x_2, x_4\}$
c_5	$\{x_3\}$

Theorem 3.1.2. *For any finite hypothesis space C over a finite value domain X holds*

$$TD(C) \leq \#MQ(C)$$

Proof. Let T_{\min} be the best MQ-algorithm for the hypothesis space C . Let $c_{\max} \in C$ be some of its concepts with the largest teaching set. Thanks to the partitioning property of the membership queries, every concept c_{\max} is assigned only to one leaf node of T_{\min} . The depth of each concept c_{\max} must be at least $TD(c_{\max})$. Otherwise, we could construct a smaller teaching set for the concept c_{\max} consisting of samples asked in the membership queries on the path from the leaf node with c_{\max} to the root node of the tree T_{\min} . And this would be a contradiction to the maximality of $TD(C)$ for all c_{\max} nodes. \square

Rewriting the above theorem in terms of polynomial learning we obtain the following necessary condition:

Corollary 3.1.3. [GK95] *If a finite hypothesis space C over domain X is learnable with polynomial number of membership queries, then it has polynomial teaching dimension.*

Example 3.1.4. Known teaching dimensions for some concept classes [GK95]:

concept class	upper bound	lower bound
monotone k-term DNF (with l literals)	$l + 1$	$l + k$
k-term μ -DNF (with n variables)	n	$n + 2k$
monotone decision lists (with n variables)	n	$2n - 1$

3.1.2 Exclusion Dimension

Now, let us imagine the problem of convincing a sceptical user that some concept $c \in 2^X$ does not belong to the hypothesis space C . That is, how many examples are required to show that some concept $c \notin C$?

A 0-specifying set of a concept $\hat{c} \notin C$ is the minimal set of examples required to distinguish the concept \hat{c} from all concepts in the hypothesis space C . Let $XD_0(C)$ be the size of the largest 0-specifying set required to distinguish a concept $\hat{c} \notin C$ from all concepts in C . Formally,

$$XD_0(C) = \max_{\hat{c} \notin C} \min_{T \in T_{MQ}(C)} \text{depth}(\hat{c}, T)$$

Example 3.1.5. (Example 2.0.8 continued) the exclusion dimension of hypothesis space $C_{2.0.8}$ is 3. 0-specifying sets for concepts not from $C_{2.0.8}$ are the following:

concept	0-specifying set	concept	0-specifying set
\emptyset	$\{x_1, x_3\}$	$\{x_1, x_2\}$	$\{x_2, x_3\}$
$\{x_2\}$	$\{x_1, x_3\}$	$\{x_1, x_3\}$	$\{x_2, x_3, x_4\}$
$\{x_3\}$	$\{x_1, x_4\}$	$\{x_1, x_4\}$	$\{x_3, x_4\}$
$\{x_4\}$	$\{x_1, x_3\}$	$\{x_2, x_3\}$	$\{x_1, x_2\}$
$\{x_2, x_3, x_4\}$	$\{x_1, x_2\}$	$\{x_2, x_4\}$	$\{x_1, x_2\}$
$\{x_1, x_2, x_4\}$	$\{x_2, x_3\}$		

If the concept \hat{c} would belong to the hypothesis space, the 0-specifying set would match with the teaching set. Therefore,

$$XD_0(C) = \max_{\hat{c} \notin C} TD(C \cup \{\hat{c}\})$$

Theorem 3.1.6. For any finite hypothesis space C over a finite value domain X holds

$$XD_0(C) \leq \#MQ(C) + 1$$

Proof. Let T_{\min} be the best MQ-algorithm for the hypothesis space C . Let \hat{c} be some concept not from the hypothesis space C . Due to the partitioning property of membership queries, there is exactly one leaf node such that \hat{c} is consistent with all the membership queries on the path from this leaf node to the root node of the tree T_{\min} . Let c_C be the concept from the hypothesis space C assigned to that leaf node. Because $\hat{c} \notin C$, there must exist at least one example $x \in X$ such that $x \in \hat{c} \iff x \notin c_C$. Then, the set

consisting of examples asked in membership queries on the path from the leaf node with assigned c_C to the root of T_{\min} together with the example x form a 0-specifying set for \hat{c} . Therefore, for every concept not in C we can construct a 0-specifying set with at most $\#MQ(C) + 1$ elements and so, $XD_0(C) \leq \#MQ(C) + 1$. \square

Example 3.1.7. Let $C_{3.1.7}$ be a concept class consisting of binary vectors of the form $0^m\{0,1\}^n$. The teaching dimension and the exclusion dimension may be arbitrary different for $C_{3.1.7}$, as shown by this example. $TD(C_{3.1.7})$ can be made arbitrary $n \in \mathbb{N}$ and $XD(C_{3.1.7})$ will remain still 1.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
c_1	—	—	—	—	—	—	—
c_2	—	—	—	—	+	—	—
c_3	—	—	—	—	—	+	—
c_4	—	—	—	—	—	—	+
c_5	—	—	—	—	+	+	—
c_6	—	—	—	—	+	—	+
c_7	—	—	—	—	—	+	+
c_8	—	—	—	—	+	+	+

3.1.3 Replacing Equivalence Queries

The teaching and specifying sets are useful for replacing (extended) equivalence queries with a sequence of membership queries. The question is how many membership queries are required for such a replacement.

Assume we want to replace a proper equivalence query $? : c \equiv c^?$ with a sequence of membership queries. Let S_c be the teaching set of the concept c . From $TD(C)$ being maximal, we have $|S_c| \leq TD(C)$. Because concepts c and $c^?$ belong to the same hypothesis space C , the set S_c uniquely distinguishes the concept c_c from all concepts in C , including $c^?$. Therefore, each proper equivalence query can be replaced with a sequence of at most $TD(C)$ membership queries asked about samples in the set S_c . We answer the equivalence query with “yes”, if answers to all membership queries are consistent with c .

Now, let us analyse the case of replacing an extended equivalence query $? : \hat{c} \equiv c^?$ with a sequence of membership queries. Without loss of generality $\hat{c} \notin C$, otherwise we proceed as in the case above.

We define ≤ 1 -specifying set of concept $\hat{c} \notin C$ as the minimal set distinguishing the concept \hat{c} from all but at most one concept in C . Said differently,

≤ 1 -specifying set is such a set that \hat{c} is consistent with this set and additionally, zero or at most one concept from C are consistent with this set.

Let $XD_{\leq 1}(C)$ be the size of the largest ≤ 1 -specifying set required to distinguish a concept $\hat{c} \notin C$ from all but at most one concept in C .

Clearly, with adding zero or at most one more sample $x \in X$ into a ≤ 1 -specifying set we can construct a 0-specifying set for a concept $\hat{c} \notin C$. Therefore,

$$XD_0(C) \leq XD_{\leq 1}(C) + 1$$

Moreover, every 0-specifying set is also a ≤ 1 -specifying set and therefore,

$$XD_{\leq 1}(C) \leq XD_0(C)$$

The extended equivalence query can be replaced with a sequence of at most $XD_{\leq 1}(C)$ membership queries asked about samples in the ≤ 1 -specifying set of the concept $\hat{c} \notin C$. We answer this equivalence query always with “no”, because the concept \hat{c} is not from the hypothesis space C , but the target concept c^* is. A counterexample is either a sample from a membership query, where the answer is inconsistent with \hat{c} or a sample distinguishing the concept \hat{c} and the only possible concept consistent with all answers to the membership queries.

To summarise, for replacing an equivalence query, a sequence of at most $\max\{TD(C), XD_{\leq 1}(C)\}$ membership queries is needed. This measure we define as the extended teaching dimension XTD:

$$XTD(C) = \max\{TD(C), XD_{\leq 1}(C)\} = \max_{c \in 2^X} \min_{T \in T_{MQ}(C)} \text{depth}(c, T)$$

If we apply this technique of replacing the extended equivalence queries to the standard optimal algorithm, we obtain the following two theorems:

Theorem 3.1.8. *For any finite hypothesis space C over a finite domain X holds*

$$XTD(C) \leq \#MQ(C) \leq \#XEQ(C) XTD(C) \stackrel{(2.7)}{\leq} \lg |C| XTD(C)$$

Proof. The lower bound is implied from Theorems 3.1.2 and 3.1.6, the upper bound from replacing of the equivalence queries described above. \square

Theorem 3.1.9. *For any finite hypothesis space C over a finite domain X holds*

$$XD_{\leq 1}(C) \leq \#MQ\&EQ(C) \leq \#XEQ(C) XD_{\leq 1}(C) \stackrel{(2.7)}{\leq} \lg |C| XD_{\leq 1}(C)$$

Proof. To show the upper bound, we construct a MQ&EQ learning algorithm for C using at most $\#XEQ(C) XD_{\leq 1}(C)$ queries. We do this with simulating the standard optimal algorithm. Each time an extended equivalence query is asked with a concept $\hat{c} \notin C$, we replace it with a sequence of at most $XD_{\leq 1}(C)$ membership queries as described in the text above.

For the lower bound, we show a strategy for an adversary that forces any learning algorithm to ask at least $XD_{\leq 1}(C)$ queries. Let \hat{c}_{\max} be some of the concepts with ≤ 1 - specifying set of size $XD_{\leq 1}(C)$. The adversary answers questions consistently with \hat{c}_{\max} . After answering $XD_{\leq 1}(C) - 1$ queries at least two concepts in C must be consistent with all the previous answers. Otherwise there would be a smaller ≤ 1 - specifying set for the concept \hat{c}_{\max} . And so, any MQ&EQ-algorithm must ask at least $XD_{\leq 1}(C)$ queries. \square

As a corollary of the above theorem we have the next result for polynomial learning.

Corollary 3.1.10. [HPRW96] *The finite hypothesis space C over domain X is learnable with polynomial number of membership and equivalence queries if and only if it has polynomial exclusion dimension.*

3.2 Approximate Fingerprint Dimension

Assume that we use a concept $c_{\text{query}} \in C$ for a proper equivalence query $? : c_{\text{query}} \equiv c?$. Then each counterexample $x_{\text{ans}} \in X$ partitions the hypothesis space into two disjoint subsets of hypotheses, depending on their consistency with this answer. Therefore, the number of concepts that are at least disqualified with the answer x_{ans} to the query c_{query} is

$$\#\text{min-disqualified}(c_{\text{query}}, x_{\text{ans}}) = |\{c \in C \mid x_{\text{ans}} \in c \iff x_{\text{ans}} \in c_{\text{query}}\}| \geq 1$$

and there is some integer $1 \leq d_{x_{\text{ans}}} \leq |C|$ such that

$$\#\text{min-disqualified}(c_{\text{query}}, x_{\text{ans}}) \geq \frac{1}{d_{x_{\text{ans}}}} |C|$$

If the adversary teacher provides the best possible answer, that is the answer that disqualifies as few hypotheses as possible, from the previous equation results:

$$\#\text{min-disqualified}(c_{\text{query}}) = \min_{x \in X} \#\text{min-disqualified}(c_{\text{query}}, x)$$

and there is again some integer $d_{c_{\text{query}}}$ such that

$$\#\text{min-disqualified}(c_{\text{query}}) \geq \frac{1}{d_{c_{\text{query}}}} |C|$$

Definition 3.2.1. A subset $C' \subseteq C$ of the hypothesis space C is denoted as reachable from C if there exists a set of examples $X_{\text{reach}} = \{x_1, x_2 \dots x_r\}$ such that C' can be constructed from C by removing all hypotheses not consistent with the examples in the set X_{reach} .

Definition 3.2.2. The fingerprint dimension of the hypothesis space C is the least possible number $FD(C) \in \mathbb{N}$ such that for every reachable hypotheses subset $C' \subseteq C$ exists a proper equivalence query such that after any answer, the number of disqualified hypotheses is at least $\frac{1}{FD(C)}|C'|$.

Formally,

$$\forall C' \subseteq C, C' \text{ is reachable} : \left[\max_{c \in C'} \# \text{min-disqualified}(c) \right] \geq \frac{1}{FD(C)}|C'|$$

The situation with the minimal fragment of the disqualified hypotheses after each query is depicted in the next figure.

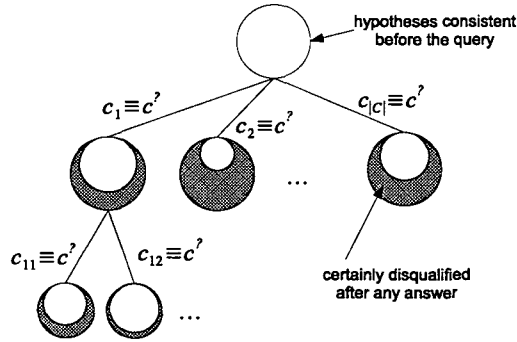


Figure 3.1: FD (fingerprint dimension) fraction of certainly disqualified hypotheses

Imagine a learning algorithm that in every step asks an equivalence query that disqualifies at least $\frac{1}{FD(C)}$ -fraction of the currently consistent hypotheses. Such query always exists, as implied from the definition of the fingerprint dimension. Then after k equivalence queries, the number of remaining consistent hypotheses is at most $\left(1 - \frac{1}{FD(C)}\right)^k |C|$. The learning algorithm must keep asking equivalence queries, while there are at least 2 remaining consistent hypotheses. Therefore,

$$\left(1 - \frac{1}{FD(C)}\right)^{\#EQ(C)} |C| \leq 1$$

which yields

$$\#EQ(C) \leq \lceil FD(C) \ln |C| \rceil \quad (3.1)$$

The fingerprint dimension defines also a lower bound for the number of required proper equivalence queries. We describe a strategy for an adversary teacher that forces this lower bound.

Because the value of $FD(C)$ is minimal there must be a reachable subset of hypotheses $C' \subseteq C$ such that for any proper equivalence query with a concept from C' there is an answer disqualifying at most $\frac{1}{FD(C)-1}|C'|$ hypotheses.

Assume that the learning algorithm asks a query $? : c \equiv c'$ with a concept:

- $c \in C'$ - Then according to the definition of $FD(C)$, there exists an answer $x \in X$ such that at most $\frac{1}{FD(C)-1}|C'|$ concepts are disqualified.
- $c \notin C'$ - Then among the examples x_1, \dots, x_r that prove the reachability of C' from the hypothesis space C must exist x_i such that $x_i \in c \iff x_i \notin c'$. Therefore, this x_i can be returned as a counterexample and the answer does not disqualify any concepts from C' .

Therefore, after k equivalence queries, the number of hypotheses consistent with the previous answers is at least

$$|C| - \frac{k}{FD(C)-1}|C|$$

Because the learning algorithm must keep asking queries until there is only one remaining consistent hypothesis, it follows:

$$|C| - \frac{k}{FD(C)-1}|C| \leq 1$$

and hence

$$\#EQ(C) \geq FD(C) - 1 \quad (3.2)$$

Combining (3.1) and (3.2) we finally retrieve:

$$FD(C) - 1 \leq \#EQ(C) \leq \lceil FD(C) \ln |C| \rceil$$

3.3 Vapnik-Chervonenkis Dimension

Consider set $S = \{x_1, x_2, \dots, x_m\}$ of m unlabelled samples from domain X . There are 2^m possible ways of division it into two disjoint and exhaustive subsets. As illustrated in Figure 3.2, each possible dividing of the sample S into these subsets is called partition. For example, the set $\{x_1, x_2\}$ partitions the sample set S into the pair $\langle \{x_1, x_2\}, \{x_3, \dots, x_m\} \rangle$.

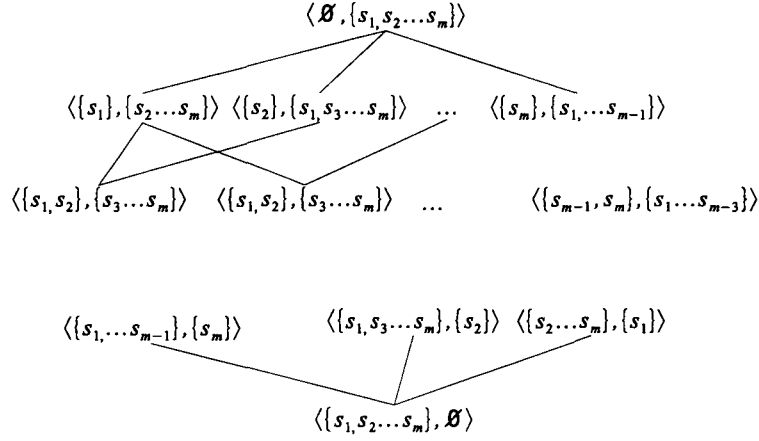


Figure 3.2: Division of samples

Definition 3.3.1. We say that the hypothesis space C shatters the set of samples S if and only if for every subset $T \subseteq S$ there exists a concept $c \in C$ which partitions S in the same way as T does. In symbolic form:

$$\forall T \subseteq S \exists c \in C \forall s \in S \quad s \in T \iff s \in c$$

or

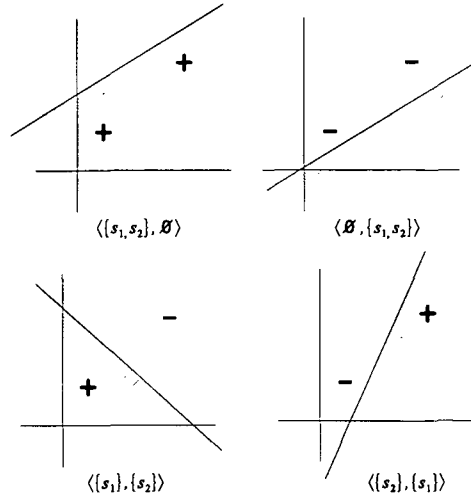
$$\{S \cap c \mid c \in C\} = \mathcal{P}(S)$$

Because for every subset of the shattered sample set S there exists a matching concept c and these concepts must be disjoint between each other, the definition of shattering implies that

$$|C| \geq 2^{|S|} \quad (3.3)$$

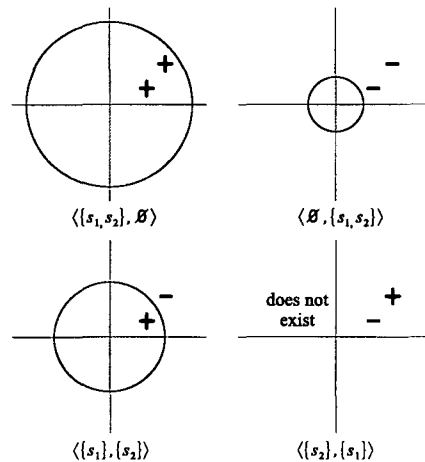
Example 3.3.2. Samples set S shattered by hypothesis space $C_{3.3.2}$:

- $C_{3.3.2}$ - set of all half-planes ($y \leq ax + b$)
- S - two points s_1, s_2 from \mathbb{R}^2 in general position



Example 3.3.3. Hypothesis space $C_{3.3.3}$ not shattering any sample set of size 2:

- $C_{3.3.3}$ - set of centred circles ($y \leq ax^2$)
- S - two points s_1, s_2 from \mathbb{R}^2 in general position



Example 3.3.4. Hypothesis space $C_{3.3.4}$ shattering any finite set of samples:

- $C_{3.3.4}$ - set of all XPath expressions
- S - Let $\langle \{i_1, i_2, \dots, i_n\}, \{j_1, j_2, \dots, j_n\} \rangle$ be a partition of some finite set of trees. Then the matching XPath expression is $i_1 \mid i_2 \mid \dots \mid i_n$.

Definition 3.3.5. The Vapnik-Chervonenkis Dimension of the hypothesis space C , denoted as $VCD(C)$ is the maximal number $n \in \mathbb{N}$ such that there is a set of samples with n elements shattered by the hypothesis space C .

Example 3.3.6. The examples above imply that there are hypothesis spaces with both finite and infinite VC-dimension. We have $VCD(C_{3.3.2}) = 4$, $VCD(C_{3.3.3}) = 1$, $VCD(C_{3.3.4}) = \infty$.

The interesting fact behind the Vapnik-Chervonenkis dimension is that it measures only the combinatorial parameters of the hypothesis space, independently from the computational complexity. Also, there is a well understood connection between of the VC dimension and the PAC learning model. The most known results are summarised in the following theorem.

Theorem 3.3.7. [BEHW89] *A hypothesis space C is PAC learnable (with unbounded complexity) if and only if it has finite VC dimension.*

Upper bound: Any hypothesis space C is properly PAC learnable if

- *there is an algorithm that outputs concept $c \in C$ consistent with the training set in polynomial time (depending on size of the target concept and number of seen examples)*
- *and the number of seen (available) samples is at least*

$$\max\left(\frac{4}{\delta} \lg \frac{2}{\delta}, VCD(C) \frac{8}{\delta} \lg \frac{13}{\delta}\right)$$

Lower bound: Any PAC learning algorithm must examine at least

$$\Omega\left(\frac{1}{\delta} \lg \frac{1}{\delta} + VCD(C)\right)$$

samples.

As the consequence of the above Theorem 3.3.7, example 3.3.4 and the transformation of EQ-learning algorithms into PAC learning algorithms [Ang87] the following statement results:

Corollary 3.3.8. *The hypothesis space of all XPath expressions is not PAC learnable and is not learnable using only equivalence queries.*

After seeing good connection between the PAC model and VC dimension, it is natural to ask about its relationship to the query based learning.

Combining the equations 3.3 and 2.3, we obtain for any finite hypothesis space C the following lower bound for the number of membership queries:

$$VCD(C) \leq \lg |C| \leq \#MQ(C)$$

Theorem 3.3.9. [Lit88] *For every finite hypothesis space C over finite domain X holds*

$$VCD(C) \leq \#XEQ(C) \leq \#EQ(C)$$

Proof. The definition of shattering implies that it is possible to construct an MQ-algorithm T_0 such that T_0 is up to depth $VCD(C)$ a complete binary tree. Therefore,

$$\min_{c \in C} d(c, T_0) \geq VCD(C)$$

Combining this inequality with Theorem 2.2.4 we have

$$\#XEQ(C) = \max_{T \in T_{MQ}(C)} \min_{c \in C} d(c, T) \geq \min_{c \in C} d(c, T_0) \geq VCD(C)$$

□

Again, this lower bound is not tight as shown by the following example.

Example 3.3.10. The VC-dimension is not the tight lower bound for $\#XEQ$.

$$\begin{aligned} X &- \text{finite value domain, } X = \{0, 1, 2, \dots, 2^{n-1}\} \\ C_{3.3.10} &- \text{finite set of concepts, } C_{3.3.10} = \\ &\quad \{c_0, c_1, \dots, c_{2^n-1}\}, \text{ where} \\ &\quad c_0 = \{0\} \\ &\quad c_1 = \{0, 1\} \\ &\quad c_2 = \{0, 1, 2\} \\ &\quad \dots \\ &\quad c_j = \{i \mid i \in \mathbb{N}_0 \wedge i \leq j\} \end{aligned}$$

$VCD(C_{3.3.10}) = 1$, because every pair of integers i and j , ($i < j$) with partition $< \{j\}, \{i\} >$ does not shatter the hypothesis space $C_{3.3.10}$.

The equation 2.7 implies $\#XEQ(C_{3.3.10}) \leq \lg |C_{3.3.10}| = n$. The binary search over the sorted array $[0, 1, \dots, 2^{n-1}]$ is a complete MQ-algorithm of depth n . Therefore, from the Theorem 2.2.4 is implied that $\#XEQ(C_{3.3.10}) \geq n$, and we have $\#XEQ(C_{3.3.10}) = n$.

Note, that [MT92] shows also another tighter lower bound for the Vapnik-Chervonenkis dimension

$$\frac{1}{7}VCD(C) \leq \#MQ\&EQ(C)$$

Chapter 4

Learning of String Automata

After studying the generic approaches and bounds for learnability, we turn our attention to specialised learning algorithms.

We will present one of the most known results in the active learning theory - the Angluin's polynomial algorithm for learning of deterministic finite automata using membership and equivalence queries [Ang87]. Let us remark that after the original work of Angluin, appeared several other modifications of the DFA learning algorithm for example reducing number of membership or equivalence queries [BDGW97], reducing space complexity [KV94] or adapted for training of robots [RS93].

After presenting the algorithm together with proofs and examples explaining why it works, we analyse its query complexity and the relation between the number of required membership and equivalence queries.

4.1 Packs and Access Strings

The DFA learning algorithm learns from observing of (input, output) examples. By an example we mean a pair $(x, +)$ or $(x, -)$, which expresses that the word x is (is not) member of a target language L that we want to learn. Set of positive/negative examples is called observation pack (P).

It is assumed, that the set of examples used for training is consistent. That is, there is no word y such that $(y, +) \in P$ and $(y, -) \in P$. In other words, there is no noise in the examples.

The examples in the observation pack are distributed into a finite system of sets (not necessarily disjoint) $P = (P_1, P_2, \dots, P_p)$, so that these two following conditions are fulfilled:

1. Let s_i be the shortest word in component P_i . Then word s_i is prefix of

all words in the component P_i , that is

$$\forall w \in P_i \quad \exists e \in \Sigma^* : w = s_i e$$

We will call s_i as an *access string* of the component P_i . We define the set of experiments E_i as a set of all suffixes of the access string s_i in the pack P_i , that is

$$E_i = \{e_i \mid s_i e_i \in P_i\}$$

2. For each pair of the components P_i, P_j from an observation pack $P = (P_1, P_2, \dots, P_p)$ exists a known experiment that distinguishes them. That is, there exists a suffix string e in both sets E_i and E_j such that the words $s_i e$ and $s_j e$ have different $+/-$ labels. Formally,

$$\forall P_i, P_j \quad \exists e \in E_i \cap E_j : s_i e \in L \iff s_j e \notin L$$

Where not clear from the context, we will put the access string into brackets (e.g. $[s]$) for better understanding. We denote S as set of all access strings in observation pack $P = (P_1, P_2, \dots, P_p)$.

$$S = \{s_i \mid s_i \text{ is accesstring of } P_i\}$$

Example 4.1.1. Observation pack

Let $L_{\text{ex}}^?$ be the unknown language we would like to learn. Of course, the learning algorithm does not know this language, we dispose it here only for better tracking and understanding of the next examples.

$$L_{\text{ex}}^? = ab^*ab$$

The minimal automaton (with no dangling transitions) accepting that language is $(\Sigma, Q, \delta, q_0, F)$, where:

$$\Sigma = \{a, b\},$$

$$Q = \{q, q, q, q, q\}, F = \{q_+\}$$

and the transition function δ is defined as:

$$\begin{array}{ll} \delta(q_0, a) = q_1 & \delta(q_3, a) = q_- \\ \delta(q_0, b) = q_- & \delta(q_3, b) = q_- \end{array}$$

$$\begin{array}{ll} \delta(q_1, a) = q_2 & \delta(q_-, a) = q_- \\ \delta(q_1, b) = q_1 & \delta(q_-, b) = q_- \end{array}$$

$$\begin{array}{l} \delta(q_2, a) = q_- \\ \delta(q_2, b) = q_3 \end{array}$$

Let us have the following observation pack of labelled samples:

$(\epsilon, -), (b, -), (ab, -), (aba, -), (bab, -), (abab, +),$
 $(babab, -), (bbbab, -), (abbbab, +), (ababbab, -)$

This pack can be distributed into following components $(P_1, P_2, P_3, P_4, P_5)$ (one distribution from several possible):

access string		experiments				
P_1	$[\epsilon]$	$(\epsilon ab, -)$		$(\epsilon abab, +)$	$(\epsilon b, -)$	$(\epsilon \epsilon, -)$
P_2	$[b]$	$(b ab, -)$	$(b bbab, -)$	$(b abab, -)$		$(b \epsilon, -)$
P_3	$[ab]$	$(ab ab, +)$	$(ab bbab, +)$			$(ab \epsilon, -)$
P_4	$[aba]$		$(aba bbab, -)$		$(aba b, +)$	$(aba \epsilon, -)$
P_5	$[abab]$					$(abab \epsilon, +)$

The intuition behind the access strings is that each access string corresponds to one state of the automaton that accepts the unknown language $L^?$. Each access string defines on language $L^?$ a right-congruence class. Thanks to the condition (2) for existence of the distinguishing experiment are the congruence classes of access string unequal between each other, because there is always a witness proving their inequality. But this implies, that for every pair of different access strings the unknown automaton $M_{L^?}$ must have reached two different states. And therefore, every automaton accepting the language $L^?$ must have at least as many states as is number of access strings in the set S . Formally,

Theorem 4.1.2. *Let S be a set of access strings for an observation pack $P = (P_1, P_2, \dots, P_p)$ of language $L^?$. Let $M_{L^?}$ be the minimal (with respect to $|Q|$ – number of states in $M_{L^?}$) deterministic finite state automaton accepting $L^?$. Then*

$$|M_{L^?}| \geq |S|$$

Proof. Assume the automaton $M_{L^?}$ would reach the same state while running on two different access strings s_i and s_j . That is $\delta(q_0, s_i) = \delta(q_0, s_j)$. Because the automaton has reached the same state, for each string x holds that $\delta(q_0, s_i x) = \delta(q_0, s_j x)$ and therefore, $\forall e \in E_i \cap E_j : s_i e \in L^? \iff s_j e \in L^?$. But this is a contradiction with point (2) of the pack-component definition. Because for every pair of i, j the inequality $s_1 \neq s_2$ implies $\delta_{M_L}(q_0, s_1) \neq \delta_{M_L}(q_0, s_2)$, the automaton $M_{L^?}$ must have at least $|S|$ states. \square

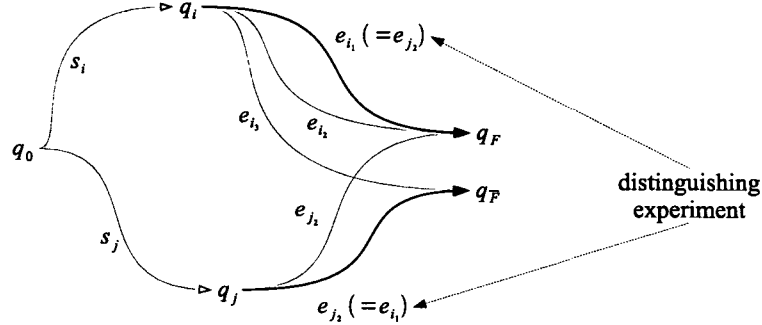


Figure 4.1: Distinguishing experiment leading to different states

Definition 4.1.3. String x “is like” an access string s_i if x and s_i behave the same way for all experiments in E_i , i.e.

$$\forall e \in E_i : s_i e \in L^? \iff x e \in L^?$$

Being unable to distinguish the “is like string” x from an access string on all its known experiments suggests that the automaton $M_{L^?}$ reaches on the string x the same state as on the access string.

Example 4.1.4. (continued Example 4.1.1) An access string and its “is like” string

		experiments		
access string	$[ab]$	$(ab ab, +)$	$(ab bbab, +)$	$(ab \epsilon, -)$
“is like” string	abb	$(abb ab, +)$	$(abb bbab, +)$	$(abb \epsilon, -)$

Let us clarify that the unknown automaton $M_{L^?}$ may not always reach the same state on an access string and its “is-like” string. The reason is that we just do not have currently enough examples to distinguish them.

In the next part, we will use a function that maps word from Σ^* to its “is like” access string. The following claim ensures that the function will be well defined, that is no word from Σ^* will map onto two different access strings.

Theorem 4.1.5. Let S be a set of access strings for an observation pack $P = (P_1, P_2, \dots, P_p)$ of language $L^?$. For each string $x \in \Sigma^*$, there is at most one access string s_i such that x is like s_i .

Proof. Assume x is like two different access strings s_i and s_j . The point (2) in the pack-components definition implies existence of experiment e such that $x e \in L^? \iff s_i e \in L^? \iff s_j e \notin L^? \iff x e \notin L^?$ and that is a contradiction. \square

4.2 Expanding of Pack/Adding Access Strings

We define partial function $\gamma : \Sigma^* \rightarrow S$ that maps string $w \in \Sigma^*$ to its “is like” access string with respect to the current observation pack $P = (P_1, P_2, \dots, P_p)$

$$\gamma : x \mapsto [s] \quad \text{such that } x \text{ is like } s$$

We say, string x “escapes” if $\gamma(x)$ is undefined. That is, there is no access string $s \in S$ such that x is like s .

Definition 4.2.1. An observation pack P is called *closed* if

1. $\gamma(\varepsilon)$ is defined
2. $\forall s \in S \quad \forall a \in \Sigma : \gamma(sa)$ is defined

Example 4.2.2. (continued Example 4.1.1) The observation from the example 4.1.1 is closed as demonstrated by Table 4.1

The most interesting property of the above defined function γ is that it may be undefined for some values. Discovery of such an undefined point means discovery of a new access string.

This fits then with the following theorem that expresses the core principle of the query learning algorithms for DFA by giving for the upper bound for the number of access strings. In the ground basis, all the learning algorithms just search for new access strings, what varies are only the strategies for discovery of new access strings. Moreover, as we show, when the upper bound for access strings is reached, the right-congruence classes characterised by the set of all access string define the minimal automaton accepting the target language $L^?$.

Theorem 4.2.3. [BDGW97] *Let P be an observation pack of language $L^?$. Let $M_{L^?} = (\Sigma^{M_{L^?}}, Q^{M_{L^?}}, \delta^{M_{L^?}}, q_0^{M_{L^?}}, F^{M_{L^?}})$ be the minimal automaton accepting language $L^?$. If the observation pack P has exactly $|M_{L^?}|$ components (or access strings) then*

- P is closed
- and automaton $A = (\Sigma, Q, \delta, q_0, F)$ is isomorphic with the minimal automaton $M_{L^?}$, where

$$\Sigma = \Sigma^{M_{L^?}}$$

$$Q = S$$

$$\delta : \delta([s], a) = \gamma(sa)$$

$$q_0 = \gamma(\varepsilon)$$

$$F = \{[s] \mid s \in S \wedge s \in L^?\}$$

Table 4.1: Closed observation pack example

(access) string	experiments				is like
ϵ b $[ab]$ $[aba]$ $[abab]$	$(\epsilon ab, -)$ $(b ab, -)$ $(ab ab, +)$	$(b bbab, -)$ $(ab bbab, +)$ $(aba bbab, -)$	$(\epsilon abab, +)$ $(b abab, -)$	$(\epsilon b, -)$ $(\epsilon b, -)$ $(ab b, -)$ $(aba b, -)$ $(abab b, -)$	$(\epsilon b, -)$ $(b b, -)$ $(ab b, -)$ $(aba b, -)$ $(abab b, -)$
$a a$ ϵb	$(a ab, +)$	$(a bbab, +)$		$(a b, -)$	$(a b, -)$
$b a$ $b b$	$(ba ab, -)$ $(bb ab, -)$	$(ba bbab, -)$ $(bb bbab, -)$	$(ba abab, -)$ $(bb abab, -)$	$(ba b, -)$ $(bb b, -)$	$(ba b, -)$ $(bb b, -)$
$ab a$ $ab b$	$(abb ab, +)$	$(abb bbab, +)$		$(abb b, -)$	$(abb b, -)$
$aba a$ $aba b$	$(abaa ab, -)$	$(abaa bbab, -)$	$(abaa abab, -)$	$(abaa b, -)$	$(abaa b, -)$
$abab a$ $abab b$	$(ababa ab, -)$ $(ababb ab, -)$	$(ababa bbab, -)$ $(ababb bbab, -)$	$(ababa abab, -)$ $(ababb abab, -)$	$(ababa b, -)$ $(ababb b, -)$	$(ababa b, -)$ $(ababb b, -)$

Proof. Assume that the observation pack P would not be closed. Then, there is some word $x \in \Sigma^*$ such that $\gamma(x)$ is undefined. Therefore, we can add the word x into the pack P as a new access string. But then we have $|S| > |M_{L^?}|$ which is a contradiction with Theorem 4.1.2.

The isomorphism of the automaton A with the minimal automaton $M_{L^?}$ implies from the bijection φ between S and $Q^{M_{L^?}}$ defined as:

$$\varphi : [s] \mapsto \delta^{M_{L^?}}(q_0^{M_{L^?}}, s)$$

φ is indeed a well defined function. That is, it maps each access string to at most one state of automaton $M_{L^?}$, because $M_{L^?}$ is deterministic.

The condition (2) for existence of the distinguishing experiment in the pack-component definition implies that for every pair of two access strings s_i and s_j if $s_i \neq s_j$ then $\varphi(s_i) \neq \varphi(s_j)$. Because $|S| = |M_{L^?}|$ and the sets are finite, the function φ is surjective. Therefore, the function φ is an isomorphism. \square

Example 4.2.4. Isomorphism of a closed observation pack to the minimal automaton

According to the construction in the Theorem 4.2.3, the automaton accepting the language $L_{\text{ex}}^?$ is:

$$Q = \{[\epsilon], [b], [ab], [aba], [abab]\},$$

$$F = \{[abab]\}$$

and the transition function δ defined as:

$$\begin{aligned} \delta([\epsilon], a) &= [ab] & \delta([aba], a) &= [b] \\ \delta([\epsilon], b) &= [b] & \delta([aba], b) &= [abab] \end{aligned}$$

$$\begin{aligned} \delta([b], a) &= [b] & \delta([abab], a) &= [b] \\ \delta([b], b) &= [b] & \delta([abab], b) &= [b] \end{aligned}$$

$$\begin{aligned} \delta([ab], a) &= [aba] \\ \delta([ab], b) &= [ab] \end{aligned}$$

And indeed, there is a isomorphism φ to the original automaton from Example 4.1.1 defined as:

$$\begin{aligned} q_0 &\mapsto [\epsilon] \\ q_1 &\mapsto [ab] \\ q_2 &\mapsto [aba] \\ q_+ &\mapsto [abab] \\ q_- &\mapsto [b] \end{aligned}$$

4.3 DFA Learning with Membership and Equivalence Queries

There are several version of the original algorithm [Ang87, BDGW97, KV94], essentially different only in the ways of processing and maintaining the received counterexamples and discovering of new access strings.

Typical strategy of the learning algorithm is to ask a sequence of membership queries, and continuously extend the set of known access strings using the received query answers. When the set of access string can not be expanded anymore, the learning algorithm considers the current hypothesis as a good candidate of automaton accepting the target language $L^?$. Therefore, it asks an equivalence query to confirm its hypothesis. If the answer is “yes” the algorithm succeeded, otherwise it uses the returned counterexample to discover a new access string.

Now, we describe one version of such DFA learning algorithm. First, it constructs an initial observation pack. Then the learning algorithm uses membership queries until that the pack is closed. That means, it determines the “is-like” access string for all strings of the form sa , where $s \in S$ and $a \in \Sigma$. In the next step, according to the construction in theorem 4.2.3 builds a hypothesis automaton A_{hyp} and it tests the automaton A_{hyp} using an equivalence query.

Let $x = x_0x_1 \dots x_{n-1}$ be the counterexample received from the teacher in case of negative answer. And let

$$([s_0], x_0x_1 \dots x_{n-1}) \vdash ([s_1], x_1 \dots x_{n-1}) \vdash \dots \vdash ([s_{n-1}], x_{n-1}) \vdash ([s_n], \epsilon)$$

be the run of the hypothesis automaton A_{hyp} on the counterexample x . The construction of the automaton A_{hyp} implies $s_n \in L^? \iff x \notin L^?$. Because there is always an access string s such that ϵ is like s , we may assume without loss of generality that $s_0 = \epsilon$.

Therefore, in sequence

$$\begin{aligned} x &= s_0(x_0x_1 \dots x_{n-1}) \\ &\quad s_1(x_1 \dots x_{n-1}) \\ &\quad \dots \\ &\quad s_{n-1}x_{n-1} \\ s_n\epsilon &= s_n \end{aligned}$$

must exist at least one k such that

$$s_k(x_k \dots x_{n-1}) \in L^? \iff s_{k+1}(x_{k+1} \dots x_{n-1}) \notin L^?$$

For finding that k we use binary search of at most $\lg n$ membership queries.

Then, if we look at $\gamma(s_k x_k)$, two cases are possible:

- $\gamma(s_k x_k)$ escapes. We have a new access string and can restart asking membership queries to close the observation pack.
- $\gamma(s_k x_k)$ is defined. Then, the transition $\delta(s_k, x_k) = s_{k+1}$ and construction of the automaton A_{hyp} imply that $s_k x_k$ is like s_{k+1} . Therefore, if we extend experiments of s_{k+1} with new experiment $x_{k+1} \dots x_{n-1}$ to

$$E_{k+1} = E_{k+1} \cup \{x_{k+1} \dots x_{n-1}\}$$

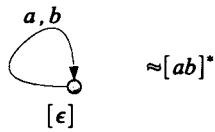
the string $s_k x_k$ will escape. So also in this case, we yield a new access string and can restart asking membership queries to close the observation pack.

Pseudo code of the DFA learning algorithm using membership and equivalence queries is summarised in Algorithm 6.

Example 4.3.1. (continued Example 4.1.1) Trace of the learning algorithm during learning of the language $L_{\text{ex}}^?$.

γ	experiments	is like
$[\epsilon]$	$(\epsilon \epsilon, -)$	
a	$(a \epsilon, -)$	$[\epsilon]$
b	$(b \epsilon, -)$	$[\epsilon]$

Asks an equivalence query $? : L_{A_{\text{hyp}}} \equiv L^?$ with automaton A_{hyp} :



Receives answer: no, counterexample: $abab$. Applies binary search:

$$\begin{array}{ccccccc} ([\epsilon], abab) & \vdash & ([\epsilon], bab) & \vdash & ([\epsilon], ab) & \vdash & ([\epsilon], b) & \vdash & ([\epsilon], \epsilon) \\ abab \in L & \iff & bab \notin L & & ab \notin L & & & & \end{array}$$

$\gamma(a)$ is defined. Therefore, add experiment bab and close the pack.

Algorithm 6 MQ&EQ learning algorithm for DFA

```

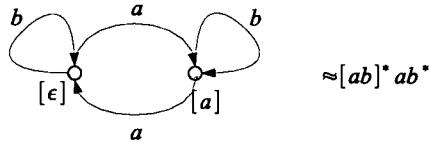
construct initial observation pack  $P$ 
while true:
    ask membership queries until the observation pack  $P$  is closed
    construct the hypothesis automaton  $A_{\text{hyp}}$ 

    ask the equivalence query  $? : L_{A_{\text{hyp}}} \equiv L?$ 
    if answer to query is "yes"
        return  $A_{\text{hyp}}$ 
    else
        # answer to query is "no"
        # with a counterexample  $x = x_0x_1 \dots x_{n-1}$ 
        use binary search and
        counterexample  $x_0x_1 \dots x_{n-1}$  received from teacher
        to find access string  $s_k$  such that
         $s_kx_kx_{k+1} \dots x_n \in L? \iff s_{k+1}x_{k+1} \dots x_n \notin L?$ 

        if  $\gamma(s_kx_k)$  is undefined
            add new access string  $s_kx_k$  to the pack  $P$ 
        else
            update  $E_{k+1} = E_{k+1} \cup \{x_{k+1} \dots x_{n-1}\}$ 
            add new access string  $s_kx_k$  to the pack  $P$ 
  
```

γ	experiments	is like
$[\epsilon]$	$(\epsilon \epsilon, -)$ $(\epsilon bab, -)$	
a	$(a \epsilon, -)$	$\{\epsilon\}$
b	$(b \epsilon, -)$	$[\epsilon]$
$[a]$	$(a \epsilon, -)$ $(a bab, +)$	
aa	$(aa \epsilon, -)$ $(aa bab, -)$	$[\epsilon]$
ab	$(ab \epsilon, -)$ $(ab bab, +)$	$[a]$

Asks an equivalence query $? : L_{A_{\text{hyp}}} \equiv L?$ with automaton A_{hyp} :



Receives answer: no, again counterexample: $abab$. Applies binary search:

$$([\epsilon], abab) \vdash ([a], bab) \vdash ([a], ab) \vdash ([\epsilon], b) \vdash ([\epsilon], \epsilon)$$

$$aab \in L \iff b \notin L$$

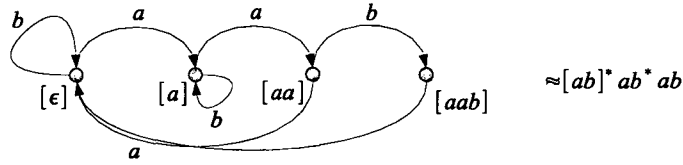
$\gamma(aa)$ is defined. Therefore, add experiment b and close the pack.

γ	experiments	is like
$[\epsilon]$	$(\epsilon \epsilon, -)$ $(\epsilon bab, -)$ $(\epsilon b, -)$	
b	$(b \epsilon, -)$	$[\epsilon]$
$[a]$	$(a \epsilon, -)$ $(a bab, +)$	
aa	$(aa \epsilon, -)$ $(aa bab, -)$	$\{\epsilon\}$
ab	$(ab \epsilon, -)$ $(ab bab, +)$	$[a]$
$[aa]$	$(aa \epsilon, -)$ $(aa bab, -)$ $(aa b, +)$	
aaa	$(aaa \epsilon, -)$ $(aaa bab, -)$ $(aaa b, -)$	$[\epsilon]$
aab	$(aab \epsilon, +)$ $(aab bab, -)$ $(aab b, -)$	new

While closing the pack a new access string is been discovered. Algorithm adds it and tries to close the pack again.

γ	experiments			is like
$[\epsilon]$	$(\epsilon \epsilon, -)$	$(\epsilon bab, -)$	$(\epsilon b, -)$	$[\epsilon]$
b	$(b \epsilon, -)$			
$[a]$	$(a \epsilon, -)$	$(a bab, +)$		$[a]$
ab	$(ab \epsilon, -)$	$(ab bab, +)$		
$[aa]$	$(aa \epsilon, -)$	$(aa bab, -)$	$(aa b, +)$	$[\epsilon]$
aaa	$(aaa \epsilon, -)$	$(aaa bab, -)$	$(aaa b, -)$	
aab	$(aab \epsilon, +)$	$(aab bab, -)$	$(aab b, -)$	
$[aab]$	$(aab \epsilon, +)$	$(aab bab, -)$	$(aab b, -)$	new
$aaba$	$(aaba \epsilon, -)$	$(aaba bab, -)$	$(aaba b, -)$	
$aabb$	$(aabb \epsilon, -)$	$(aabb bab, -)$	$(aabb b, -)$	

Asks an equivalence query ? : $L_{A_{\text{hyp}}} \equiv L^?$ with automaton A_{hyp} :



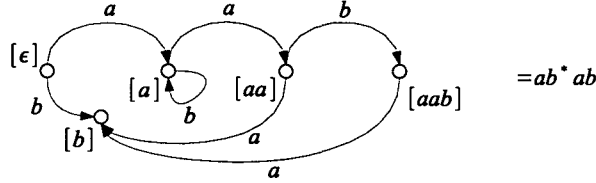
Receives answer: no, again counterexample: $baab$. Applies binary search:

$$\begin{array}{lcl}
 ([\epsilon], baab) & \vdash & ([\epsilon], aab) \vdash ([a], ab) \vdash ([aa], b) \vdash ([aab], \epsilon) \\
 baab \notin L & \iff & aab \in L \quad aab \in L
 \end{array}$$

$\gamma(b)$ is defined. Therefore, add experiment aab and close the pack.

γ	experiments				is like
$[\epsilon]$	$(\epsilon \epsilon, -)$	$(\epsilon bab, -)$	$(\epsilon b, -)$	$(\epsilon aab, +)$	$[\epsilon]$
b	$(b \epsilon, -)$				
$[a]$	$(a \epsilon, -)$	$(a bab, +)$			$[a]$
ab	$(ab \epsilon, -)$	$(ab bab, +)$			
$[b]$	$(b \epsilon, -)$			$(b aab, -)$	$[b]$
ba	$(ba \epsilon, -)$	$(ba bab, -)$	$(ba b, -)$	$(ba aab, -)$	
bb	$(b \epsilon, -)$	$(bb bab, -)$	$(bb b, -)$	$(bb aab, -)$	
$[aa]$	$(aa \epsilon, -)$	$(aa bab, -)$	$(aa b, +)$		$[b]$
aaa	$(aaa \epsilon, -)$	$(aaa bab, -)$	$(aaa b, -)$		
$[aab]$	$(aab \epsilon, +)$	$(aab bab, -)$	$(aab b, -)$		$[b]$
$aaba$	$(aaba \epsilon, -)$	$(aaba bab, -)$	$(aaba b, -)$	$(aaba aab, -)$	
$aabb$	$(aabb \epsilon, -)$	$(aabb bab, -)$	$(aabb b, -)$	$(aabb aab, -)$	

Asks an equivalence query ? : $L_{A_{\text{hyp}}} \equiv L^?$ with automaton A_{hyp} :



Receives answer: yes.

Note, that we did not store into the observation pack all of the counterexamples obtained from membership queries during the binary search.

The reason for this was that we wanted the observation pack to be smaller. And so, we have traded usage of more equivalence queries for reduction of the observation pack and therefore also for reduction of the number of membership queries.

But this implies that the hypothesis automaton may still incorrectly classify an already received counterexample. Therefore, there is a case possible, and happens also in the Example 4.3.1, that the teacher will answer more queries using the same counterexample.

4.3.1 Complexity

The Algorithm 6 runs in polynomial time. It asks at most $|M_L|$ equivalence queries as implied by the Theorem 4.2.3.

The membership queries in the algorithm are used in three contexts: for the binary search, for closing of the observation pack and for testing whether an access string is an accepting state.

To distinguish each access string from the other $|M_L| - 1$ access strings at most $|M_L| - 1$ experiments are required. Therefore, for closing of the observation pack requires the algorithm in each iteration at most $\mathcal{O}(|\Sigma||M_L|)$ membership queries. And so, in the whole run at most $\mathcal{O}(|\Sigma||M_L|^2)$ membership queries are used for closings of the observation packs.

Between the while iterations we have two possibilities:

- Either we remember all answers to the membership queries from the previous pack closings. In this case, for the whole run we use at most $\mathcal{O}(|\Sigma||M_L|^2)$ membership queries.
- Or we do not store the answers from the previous pack closings and during the current pack closing we re-query them again. In this case, for the whole run at most $\mathcal{O}(|\Sigma||M_L|^3)$ membership queries are used.

Let m be the size of the largest counterexample received from equivalence queries. Then for one binary search we use at most $\mathcal{O}(\log m)$ queries. Therefore, in the whole run at most $\mathcal{O}(|M_L| \log m)$ membership queries are used for the binary searches.

To summarise, there is a learning algorithm (Algorithm 6) that stores previous answers to membership queries and uses at most

$$\mathcal{O}(|\Sigma| |M_L|^2 + |M_L| \log m)$$

membership queries.

4.4 Lower Bounds for Number of Queries

In this section, we return back to the problem of relation between the number of required membership and equivalence queries left open in Section 2.3. But this time, we restrict to the hypothesis space of deterministic finite automata to obtain stronger results.

The first result stated here was obtained using the Vapnik-Chervonenkis dimension. It confirms that the VC-dimension is not the tightest lower bound, because later we will see a better bound.

Theorem 4.4.1. *[IS93] Every learning algorithm making $\mathcal{O}(n)$ equivalence queries must make $\Omega(|\Sigma| n \log n)$ membership queries.*

In the next theorem, we enlarge the lower bound to about n^2 . The interesting point about the theorem is that it shows a general strategy for construction of hard-to-learn learning problems.

Here, first a non-regular language is chosen. Then the language is turned into finite (and so regular) by restricting the maximum word length. This yields that any finite automaton accepting that language will have a lot of states. Note, that just having a lot of states does not imply this language is hard to learn, because the polynomial bounding the number of required queries may still have a relatively low degree.

But the learning can be made harder when the language is extended by union with one more set. For construction of this set we choose few random words as members of the regular language to be learned. We choose the candidates from an exponentially large superset with respect to the length of words it contains. Therefore, discovering those positive words requires searching the whole exponentially large set and so usage of many queries.

Theorem 4.4.2. *[BDGW97] For given $n \in \mathbb{N}$ and input alphabet Σ , $|\Sigma| \geq 2$, there exist constant $c > 0$ and a class \mathcal{C} of regular languages such that*

minimal automata accepting languages from this class have at least n states and for every learning algorithm successfully learning that class holds:

$$\#MQ(C) \geq c^2(|\Sigma| - 1)n^2 - cn\#EQ(C)$$

Proof. Let

$$m = \left\lfloor \log_{|\Sigma|} \left[\frac{(|\Sigma| - 1)(n - 1)}{2} + 1 \right] - 1 \right\rfloor$$

Let $a \in \Sigma$ be a chosen fixed input symbol. Let L_{CF} be a copy-language of the form

$$L_{CF} = \{xax \mid x \in \Sigma^m\}$$

It is a finite restriction of a context free grammar. Let us define

$$q_x := \delta(q_0, x)$$

$$p_x := \delta(q_0, xa)$$

as the states reached after processing the word x , respectively xa , by some minimal automaton accepting the language L_{CF} . Let Q_x, P_x be sets of all such possible states:

$$Q_x = \{q_x \mid q_x = \delta(q_0, x) \quad \forall x \in \Sigma^m\}$$

$$P_x = \{p_x \mid p_x = \delta(q_0, xa) \quad \forall x \in \Sigma^m\}$$

It is easy to verify that all states involved in the run of the minimal automaton on words of form xax must cumulatively store all the previously seen symbols.

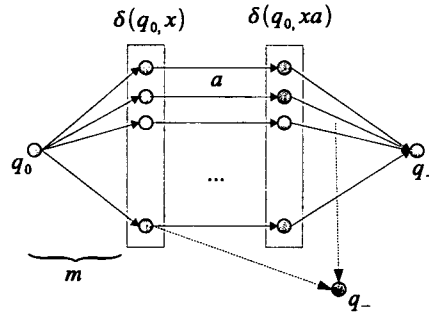


Figure 4.2: Automaton accepting the finite restriction L_{CF}

Therefore, all those states are different between each other and so the sets Q_x and P_x have cardinality

$$|Q_x| = |\Sigma|^m$$

$$|P_x| = |\Sigma|^m$$

and the minimal automaton accepting the language L_{CF} has at least

$$2 \frac{|\Sigma|^{m+1} - 1}{|\Sigma| - 1} + 1$$

states. Thanks to the choice m , this is larger or equal than the given n .

Now, we increase the query complexity of the regular language by adding new words into the language. For all triples in

$$Q_x \times \Sigma - \{a\} \times P_x$$

we to the original minimal automaton either add a transition defined by the triple or leave it undefined. Let S be the set of word we add this way into the language. Together, there are

$$(|\Sigma| - 1)|\Sigma|^{2m} \quad (4.1)$$

possibilities for adding a transition or leaving it undefined. Because we want the automaton accepting the language to be deterministic, at most $(|\Sigma| - 1)|\Sigma|^m$ transitions can be added from all the available possibilities. Therefore,

$$|S| \leq (|\Sigma| - 1)|\Sigma|^m \quad (4.2)$$

The final language constructed with the above described method has the form of

$$L^? = \{xax \mid x \in \Sigma^m\} \cup S$$

where S is some subset of

$$L_{xy} = \{xby \mid x, y \in \Sigma^m, b \in \Sigma - \{a\}\}$$

Now, let us analyse the number of queries required to learn the language $L^?$ by any learning algorithm. We assume that the teacher behaves adversary that is, she uses the power of the language $L^?$ and returns to posed queries little informative answers.

To achieve this, the adversary teacher maintains set of words X_{POS} that have been returned as positive query answers and for answering of the queries she plays then the “moving target” game. That is, she tries to delay the situation, where only the last hypothesis remains consistent with all the previous query answers.

These following situations are possible during answering of a query:

query	situation	answer	description
$? : w \in L?$	$w \in L_{CF} \cup X_{POS}$	yes	
	else	no	
$? : L_A \equiv L?$	exists some $xcy \in X_{POS}$ and $xcy \notin L_A$	no, xcy	if some of the previously returned positive examples in X_{POS} is not still accepted
	exist some $xcy \in X_{POS}$ and $xcz \in L_A$ such that $y \neq z$	no, xcz	if the previously returned positive examples in X_{POS} imply that some word should not be accepted anymore, but the automaton L_A still accepts it.
	exists some $xdy \in X_{POS}$ and $\forall z \ xdz \notin X_{POS}$	no, xdy	if L_A accepts consistently with all the previously answered positive examples in X_{POS} and also consistently with all the negative words implied from X_{POS}
	else	yes	

From a positive answer to a membership query with particular transition xcy , a learning algorithm can not yield more new informations about other possible transitions from $Q_x \times \Sigma - \{a, c\} \times P_x$. Similarly, from a negative answer to a membership query it can not yield more new information than just with plain trying of all the possibilities from $Q_x \times \Sigma - \{a, c\} \times P_x$. From a negative answer to an equivalence queries the algorithm can yield information about 0, 1 or Σ^m transitions.

The learning algorithm must keep asking queries until the set X_{POS} reaches $(\Sigma - 1)\Sigma^m$ elements which means that the presence or absence of all $(\Sigma - 1)\Sigma^{2m}$ transitions has been resolved. Therefore, combining with Equations 4.1 and 4.2 we have the following bound:

$$(\Sigma - 1)\Sigma^{2m} - \Sigma^m \#EQ - \#MQ \leq 0$$

With c approximately $\frac{1}{4}$ this can be adapted to the form presented in theorem statement. \square

Rewriting the above theorem using asymptotic notation, we have

Corollary 4.4.3. *[BDGW97] Every learning algorithm making $o(|\Sigma|n)$ equivalence queries must make $\Omega(|\Sigma|n^2)$ membership queries.*

Part II

HTML Wrapping

Chapter 5

Wrapper Induction and Learning

5.1 Wrapping Approaches

Commonly adopted strategies for accessing data located on Web pages are technologies known as Web information extraction or HTML wrapping. They turn interesting data in Web pages into formats suitable for further machine processing such as XML [Lix05], data models of integrated applications [Tec05b] or relational databases [LSPS05].

Over the time various approaches of building the HTML wrappers have been researched. Chronologically, the following mainstreams can be recognised:

- hand coded wrapping programs expressed in Perl, special purpose languages or as finite state machines [HGMC⁺97, HFAN98]
- inductive learning of string based automata [CMM01, KWD97]
- semi-automatic visual systems with specially designed wrapping languages [BFG01, LPH00]
- machine learning techniques:
 - support vector machines [MK05]
 - hidden Markov models [LSPS05]
 - conditional Markov models [MJ05]

Many parties see the strategical value of information extraction solutions. This value is even magnified, if the information extraction can be additionally interconnected with data mining or knowledge acquisition techniques.

Therefore, nowadays one can see a lot of research projects and commercial ventures battling in the field.

Reviewing the current successful commercial systems, it is possible to recognise two mainstream categories. The first category are wrapping systems based on various machine learning techniques with lower roll-out and maintenance costs, but reaching worse results and usually being specialised to particular domains. Examples of them are:

- IBM WebFountain¹ - aims to transform data available on the Web into business trends, for example collects user feedback about various products in online discussion forums.
- Froogle², the former Netbot Jango³ - collect information about items sold in Web shops.
- KnowItAll⁴ - accumulates large collections of facts from the Web.

The second category of commercial systems are tools such as [Tec05b, Lix05, Tec05a] which allow to build more complicated wrapping solutions. But here a human wrapper designer is required to build and maintain the wrappers. Therefore, it increases costs for acquiring of the information.

Most of the current approaches that are subject of our interest have the Web page represented in form of a DOM tree [Rec98, Rec03] and on these DOM trees the extraction process is executed. The DOM trees are constructed using HTML parsing libraries such as [ea04, Mic04] or leveraging parsers in modern Web browsers such as Mozilla [Fou05] and Internet Explorer [Cor05]. Example of a Web page and its corresponding DOM tree are depicted in the Figure 5.1.

Having a DOM tree, the HTML wrapping task can be viewed as identification of the relevant parts, for example subtrees in the input DOM tree. From applications point of view, the interested data chunks to be extracted usually are:

- sets of tree nodes in the given input DOM tree
- substrings from the text content nodes
- values from tree node attributes

¹<http://www.almaden.ibm.com/webfountain/>

²<http://froogle.google.com>

³<http://www.jango.com>

⁴<http://www.cs.washington.edu/research/knowitall/>

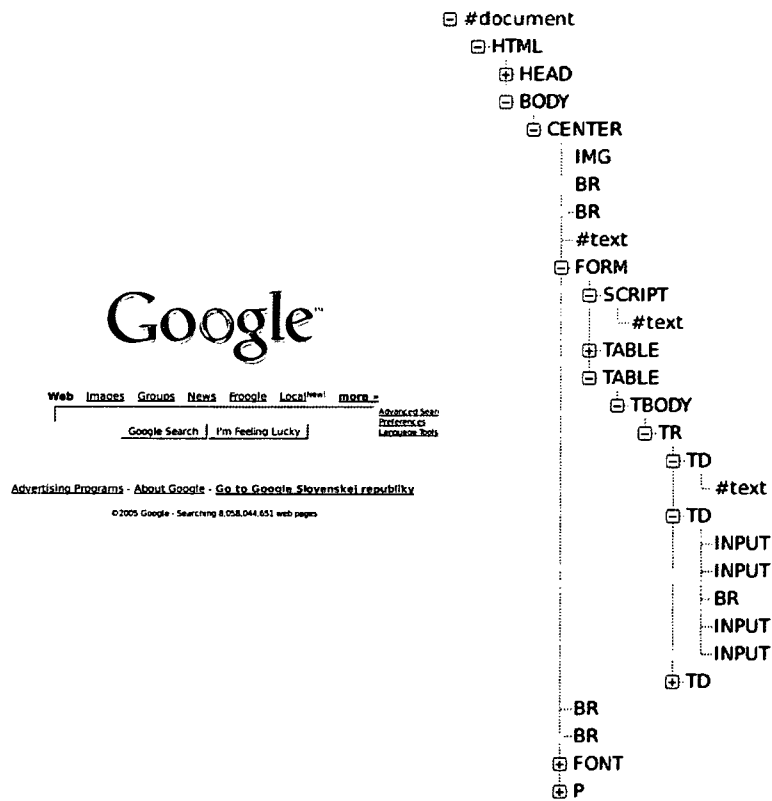


Figure 5.1: Web page and its DOM tree

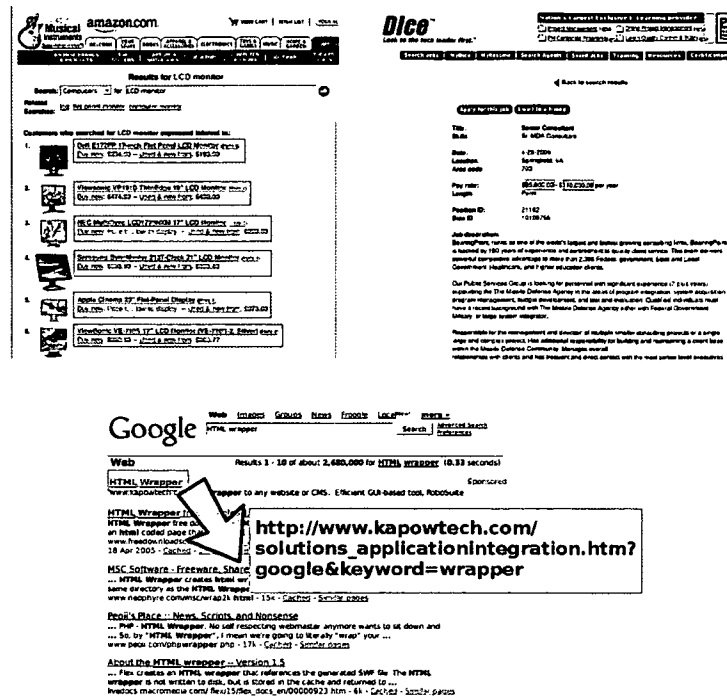


Figure 5.2: Different types of instances extracted from Web pages

Further, we will call the interesting data chunks to be extracted as **instances**. Examples of the various instance types are in Figure 5.2 highlighted with green background.

Operating on the DOM-tree, various techniques are used to express how to extract the interested instances from the given Web page. We have identified the following approaches:

- Datalog-like programs evaluated over tree domains [Lix05, HL98]
- finite (tree) automata [CLN04, GMTT05]
- XPath/XQuery node selection queries [Goe05]

In the following part, we will focus on interactive wrapper generators operating on DOM trees that create wrappers from a visual interaction with a human wrapper designer. During the visual interaction the user is asked to provide an example input HTML document and then to mark via mouse clicks positive and negative examples inside of the rendered input HTML page. Such an interaction is technically equivalent to the marking of nodes in the DOM tree.

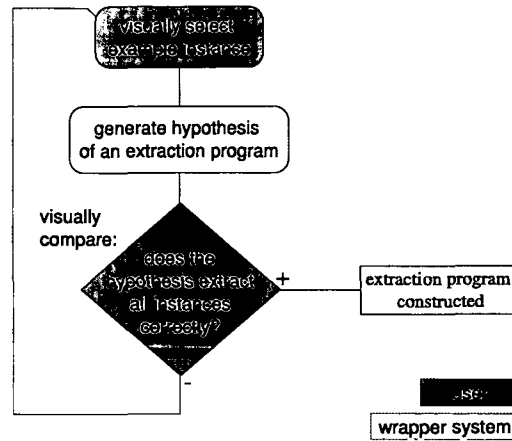


Figure 5.3: Wrapper creation process

5.2 Interaction with the Wrapping System

The goal of an interaction between a user and a wrapper-building system is to help the system to discover an extraction program that correctly identifies all the desired instances. The process of the interaction is outlined in Figure 5.3 as a loop of the following steps:

1. Select an example HTML document.
2. On this HTML document mark missing instances not yet recognised by the system or drop instances that were incorrectly identified by the system.
3. After the system has correctly matched all of the intended instances inside of the current HTML document, decide whether to continue with training/testing on another HTML document.

To formalise this interaction we will use in the next part the terminology of game theory, because it allows us to capture both the adversary and cooperative forms of behaviour between a wrapper system WS and a user U .

Let T_{in} be a set of all input trees and let $T_{out} = 2^{\text{subtrees}(T_{in})}$ be the set of all possible subsets of tree fragments from T_{in} . For example, in case of the Web pages, T_{in} is the set of all well-formed DOM-trees and T_{out} are all possible subsets of DOM-tree fragments.

Algorithm 7 Protocol of the wrapper-simulating game

-
1. $U \rightarrow WS, C$: user chooses tree $t_{in} \in T_{in}$
 # for example an HTML document or
 # fragment of an HTML document
 2. $U \rightarrow C$: user announces the set of intended instances
 $P(t_{in}) \in T_{out}$ from the document t to the committee C
 3. $WS \rightarrow U, C$: WS responds with an estimated set
 of matched instances $t_{out} \in T_{out}$
 4. C decides: **if** $t_{out} = P(t_{in})$ (precisely $\rho(t_{out}, P(t_{in})) = 0$)
 # user has lost this round, wrapper system
 # matched all intended instances
 if user resigned
 end successful wrapper learning process
 else
 goto step 1
 else
 # wrapper system did not match
 # the correct set of instances,
 # user provides more examples
 5. $U \rightarrow WS, C$: user chooses set of instances $t_{exam} \in T_{out}$
 such that $\rho(t_{exam}, P(t_{in})) < \rho(t_{out}, P(t_{in}))$
 if t_{exam} does not exist
 # wrapper system has lost
 end failed wrapper learning process
 else
 goto step 3
-

A wrapper program is an algorithm that computes a mapping $P : T_{in} \rightarrow T_{out}$. The game space (or also the hypotheses space of extraction programs) is set of all such possible mappings. Of course, there may be additional restrictions about the valid form of this mapping, such as being a valid tree automaton or XPath expression.

We define the wrapper building process as a game being played according to the protocol outlined in Algorithm 7 between the user U and the wrapper generation system WS . For the WS player to win the game, it must be able to successfully infer the extraction program P intended by the user U . The winning region is the set of all extraction programs matching exactly the intended instances. Let us note that the winning region is not constant for all games.

From wrapper system's point of view, the user U is just a black-box that knows the winning region. Because the wrapper system has merely the possibility of observing the moves (1)-(3) played by the user U during the game, we need an additional mechanism that prohibits cheating in case of an adversary behaving user. An example of such cheating would be changing of the winning region (the desired set of instances to be extracted) once the game playing has been started. Therefore, we add into our model one more party called *committee* C , that performs the necessary guarding checks guarantying fairness of the game.

Note, that for real-world implementations the committee player is again played by the user that interacts with the wrapper generation system. We may assume that it never triggers a warning about failure of the guarding checks. This is based on an assumption that the user wants to do some meaningful work with the system and so, she will not cheat intentionally.

The metrics function ρ defined on the set T_{out} forces the user in each game round to correct the set instance hypotheses given by the wrapper system in such a way that the corrected set of instances is closer to the right answer than the original estimation of the wrapper system (Step 5). Example of such a metrics ρ may be a number of elements in a symmetric difference between these two sets.

Once the wrapper system has identified the correct extraction program, it returns the correctly matched set of instances on all input trees. Therefore, a (learning) algorithm A is a winning strategy for the wrapper system WS for the class of the wrapper-simulating games, if in each game played between the user U and the wrapper system WS is after a finite number of rounds each following round won by the WS with a single move. That is, the WS will immediately mark all the intended instances for any given input tree.

Protocol Implications

The wrapper systems are real-world applications being practically implemented and used. Therefore, it is natural to require that for any extraction program, kept in the mind of the user, the wrapper system is able to discover and construct it. That is, expressed for the game model, we insist on existence of a winning strategy for the wrapper system WS .

Lemma 5.2.1. *There exists always a winning strategy for wrapper system WS , assuming there is a computable enumeration of extraction program hypotheses.*

Proof. One of the winning strategies is for example the enumeration of hypotheses, with skipping of hypotheses that are not compatible with the set

of positive/negative examples instances in the current DOM tree chosen by the user U . \square

Moves that are played in a single round, that is inside of one DOM-tree chosen by the user U , do not add to the wrapper system WS a real computational advantage. As shown by the next Lemma, they put the WS into a weaker position as if the user U would directly label all nodes of the current tree which is equivalent to a passive form of learning.

Lemma 5.2.2. *If there exists a winning strategy for the wrapper-simulating games, then there exists also a winning strategy for the simplified wrapper-simulating games. That is for such games, where in each round (on each chosen input DOM tree) after the first move of the wrapper system WS the user U directly announces all correct (intended) instances to the WS .*

Proof. Let S be a winning strategy full wrapper-simulating games. Then winning strategy S' for the simplified wrappers-simulating games can be constructed as follows:

Let t be an input tree chosen by the user U in a round of a simplified wrapper-simulating game. S' asks the strategy S to mark instances. Let $H_1(t)$ be the set of instances that will be answered by the strategy S . The strategy S' plays a move with $H_1(t)$ as an answer.

Let $P(t)$ be set of correct instances that are then announced by the user U . If $P(t)$ is equal to $H_1(t)$, the next round between U and S' is played. Otherwise if $P(t)$ is not equal to $H_1(t)$, the strategy S' plays the game against the strategy S until it replies with $P(t)$. This must occur, because S is a winning strategy.

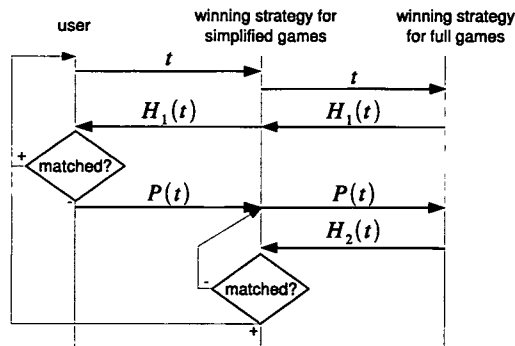


Figure 5.4: Reduction of simplified wrapper-simulating games to full wrapper-simulating games

\square

The situation with labelling of all nodes at once shows that the sequential interaction, known from active learning methods, is not itself the key feature that leads to the stronger learners when compared to the passive form of learning. As we will see later, it is the sequential interaction combined together with more universally quantified queries (answers) in Step 5 of Algorithm 7 that bring the positive learnability results. Formally said, the user will answer in the Step 5 with an example instance t_{exam} fulfilling

$$\rho(t_{\text{exam}}, P(t_{\text{in}})) < \rho(T_{\text{query}}, P(t_{\text{in}}))$$

where T_{query} is a set of instances with a similar structure as the instance t_{out} estimated previously in the Step 3.

Chapter 6

Query Based Learning of XPath

6.1 XPath Definitions and Background

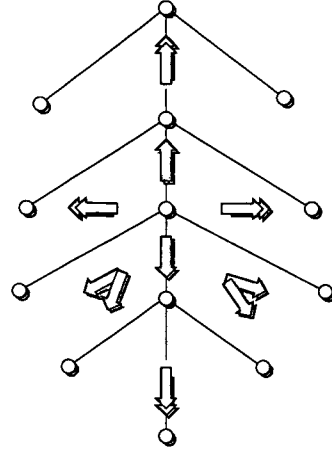
XPath [Rec99] is a language for navigation in XML documents and their physical representations such as DOM trees. Nowadays, it is part of many XML related applications and standards such as XSLT, XQuery, XLink, etc.

Therefore, we will study in this chapter active forms of learnability for various fragments of the CoreXPath [GKP02]. CoreXPath is the full XPath language stripped from functional symbols and predicates. It consists of finite length expressions generated by the following grammar:

$$\begin{array}{lcl} p & \rightarrow & a \\ & & * \\ & & p/\text{axe}::p \\ & & p[f] \\ f & \rightarrow & p \\ & & p|p \end{array}$$

Here, the symbol $a \in \Sigma$ represents a node label. The symbol $*$ is a wild card matching any node label. The terminal symbol $/\text{axe}::$ is one of the following tree navigation steps:

- child (/), parent
- descendant, descendant-or-self (//)
- ancestor, ancestor-or-self
- following, following-sibling
- preceding, preceding-sibling
- self (.)



There are several commonly usage shortcuts: $\cdot \equiv /self::*$, $/a \equiv /child::a$ and $//a \equiv /descendant-or-self::node()/a$. The subexpression f is called filter condition. We define the size of an XPath expression $|p|$ as the number of its axes.

We define the XML document as an ordered and unranked tree with nodes labelled from an infinite alphabet Σ . By T_Σ we denote the set of all XML documents over the alphabet Σ . For a tree $t \in T_\Sigma$, the size of the tree, denoted as $|t|$, is the number of its nodes.

Let $p(t)$ be the set of nodes navigated by the expression p on the tree t . The semantics of $p(t)$ is defined inductively on the structure of p according to the Table 6.1.

Because we will deal in this chapter with learning on trees, we additionally denote the target XPath expression that should be learned as $p^?$. For the purpose of sample labelling needed by learning algorithms, we add a \star -mark to nodes that should be the result of the XPath navigation on some example instance. Let $nodes_\star(t)$ be the set of all \star -marked nodes in the tree t .

We say that the expression p extracts a tree $t \in T_\Sigma^*$ if $p(t) = nodes_\star(t)$. Informally said, the expression p navigates exactly to the \star -marked nodes.

6.2 Boundaries of Interactive Wrapper Induction

Now, let us summarise the known results of the query based learning that apply also to the query based learning of XPath fragments.

Corollary 6.2.1. *XPath or its fragments with expressions represented using deterministic tree automata are polynomially learnable from characteristic sets and with MQ&EQ queries.*

Table 6.1: Semantics of CoreXPath expressions

$q_1/\text{self}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), \text{label}(u) = a, w \in q_2(u)\}$
$q_1/\text{child}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{children}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{parent}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{parent}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{self}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), w \in q_2(u)\}$
$q_1/\text{child}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{children}(u), w \in q_2(v)\}$
$q_1/\text{parent}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{parent}(u), w \in q_2(v)\}$
$q_1/\text{descendant}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{descendants}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{descendant-or-self}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{descendants}(u) \cup \{u\}, \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{descendant}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{descendants}(u), w \in q_2(v)\}$
$q_1/\text{descendant-or-self}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{descendants}(u) \cup \{u\}, w \in q_2(v)\}$
$q_1/\text{ancestor}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{ancestors}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{ancestor-or-self}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{ancestors}(u) \cup \{u\}, \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{ancestor}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{ancestors}(u), w \in q_2(v)\}$
$q_1/\text{ancestor-or-self}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{ancestors}(u) \cup \{u\}, w \in q_2(v)\}$
$q_1/\text{following-sibling}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{right-siblings}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{following}::a/q_2(x)$	$=$	$\{w \mid t \in q_1(x), u \in \text{right-siblings}(t), v \in \text{descendants}(u) \cup \{u\}, \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{following-sibling}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{right-siblings}(u), w \in q_2(v)\}$
$q_1/\text{following}::*/q_2(x)$	$=$	$\{w \mid t \in q_1(x), u \in \text{right-siblings}(t), v \in \text{descendants}(u) \cup \{u\}, w \in q_2(v)\}$
$q_1/\text{preceding-sibling}::a/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{left-siblings}(u), \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{preceding}::a/q_2(x)$	$=$	$\{w \mid t \in q_1(x), u \in \text{left-siblings}(t), v \in \text{descendants}(u) \cup \{u\}, \text{label}(v) = a, w \in q_2(v)\}$
$q_1/\text{preceding-sibling}::*/q_2(x)$	$=$	$\{w \mid u \in q_1(x), v \in \text{left-siblings}(u), w \in q_2(v)\}$
$q_1/\text{preceding}::*/q_2(x)$	$=$	$\{w \mid t \in q_1(x), u \in \text{left-siblings}(t), v \in \text{descendants}(u) \cup \{u\}, w \in q_2(v)\}$
$q_1[q_2](x)$	$=$	$\{w \mid w \in q_1(x), q_2(x) \neq \emptyset\}$
$(q_1 \mid q_2)(x)$	$=$	$\{w \mid w \in q_1(x) \cup q_2(x)\}$

Proof. Each XPath expression from the CoreXPath fragment can be rewritten into an equivalent deterministic bottom-up tree automaton. Therefore, using the results from [CLN04] and [Sak90], they are learnable from characteristic sets and learnable using the *MQ&EQ* queries. \square

Let us note that the result from the previous corollary is not a contradiction to the other polynomial non-learnability results in this chapter. This is due the transformation, which may create an exponential enlargement of the original representation class.

Corollary 6.2.2. *All concept classes with representations from NFA (or higher in Chomsky's hierarchy) are not polynomially learnable with example based queries.*

Proof. Implied from Theorem 1.4.2, Corollary 1.4.3 and 1.6.7. \square

As implied by the previous corollaries, the full XPath is not polynomially learnable with example based queries. Luckily, there are many types of XPath (sub-)fragments with good properties. Some of the fragments have only exponentially enlarging translation to deterministic tree automata and are not good candidates for studying of their polynomial learnability. But there are also fragments (for example without transitive axes such as ancestor) that have a translation to the deterministic automata with only a polynomially bounded enlargement. These fragments may be good candidates for being polynomially learnable and we will analyse their learnability.

The Figure 6.1 shows the boundary of learnability between deterministic and nondeterministic tree automata and how the XPath and its fragments fit into it.

6.3 Equivalence Queries

Applied to DOM trees, an equivalence query is a question of the form $\forall t \in T_\Sigma : p(t) = p'(t)$. Answers are either "yes" or "no" with a counterexample of an instance witnessing the inequality.

Equivalence queries have, in context of the HTML wrapper induction, a very natural visualisation. To answer them a human wrapper designer just marks the missing or unwanted example instances inside of the currently rendered HTML page. Optionally, she tests the hypothesis on other Web pages. Example of such visualised interaction is in Figure 6.2.

Angluin [Ang90] presents a sufficient condition for non-learnability using only equivalence queries, called approximate fingerprints. Applied to XPath fragments, the definition can be reformulated as:

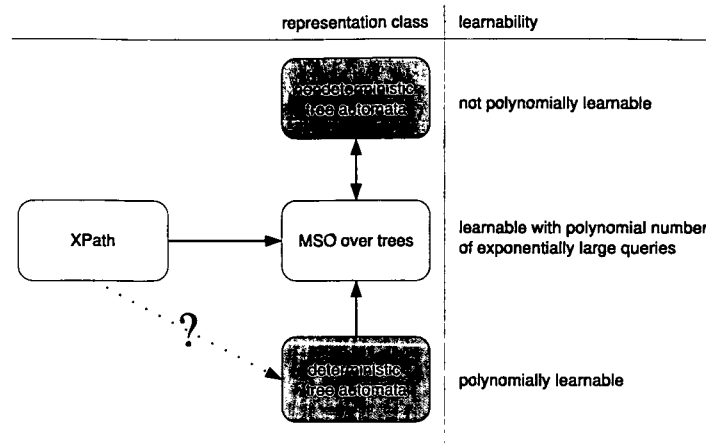


Figure 6.1: XPath situated on boundaries of polynomial learnability and non-learnability

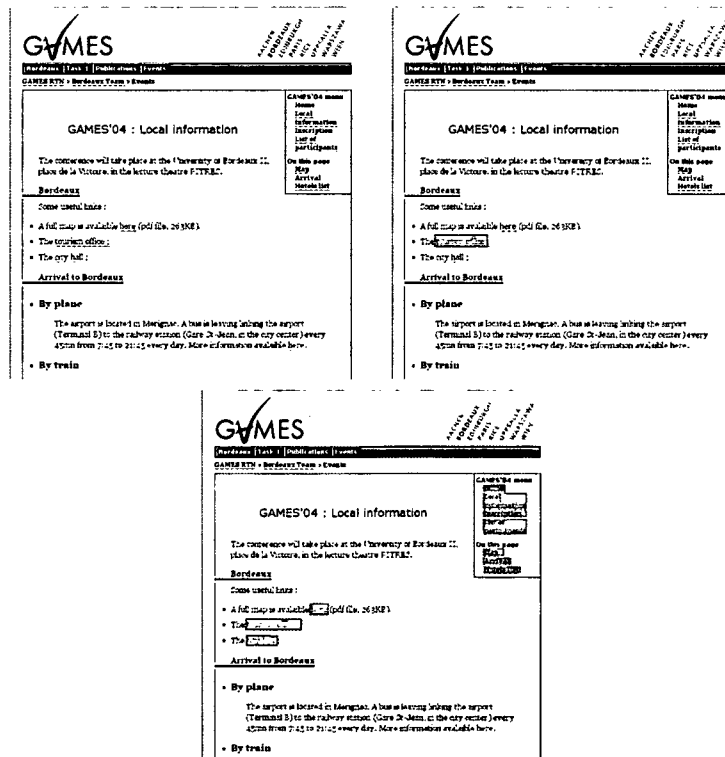


Figure 6.2: Visualisation of equivalence queries

Definition 6.3.1. An XPath fragment XP has approximate fingerprints if there exist nondecreasing polynomials p_1 and p_2 such that for every positive nondecreasing polynomial q there exists a sequence of classes T_1, T_2, \dots containing the XP fragment expressions with size bounded by p_1 . And additionally, for infinitely many n :

- class T_n contains at least two XPath expressions of size at most $p_2(n)$;
- for every XPath expression $r \in XP$, $|r| \leq q(n)$ there exists a tree $t \in T_\Sigma$ with at most $p_2(n)$ nodes such that number of XPath expression in T_n that navigate on tree t to the same set of nodes as r does, is less than $\frac{|T_n|}{q(n)}$.

Or in a concise symbolic form:

$$\begin{aligned} & \exists p_1, p_2 \forall q \\ & \exists \{T_i\}_{i \geq 0}, T_i \subseteq XP, \forall p \in T_i : |p| < p_1(i) \\ & \text{for infinitely many } n \\ & |T_n^{\leq p_2(n)}| \geq 2 \wedge \\ & \forall r \in XP, |r| \leq q(n) \exists t \in T_\Sigma, |t| \leq p_2(n) \\ & |\{p \in T_n \mid p(t) = r(t)\}| < \frac{|T_n|}{q(n)} \end{aligned}$$

We use this definition of fingerprints and apply techniques from [Ang90] to show non-learnability of certain XPath fragments using only equivalence queries.

Definition 6.3.2. By $XP(/, //, [\leq^{m^\alpha}, |, *])$ we denote an XPath fragment which has union ($|$), wild card node matching ($*$) and only child and descendant-or-self axes. Additionally, there exists a single constant $0 < \alpha < 1$, such that for each expression p from this fragment is the number of all location paths from all filter conditions bounded by m^α , where m is the number of the axes on the main (top-level) location path of p .

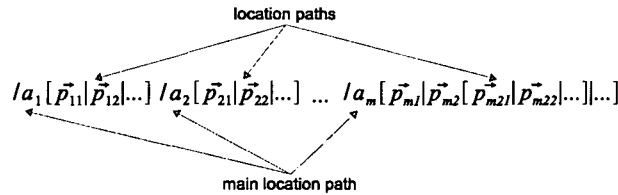


Figure 6.3: Location paths in an XPath expression

Theorem 6.3.3. The XPath fragment $XP(/, //, [\leq^{m^\alpha}, |, *])$ has the approximate fingerprints.

To prove the theorem we proceed as follows. First we define a sequence of concept classes T_1, T_2, T_3, \dots . Then for any learning algorithm A with running time bound by polynomial q , there exists for a sufficiently large s a concept class T_s which can be used by an adversary teacher for answering of the asked equivalence queries asked by the learning algorithm.

Next, for each concept class T_n we define two sets A_n, B_n and we show in a lemma that for all sufficiently large n each XPath expression p which is subject of an equivalence query either does not extract all trees in A_n or extracts a tree from B_n .

Thanks to this property an adversary teacher is able to provide such an answer to each equivalence query which rules out only exponentially small fragment of all concepts. And this implies that no polynomial learning algorithm is able to find the target concept in the exponentially large space of all concepts.

Let $\Sigma, |\Sigma| \geq 2$ be some finite subset of the (infinite) input alphabet of XML documents. Moreover, let $k \in \mathbb{N}$ be a sufficiently large number such that for $0 < \alpha < 1$:

$$k > \frac{\lg |\Sigma| + 1 - \alpha}{\alpha} \quad (6.1)$$

$$k > \frac{1 - \alpha}{\alpha} \quad (6.2)$$

$$k > \frac{1 + 2\alpha}{1 - \alpha} \quad (6.3)$$

Let T_1, T_2, T_3, \dots be a sequence of concept classes defined as follows. For $i = 1 \dots n^k$ let L_{in} be the set of all single-branched trees of size $2n^k + 2$ with the first and the middle symbol $\#$, such that the $(i+1)$ -th and $(n^k + i + 2)$ -th node have the same label. That is

$$L_{in} = \{\#x_1ax_2\#y_1ay_2 \mid x_1, y_1 \in \Sigma^{i-1}, x_2, y_2 \in \Sigma^{n^k-i}, a \in \Sigma\}$$

A concept is n concatenations of L_{in} languages plus the appended symbol $\#^*$. For each tree t from the concept only the last node ($\#^*$) is result of the XPath navigation on t . The concept class T_n is a set of all possible concatenations. That is

$$T_n = \{L_{i_1,n} \cdot L_{i_2,n} \cdot L_{i_3,n} \cdot \dots \cdot L_{i_n,n} \#^* \mid 1 \leq i_1, \dots, i_n \leq n^k\}$$

Clearly, $|T_n| \geq 2$ for $n \geq 2$.

Let us take an XPath expression from the fragment $XP(/, //, [\leq m^\alpha, |, *])$ of the following form:

$$\begin{aligned}
 & / \# \underbrace{ / * \dots / * }_{i_1} [\underbrace{ a / * \dots / * / a }_{n^k + 2} \mid \underbrace{ b / * \dots / * / b }_{n^k + 2} \mid \dots] \underbrace{ / * \dots / \# \dots / * }_{2n^k - i_1 + 1} \\
 & \dots \\
 & / \# \underbrace{ / * \dots / * }_{i_n} [\underbrace{ a / * \dots / * / a }_{n^k + 2} \mid \underbrace{ b / * \dots / * / b }_{n^k + 2} \mid \dots] \underbrace{ / * \dots / \# \dots / * }_{2n^k - i_n + 1} \\
 & / \#
 \end{aligned}$$

The language generated by this XPath expression is exactly $L_{i_1, n} \cdot L_{i_2, n} \cdot L_{i_3, n} \dots L_{i_n, n} \#^*$. For each tree t from this language the XPath expression navigates only to its last node. The expression has $p_1(n) = [2 + 2n^k + |\Sigma|(n^k + 2)]n + 1$ axes. Because k is constant, the sequence of concept classes T_1, T_2, \dots, T_n is bounded by the polynomial $p_1(n)$. The number of axes on the main location path is $m = (2n^k + 2)n + 1$ and the number of all location paths in all filter conditions is $|\Sigma|n$. Due to the choice of k fulfilling (6.1) and (6.2) it holds

$$|\Sigma|n \stackrel{(6.1)}{<} 2^{(k+1)\alpha-1} \stackrel{(6.2)}{\leq} [n^{k+1}]^\alpha \leq [2n^{k+1} + 2n + 1]^\alpha \leq m^\alpha$$

and therefore, the expression belongs to the fragment $XP(/, //, [\leq m^\alpha, |, *])$.

Let A_n be the intersection of all concepts in T_n

$$A_n = \bigcap_{L \in T_n} L = \{ \#x_1 \#x_1 \#x_2 \#x_2 \dots \#x_n \#x_n \#^* \mid x_i \in \Sigma^{n^k} \}$$

and let B_n be the set of n pairs of trees from Σ^{n^k} such that at least $\frac{n}{2}$ pairs have at least $\frac{n^k}{4}$ unequally labelled nodes and arbitrary subset of nodes which includes the last $\#^*$ may be result of the XPath navigation.

$$B_n = \{ \#x_1 \#y_1 \#x_2 \#y_2 \dots \#x_n \#y_n \#^* \mid x_i, y_i \in \Sigma^{n^k}, d(x_i, y_i) > \frac{n^k}{4} \text{ for at least } \frac{n}{2} \text{ values of } i \}$$

The size of any tree in the sets A_n and B_n or of any tree in any concept from T_n is bounded by polynomial $p_2(n) = (n^k + 1)2n + 1$.

In the following part we prove a lemma, which will be later used for completion of the proof of Theorem 6.3.3.

Lemma 6.3.4. *Let q be an arbitrary nondecreasing polynomial. Let p be an XPath expression from the $XP(/, //, []^{m^a}, |, *)$ fragment such that $|p| \leq q(n)$. For all sufficiently large n , if p extracts all trees from A_n , it also extracts some tree from B_n .*

Proof. Without loss of generality we may assume that the XPath expression p has the form

$$/a_1[\vec{b}_{12} | \vec{b}_{22} | \dots] / a_2[\vec{b}_{21} | \vec{b}_{22} | \dots] \dots / a_{q_1}[\vec{b}_{q_11} | \vec{b}_{q_12} | \dots] \quad (6.4)$$

where $a_1, a_2 \dots a_{q_1}$ are location steps on the primary location path. Each a_i has form $/\text{child}::x$ or $/\text{descendant-or-self}::x$. The symbol x is either from the earlier defined subset Σ of the infinite input alphabet for XML documents or $x = *$. Then \vec{b}_{ij} are XPath sub-expressions from filter conditions of \vec{a} . Recursively each \vec{b}_{ij} has again the form of (6.4).

Let s denote the number and $\vec{p}_1, \vec{p}_2, \vec{p}_3, \dots, \vec{p}_s$ denote the list of all location paths in p . For example the expression $/a/a[b/b|c[d/d]]/a/a$ has the following list of location paths: $p_1 = /a/a/a/a$, $p_2 = /b/b$, $p_3 = c$, $p_4 = d/d$.

Let q_i be number of axes in p_i . In our example $q_1 = 4$, $q_2 = 2$, $q_3 = 1$, $q_4 = 2$. Because $\vec{p}_1, \vec{p}_2, \vec{p}_3, \dots, \vec{p}_s$ is a complete partition of all axes in p we have

$$\sum_{i=1}^s q_i = q(n) \quad (6.5)$$

Let $t = \#x_1\#x_1\#x_2\#x_2 \dots \#x_n\#x_n\#^*$, $x_i \in \Sigma^{n^k}$ be an arbitrary tree from A_n . Because p extracts all trees in A_n there exists at least one embedding e of the expression p into the tree t such that the last location step (p_{1q_1}) in the top-level location path (\vec{p}_1) navigates only to the last node $\#^*$, as illustrated in Figure 6.4.

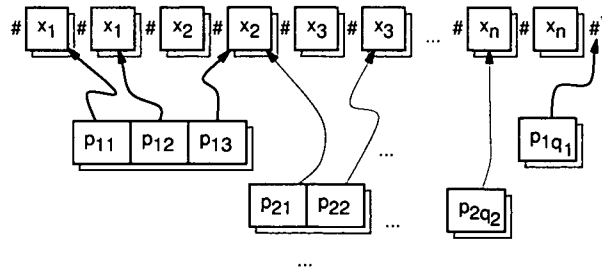


Figure 6.4: Embedding of p into $\#x_1\#x_1\#x_2\#x_2 \dots \#x_n\#x_n\#^*$

The $XP(/, //, [\leq^{m^\alpha}, |, *)$ fragment has only forward axes. Therefore, with respect to the embedding e we can for each location path $\vec{p}_i = /p_{i1}/p_{i2} \dots /p_{iq_i}$ partition the tree $\#x_1\#x_1\#x_2\#x_2 \dots \#x_n\#x_n\#^*$ into a list of subsequent intervals such that j -th interval is a sequence of nodes from right-sibling of node matched by location step $p_{i,j-1}$ up to the node matched by location step p_{ij} . Intuitively said, in the j -th interval the location step p_{ij} searches for its witness node in embedding e . For location paths from filter conditions which do not need to start at first and end at last node of tree t we allow two special prefix and suffix intervals not associated with any location step (marked with '-'), see Figure 6.5.

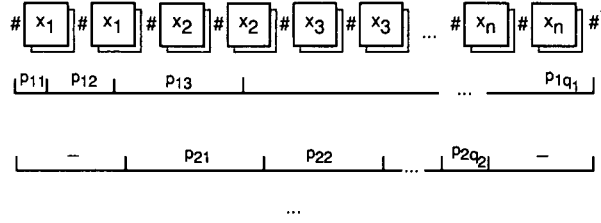


Figure 6.5: Partition of nodes in tree $\#x_1\#x_1\#x_2\#x_2 \dots \#x_n\#x_n\#^*$

For a fixed embedding of expression p into the tree $\#x_1\#x_1 \dots \#x_n\#x_n\#^*$ let $(s_{i0}, s_{i1}, \dots, s_{iq_i})$ denote the tuple of location steps (intervals) from location path \vec{p}_i to which belong the $\#$ labelled nodes.

Let S be the set of all possible tuples of location step (intervals) defined by all possible embeddings of the expression p into all trees in A_n .

Expression p has only child and descendant axes and extracts all trees in A_n which all have size $p_2(n) = 2n^{k+1} + 2n + 1$. Therefore, for m - the number of steps on the main location path of the expression p holds $m \leq p_2(n)$. From the definition of the fragment $XP(/, //, [\leq^{m^\alpha}, |, *)$ it follows that the number of all location paths in p , denoted earlier as s , is bounded by

$$s \leq 1 + m^\alpha \leq 1 + p_2(n)^\alpha \quad (6.6)$$

Because each \vec{p}_i has q_i location steps the size of S has upper bound

$$\begin{aligned} |S| &\leq q_1^{2n+1} \cdot (q_2 + 1)^{2n+1} \cdot \dots \cdot (q_s + 1)^{2n+1} \\ (6.5) \quad &\leq (q(n) + 1)^{2n+1} \cdot \dots \cdot (q(n) + 1)^{2n+1} \\ &\leq (q(n) + 1)^{s3n} \end{aligned}$$

Let A'_n be the largest subset of trees from A_n which map to the same tuple of location steps in S . Because A_n contains $|\Sigma|^{n^{k+1}}$ trees, the size of A'_n is at least

$$|A'_n| \geq \frac{|A_n|}{|S|} \geq \frac{|\Sigma|^{n^{k+1}}}{|S|} \geq \frac{|\Sigma|^{n^{k+1}}}{(q(n) + 1)^{s3n}} \quad (6.7)$$

For a fixed $x \in \Sigma^{n^k}$ there are at most

$$\sum_{i=0}^{\lfloor \frac{n^k}{4} \rfloor} \binom{n^k}{i} (|\Sigma| - 1)^i$$

trees $y \in \Sigma^{n^k}$ such that $d(x, y) \leq \frac{n^k}{4}$. Using an inequality from [Ang90], there exists a constant c , $0 < c < 1$ such that

$$\sum_{i=0}^{\lfloor \frac{n^k}{4} \rfloor} \binom{n^k}{i} (|\Sigma| - 1)^i < |\Sigma|^{cn^k} \quad (6.8)$$

Therefore, in a set with at least $|\Sigma|^{cn^k}$ elements exists y such that $d(x, y) > \frac{n^k}{4}$.

Let $A'_n[i]$ be the set of trees in the projection of A'_n to i -th block, that is

$$A'_n[i] = \{ x_i \mid x_i \in \Sigma^{n^k}, \exists x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n \in \Sigma^{n^k} : \\ \#x_1\#x_1 \dots \#x_{i-1}\#x_i\#x_i\#x_{i+1} \dots \#x_n\#x_n \in A'_n \}$$

Among the indices $i = 1, 2, \dots, n$ there are at least $\frac{n}{2}$ indices such that $|A'_n[i]| \geq |\Sigma|^{cn^k}$, that is

$$|I| = |\{i \mid |A'_n[i]| \geq |\Sigma|^{cn^k}\}| \geq \frac{n}{2} \quad (6.9)$$

Otherwise assuming $|I| < \frac{n}{2}$ the number of trees in A'_n is bounded by

$$\begin{aligned} |A'_n| &\leq \left(|\Sigma|^{n^k}\right)^{|I|} \left(|\Sigma|^{cn^k}\right)^{n-|I|} \\ &\leq \left(|\Sigma|^{n^k}\right)^{\frac{n}{2}} \left(|\Sigma|^{cn^k}\right)^{\frac{n}{2}} \\ &\leq |\Sigma|^{\frac{1}{2}n^{k+1}(1+c)} \end{aligned} \quad (6.10)$$

Then combining (6.7) and (6.10) we have

$$\frac{|\Sigma|^{n^{k+1}}}{(q(n) + 1)^{s3n}} \leq |\Sigma|^{\frac{1}{2}n^{k+1}(1+c)} \quad (6.11)$$

This leads to contradiction, because from choice of k in (6.3) it is implied that

$$s \stackrel{(6.6)}{\leq} 1 + [2n^{k+1} + 2n + 1]^\alpha \leq 2[2n^{k+1} + 2n + 1]^\alpha \leq [n^{k+2}]^\alpha \stackrel{(6.3)}{<} n^{k-1} \quad (6.12)$$

and then we obtain

$$\begin{aligned} n &> \frac{b}{a} \log_{|\Sigma|} n && \forall a \in (0, 1), \forall b > 1 \\ an &> b \log_{|\Sigma|} n \\ an^{k+1} &> bnn^{k-1} \log_{|\Sigma|} n \\ an^{k+1} &\stackrel{(6.12)}{>} bns \log_{|\Sigma|} n \\ an^{k+1} &> 3ns \log_{|\Sigma|} (q(n) + 1) && \frac{\log n^j}{n} \rightarrow 0 \\ |\Sigma|^{\frac{1}{2}n^{k+1}(1-c)} &> (q(n) + 1)^{s3n} \\ \frac{|\Sigma|^{n^{k+1}}}{(q(n)+1)^{s3n}} &> |\Sigma|^{\frac{1}{2}n^{k+1}(1+c)} \end{aligned}$$

This is a contradiction with (6.11) for all sufficiently large n and therefore, the equation (6.9) must be true.

Let us choose any fixed tree t from A_n

$$t = \#x_1\#x_1\#x_2\#x_2\#\dots\#x_n\#x_n\#^*$$

We construct a tree $t' = \#x_1\#y_1\#x_2\#y_2\#\dots\#x_n\#y_n\#^*$ from B_n in the following way: If $i \notin I$ let $y_i := x_i$. In case $i \in I$ by (6.8) there exists a subtree z_i in $A'[i]$ such that $d(x_i, z_i) > \frac{n^k}{4}$. Then let $y_i := z_i$.

Because $|I| \geq \frac{n}{2}$ there are at least $\frac{n}{2}$ indices such that $d(x_i, z_i) > \frac{n^k}{4}$ and so $t' \in B_n$. Because $t \in A_n$, all embeddings of p into t navigate only to the last $\#^*$. By replacing x_i with y_i we preserve for the $\#$ labelled nodes sets of location steps which search for their witness node, as illustrated in Figure 6.6.

Therefore, there exists at least one embedding of p to t' which navigates to the last $\#^*$ labelled node, so $t' \in B_n$. \square

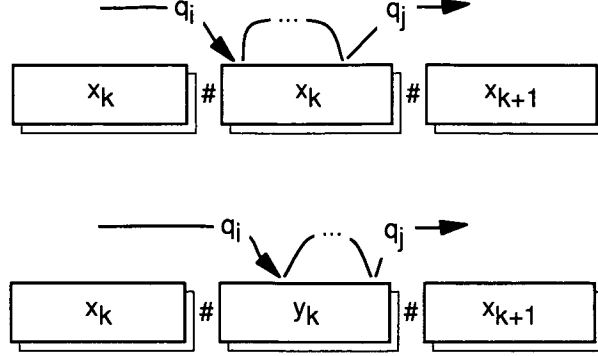


Figure 6.6: Preserving sets of location steps

Proof. (Theorem 6.3.3). Let $q(n)$ be an arbitrary nondecreasing polynomial (derived from running time of some learning algorithm A). Assume that the learning algorithm asks an equivalence query $\forall t : r(t) = p(t)$ with an XPath expression r from the $XP(/, //, \sqsubseteq^{m^a}, |, *)$ fragment such that $|r| \leq q(n)$.

Lemma 6.3.4 implies that for all sufficiently large n either r does not extract some tree from A_n or extracts a tree from B_n .

Let it be the case that r does not extract a tree t from A_n that is $t \in A_n \wedge r(t) \neq nodes_*(t)$. Because A_n is subset of each concept in T_n , the number of concepts in T_n which do not extract t is 0. Therefore, we can return t as an answer to the equivalence query while the following equation is true

$$\{p \in T_n \mid p(t) = r(t)\} = 0 < \frac{|T_n|}{q(n)}$$

Let it be the case that p extracts a tree $t = \#x_1\#y_1 \dots \#x_n\#y_n\#^*$ from B_n , that is $t \in B_n \wedge t \in L(r)$. If r navigates on t also to some other node than the last $\#^*$, the number of concepts in T_n which extract t is 0. So, assume that r navigates only to the last $\#^*$ node. From definition of L_{jn} it is implied that

$$x_i[j] \neq y_i[j] \implies x_i y_i \notin L_{jn}$$

If $d(x_i, y_i) > \frac{n^k}{4}$ there are at least $\frac{n^k}{4}$ unequal symbols and therefore, $x_i y_i$ belongs to at most $n^k - \frac{n^k}{4} = \frac{3n^k}{4}$ languages L_{jn} . So, there are at most $(n^k)^{\frac{n}{2}} \left(\frac{3n^k}{4}\right)^{\frac{n}{2}}$ concepts in T_n which extract the tree t . Therefore, we can return t as an answer to the equivalence query and for all sufficiently large n the following equation is true

$$\{p \in T_n \mid p(t) = r(t)\} \leq (n^k)^{\frac{n}{2}} \left(\frac{3n^k}{4}\right)^{\frac{n}{2}} \leq \left(\frac{3}{4}\right)^{\frac{n}{2}} |T_n| < \frac{|T_n|}{q(n)}$$

To summarise, for all sufficiently large n it is possible for each equivalence query r of size at most $q(n)$ to return an example t of size at most $p_2(n)$ which rules out only an exponentially small fragment of all concepts in T_n and therefore $XP(/, //, \sqsubseteq^{\leq m^\alpha}, |, *)$ has the property of approximate fingerprints. \square

Corollary 6.3.5. *The fragments $XP(/, //, \sqsubseteq^{\leq m^\alpha}, |, *)$ and $XP(/, \sqsubseteq^{\leq m^\alpha}, |, *)$ are not polynomially exactly learnable using only equivalence queries.*

Proof. Implied from Theorem 6.3.3, the main theorem in [Ang90] and that the Lemma 6.3.4 and Theorem 6.3.3 hold also for the $XP(/, \sqsubseteq^{\leq m^\alpha}, |, *)$ fragment. \square

Corollary 6.3.6. *The super fragments of the fragments $XP(/, \sqsubseteq^{\leq m^\alpha}, |, *)$ and $XP(/, //, \sqsubseteq^{\leq m^\alpha}, |, *)$ with added following, following-sibling, preceding or preceding-sibling axes and/or with allowed presence of attributes are not polynomially exactly learnable using only equivalence queries.*

Proof. Assume there would be an algorithm for the above defined sequence of concept classes T_1, T_2, \dots which returns a learned XPath expression p from any of these super fragments. Then we can construct an equivalent expression p' from $XP(/, //, \sqsubseteq^{\leq m^\alpha}, |, *)$ or $XP(/, \sqsubseteq^{\leq m^\alpha}, |, *)$ by removing attribute tests and location paths which contain following, following-sibling, preceding or preceding-sibling axes. \square

Theorem 6.3.7. *The XPath fragment $XP(/, //, *)$ has the approximate fingerprints.*

Proof. Let $\Sigma, |\Sigma| \geq 2$ be some finite subset of the (infinite) input alphabet of XML documents.

Let T_1, T_2, T_3, \dots be a sequence of concept classes defined as follows. For $i = 1 \dots n$ is L_{ian} set of all single-branched trees of size $2n + 2$ with the first and the middle symbol $\#$ such that $(i + 1)$ -th and $(n + i + 2)$ -th node have the label $a \in \Sigma$. That is

$$L_{ian} = \{\#x_1ax_2\#y_1ay_2 \mid x_1, y_1 \in \Sigma^{i-1}, x_2, y_2 \in \Sigma^{n-i}\}$$

A concept is n concatenations of L_{ian} languages plus the appended symbol $\#^*$. For each tree t from the concept only the last node ($\#^*$) is the result of XPath navigation on the tree t . The concept class T_n is a set of all possible concatenations:

$$T_n = \{L_{i_1a_1n} \cdot L_{i_2a_2n} \cdot L_{i_3a_3n} \cdots L_{i_na_nn}\#^* \mid 1 \leq i_1, \dots, i_n \leq n, a_1, \dots, a_n \in \Sigma\}$$

Clearly, $|T_n| \geq 2$ for $n \geq 2$.

Let us take the following XPath expression from the $XP(/, //, *)$ fragment:

$$\begin{aligned} & / \# \underbrace{/* \dots /*}_{i_1 - 1} / a_1 \underbrace{/* \dots /*}_{n - i_1} / \# \underbrace{/* \dots /*}_{i_1 - 1} / a_1 \underbrace{/* \dots /*}_{n - i_1} \\ & \dots \\ & / \# \underbrace{/* \dots /*}_{i_n - 1} / a_n \underbrace{/* \dots /*}_{n - i_n} / \# \underbrace{/* \dots /*}_{i_n - 1} / a_n \underbrace{/* \dots /*}_{n - i_n} \\ & / \# \end{aligned}$$

The language generated by this XPath expression is exactly $L_{i_1 a_1 n} \cdot L_{i_2 a_2 n} \cdot L_{i_3 a_3 n} \dots L_{i_n a_n n} \#^*$. For each tree t from this language the XPath expression navigates only to its last node. The expression has $p_1(n) = (2 + 2n)n + 1$ axes.

Let A_n be the set of n pairs of trees from Σ^n that is

$$A_n = \{ \#x_1 \#x_1 \#x_2 \#x_2 \dots \#x_n \#x_n \#^* \mid x_i \in \Sigma^n \}$$

and let B_n be set of n pairs of trees from Σ^n such that at least $\frac{n}{2}$ pairs have at least $\frac{n}{4}$ unequally labelled nodes and arbitrary subset of nodes which includes the last $\#$ may be result of the XPath navigation.

$$B_n = \{ \#x_1 \#y_1 \#x_2 \#y_2 \dots \#x_n \#y_n \#^* \mid x_i, y_i \in \Sigma^n, d(x_i, y_i) > \frac{n}{4} \text{ for at least } \frac{n}{2} \text{ values of } i \}$$

The size of any tree in A_n and B_n or of any tree in any concept from T_n is bounded by polynomial $p_2(n) = (2n + 2)n + 1$.

Let $q(n)$ be an arbitrary nondecreasing polynomial (derived from running time of some learning algorithm A). Assume that the learning algorithm asks an equivalence query $\forall t : r(t) = p^?(t)$ with an XPath expression r from the $XP(/, //, *)$ fragment such that $|r| \leq q(n)$.

Because Lemma 6.3.4 holds also for $\alpha = 0$ and $k = 1$, it implies that for all sufficiently large n either r does not extract some tree from A_n or extracts a tree from B_n .

Assume the case that r does not extract a tree $t = \#x_1 \#y_1 \dots \#x_n \#y_n \#^*$ from A_n that is $t \in A_n \wedge r(t) \neq \text{nodes}_*(t)$. From definition of L_{jan} implies

$$\begin{aligned} x_i[j] = y_i[j] = a & \implies x_i y_i \in L_{jan} \\ & \wedge \\ & x_i y_i \notin L_{jbn} \quad \forall b \in \Sigma - \{a\} \end{aligned}$$

Therefore, $x_i y_i$ belongs to at most n languages L_{jan} and so there are at most n^n concepts in T_n which extract the tree t . Therefore, we can return t as an answer to the equivalence query while the following equation is true

$$\{p \in T_n \mid p(t) = r(t)\} \leq n^n \leq \left(\frac{1}{|\Sigma|}\right)^n |T_n| < \frac{|T_n|}{q(n)}$$

Consider the case that r extracts a tree $t = \#x_1\#y_1 \dots \#x_n\#y_n\#^*$ from B_n , that is $t \in B_n \wedge t \in L(r)$. If r navigates on t also to some other node than the last $\#^*$ the number of concepts in T_n which extract t is 0. So assume that r navigates only to the last $\#^*$ node. From definition of L_{jan} it is implied that

$$x_i[j] \neq y_i[j] \implies x_i y_i \notin L_{jan} \quad \forall a \in \Sigma$$

If $d(x_i, y_i) > \frac{n}{4}$, there are at least $\frac{n}{4}$ unequal symbols and therefore, $x_i y_i$ belongs to at most $|\Sigma|n - \frac{|\Sigma|n}{4} = \frac{3|\Sigma|n}{4}$ languages L_{jan} . So, there are at most $(|\Sigma|n)^{\frac{n}{2}} \left(\frac{3|\Sigma|n}{4}\right)^{\frac{n}{2}}$ concepts in T_n which extract the tree t . Therefore, we can return t as an answer to the equivalence query and for all sufficiently large n the following equation remains true

$$\{p \in T_n \mid p(t) = r(t)\} \leq (|\Sigma|n)^{\frac{n}{2}} \left(\frac{3|\Sigma|n}{4}\right)^{\frac{n}{2}} \leq \left(\frac{3}{4}\right)^{\frac{n}{2}} |T_n| < \frac{|T_n|}{q(n)}$$

□

Corollary 6.3.8. *The XPath fragments $XP(/, //, *)$ and $XP(/, *)$ are not polynomially exactly learnable using only equivalence queries.*

Proof. Lemma 6.3.4 and Theorem 6.3.7 hold for both fragments. Then it is implied from Theorem 6.3.7 and the main theorem in [Ang90]. □

Corollary 6.3.9. *The super fragments of the XPath fragments $XP(/, //, *)$ and $XP(/, *)$ with added following, following-sibling, preceding or preceding-sibling axes and/or with allowed presence of attributes are not polynomially exactly learnable using only equivalence queries.*

6.4 Membership Queries

Another type of queries often used for active learning are membership queries. Applied to the DOM trees, a membership query is a question of the form $\text{nodes}_*(t) = p^?(t)$. Answers are either "yes" or "no".

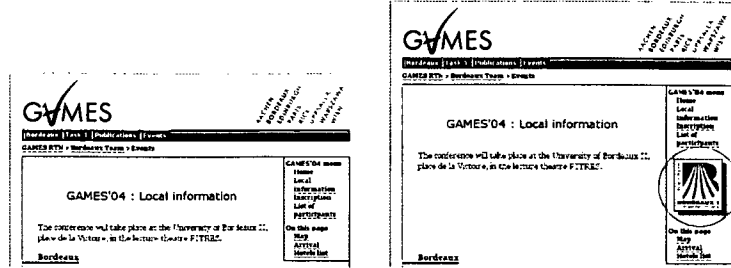


Figure 6.7: Visualisation of membership queries

Unfortunately, the membership queries do not have, in context of HTML wrapper induction, a natural visualisation. The reason is that the learning algorithm needs to ask queries about trees with various shapes. This leads to a problem of constructing for the query a valid HTML page which can be rendered. Moreover, this also leads to artificially looking Web pages. Such Web pages are then unfamiliar to the human wrapper designer. An example of such unnatural visualisation is in Figure 6.7, where the learning algorithm tries to insert an image into the right sidebar.

In the following part we present limits on learnability using only membership queries by reduction to the equivalence problem of XPath expressions.

Definition 6.4.1. XPath expressions p_1 and p_2 are equivalent if $\forall t \in T_{\Sigma} : p_1(t) = p_2(t)$, denoted as $p_1 \equiv p_2$. The XPath expression equivalence problem is for the given expressions p_1 and p_2 to decide whether $p_1 \equiv p_2$.

Theorem 6.4.2. Assuming $P \neq coNP$, the XPath fragment $XP(/, //, *, [])$ and all its super fragments are not polynomially exactly learnable using only membership queries.

Proof. Assume, there exists a learning algorithm A for exact query based learning of expressions from the fragment $XP(/, //, *, [])$ or some of its super fragments. Then Algorithm 8 solves the equivalence problem for this fragment.

Correctness is implied from the exactness of the learning. If the algorithm A outputs an expression q then all expressions which are consistent with all query answers must be equivalent. Because p_1 and p_2 are consistent with all query answers they are equivalent.

In [GKP02] a PTIME algorithm for XPath evaluation is presented. Because the learning algorithm A runs in polynomial time with respect to size of p_1 , the whole Algorithm 8 runs in PTIME.

Algorithm 8 solving equivalence of p_1 and p_2 using membership query learning

simulate learning algorithm A

 if A asks a membership query with tree $t \in T_{\Sigma^*}$ in form: $\text{nodes}_*(t) = p^?(t)$
 compute $N_1 = p_1(t)$, $N_2 = p_2(t)$
 if $N_1 \neq N_2$
 return “ p_1 not equal to p_2 ”
 else if $N_1 = \text{nodes}_*(t)$
 answer “yes” to the query
 else
 answer “no” to the query
 if A outputs learned expression q
 return “ p_1 equal to p_2 ”

The [MS02] shows that the equivalence problem is coNP-complete for the XPath fragment $XP(/, //, *, [])$ and also for its sub-fragments, where the number of $*$ is bounded by a constant $c \geq 2$ or the number of $[]$ is bounded by a constant. Therefore, the equivalence problem for each of their super fragments is also coNP-complete or harder.

Therefore, assuming $P \neq \text{coNP}$ yields a contradiction with Algorithm 8 running in PTIME. \square

6.5 Tree-prefix Queries

We have seen in the previous sections limits of learnability using equivalence and membership queries. Therefore, in this part we propose and then study a more suitable type of queries which we call tree-prefix queries. We combine these tree-prefix queries with equivalence queries into a polynomial learning algorithm for the $XP(/, [], *)$ fragment.

Definition 6.5.1. Tree-prefix query is a question about two XPath expressions p_1 and p_2 in the form:

$$\forall t \in T_{\Sigma} : p_1 // * (t) \supseteq p_2(t)$$

The answer to the query is either “yes” or “no” with a counterexample $t \in T_{\Sigma^*}$ such that $\text{nodes}_*(t) \subseteq p_2(t)$ and $\exists x : x \in \text{nodes}_*(t) \wedge x \notin p_1 // * (t)$.

Let $p_1 \triangleleft p_2$ be the short notation of the tree-prefix query.

Described with words, the query asks whether in an arbitrary tree all nodes navigated by the expression p_2 are located in the subtrees of nodes

navigated by the expression p_1 . Because we want to pose these queries to a human wrapper designer, we allow for the returned counterexample to contain only a subset of the \star -nodes navigated by the expression p_2 . Nodes navigated by the expression p_1 form a prefix-tree of the tree defined by the nodes navigated with the expression p_2 . Due to this reason name of the query results.

For example, the answer to the query $/a/b[d]/c \triangleleft /a/b[d/e]/c[f]$ is “yes” as illustrated in Figure 6.8. On the other side, answer to the query $/a/b[d]/c \triangleleft /a/b/c[f]$ is “no” with the counterexample in Figure 6.9.

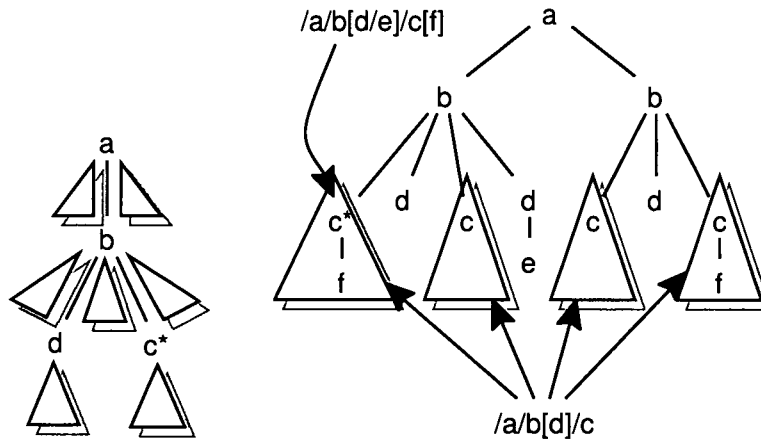


Figure 6.8: Shape of trees fulfilling $/a/b[d]/c \triangleleft \dots$, tree prefix on a particular tree instance

By $XP(/, [], *)$ we denote an XPath fragment consisting of wild card node matching ($*$), child axes ($/$) and filter conditions ($[]$).

The fragment does not contain the backward parent axis. Fortunately, adding of the parent axe does not increase the expressive power, because the fragments $XP(/, [], *)$ and $XP(/, [], *, \text{parent})$ are equal under root equivalence [BFK03]. Moreover, there exists a deterministic polynomial translation between these fragments. Therefore, a learning algorithm for the first fragment can be used also for learning of expressions from the other fragment.

The fragment $XP(/, [], *, \text{parent}::)$ also does not contain the transitive axes such as descendant-or-self ($//$), ancestor, etc. We trade their absence for good visualisation of the tree-prefix queries. Answering of an tree-prefix query by the wrapper designer then reduces on an HTML DOM tree to visual verification that all target nodes are located inside of the highlighted rectangles. The Figure 6.10 illustrates the visualisation of an interaction between a human designer and a wrapper system, including answers with a

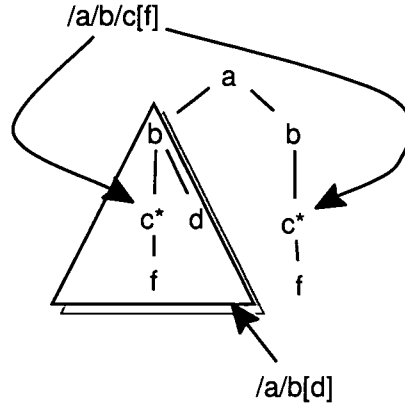


Figure 6.9: Counterexample for the query $/a/b[d]/c \triangleleft /a/b/c[f]$

counter example. Note, that this visualisation additionally assumes that in the context of HTML wrapper induction are the input HTML documents very similarly structured and therefore, checking through all possible HTML DOM trees can be reduced to verification on small number (2-3) of similarly structured documents.

Now, we present Algorithm 9 for learning of the $XP(/, [], *)$ using tree-prefix queries. The tree-prefix queries define a partial order on the XPath expressions in the fragment. With respect to this ordering the algorithm in a systematic top-down manner trims the hypotheses space using tree-prefix queries until it reaches the correct target XPath expression p .

The algorithm runs in two phases. The first phase finds labels on the main location path using a positive example. At its end, all consistent hypotheses have the form of $/a_1[\bigwedge_{\emptyset \triangleleft f} f]/a_2[\bigwedge_{\emptyset \triangleleft f} f] \dots a_n[\bigwedge_{\emptyset \triangleleft f} f]$.

In the second phase, filter conditions are discovered using the following technique. The algorithm chooses an existing node in the hypothesis expression and tries to extend it by appending another filter condition in the form $[*]$ or $[x_1 \wedge x_2 \dots]$. The function $\text{subst}(\text{expression}, \text{node}, \text{subtree})$ used in Algorithm 9 for this purpose takes an XPath expression and replaces its node with the given subtree. This trims top-down the hypothesis space of the possible XPath expressions. A subsequent equivalence query is posed to check whether the correct hypothesis has been reached.

Note, that it is possible in an implementation to reduce the number of queries posed to the user, because some of the queries are trivially fulfilled or can be ruled out with the already given examples. An example run of the algorithm is illustrated in Figure 6.11.

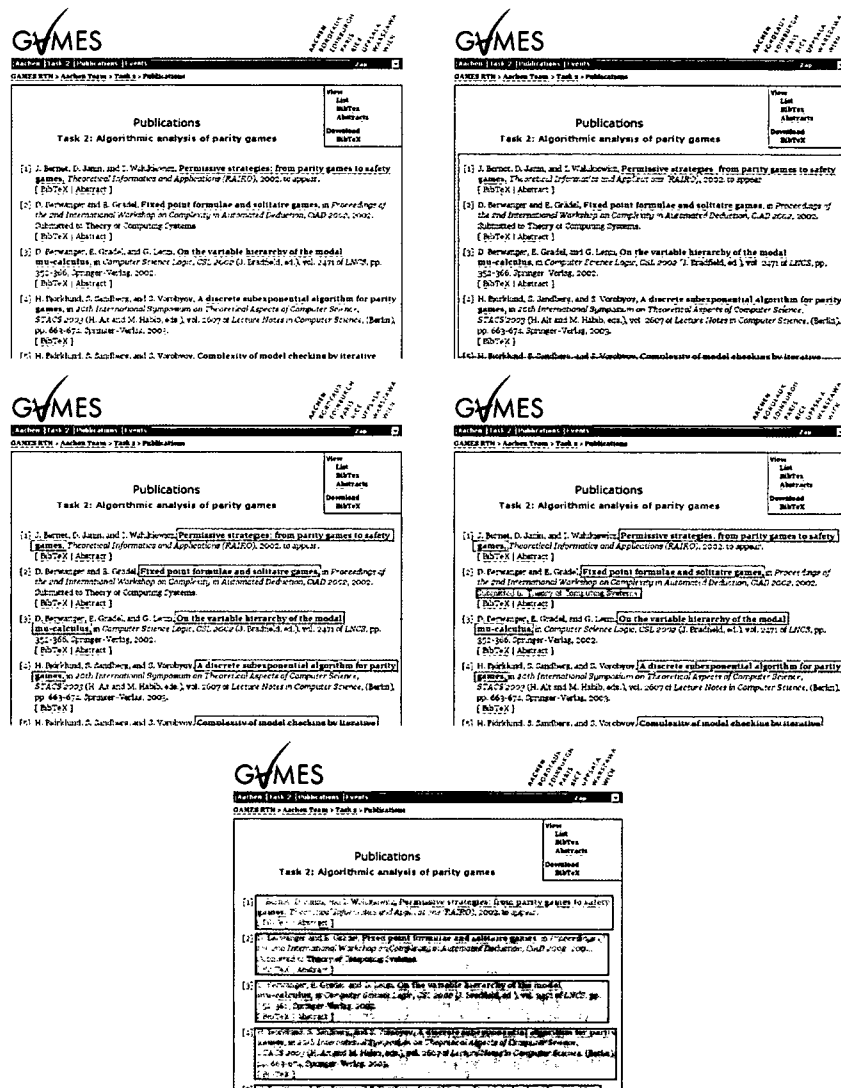


Figure 6.10: Visualisation of tree-prefix queries

Algorithm 9 learning with tree-prefix and equivalence queries

```

if ask:  $\emptyset \equiv p?$ 
    return  $\emptyset$ 
else
    let  $(x_{11}x_{12}\dots x_{1n}), \dots, (x_{k1}x_{k2}\dots x_{kn})$  be all the
    branches in the received counterexample to target
    nodes (labelled with  $\star$ )
    for  $i = 1$  to  $n$ 
        if  $|\{x_{1i}, x_{2i}, \dots, x_{ki}\}| \geq 2$ 
             $a_i = \star$ 
        else if ask:  $/a_1/a_2\dots/a_{i-1}/x_{1i} \triangleleft p?$ 
             $a_i = x_{1i}$ 
        else
             $a_i = \star$ 

    hypothesis =  $/a_1/a_2/\dots/a_n$ 
    if hypothesis  $\equiv p?$  return hypothesis

    queue = []
    for node  $a_i$  in hypothesis: queue.append( $a_i$ )

    while not queue.isEmpty()
        testedNode = queue.pop()

        if ask: subst(hypothesis, testedNode, testedNode[*])  $\triangleleft p?$ 
            U = labels of nodes matched with testNode when na-
                vigating using the hypothesis on any example
            L={}

            for label  $x$  in U
                if ask: subst(hypothesis,
                            testedNode,
                            testedNode[ $x \wedge_{y \in L} y$ ])  $\triangleleft p?$ 
                     $L = L \cup \{x\}$ 

            if L is empty
                hypothesis =
                    subst(hypothesis, testedNode, testedNode[*])
                queue.append(node * from testedNode[*])
            else
                hypothesis =
                    subst(hypothesis, testedNode, testedNode[ $\wedge_{y \in L} y$ ])
                for node  $y$  in testedNode[ $\wedge_{y \in L} y$ ]: queue.append(y)

    if ask: hypothesis  $\equiv p?$ : return hypothesis

```

query	answer
$\emptyset \equiv p?$	no - example 1
$/body \triangleleft p?$	yes
$/body/table \triangleleft p?$	yes
$/body/table/tr \triangleleft p?$	yes
$/body/table/tr/td \triangleleft p?$	no - example 2
$/body/table/tr/* \equiv p?$	no - example 3
$/body/table/tr/*[*] \triangleleft p?$	yes
$/body/table/tr/*[img] \triangleleft p?$	no - example 1
$/body/table/tr/*[h3] \triangleleft p?$	yes
$/body/table/tr/*[h3 \wedge p] \triangleleft p?$	yes
$/body/table/tr/*[h3 \wedge p] \equiv p?$	no - example 3
$/body/table/tr[*]/* [h3 \wedge p] \triangleleft p?$	yes, trivial
$/body/table/tr[td]/* [h3 \wedge p] \triangleleft p?$	no - example 3
$/body/table/tr[th]/* [h3 \wedge p] \triangleleft p?$	no - example 1
$/body/table/tr[*]/* [h3 \wedge p] \equiv p?$	no - example 3
$/body/table/tr[*]/* [h3[*] \wedge p] \triangleleft p?$	no - example 1
$/body/table/tr[*]/* [h3 \wedge p[*]] \triangleleft p?$	yes
$/body/table/tr[*]/* [h3 \wedge p[b]] \triangleleft p?$	yes
$/body/table/tr[*]/* [h3 \wedge p[b]] \equiv p?$	no - example 3
$/body/table/tr[*[*]]/* [h3 \wedge p[b]] \triangleleft p?$	yes
$/body/table/tr[*[h3]]/* [h3 \wedge p[b]] \triangleleft p?$	yes
$/body/table/tr[*[h3 \wedge p]]/* [h3 \wedge p[b]] \triangleleft p?$	yes
$/body/table/tr[*[h3 \wedge p \wedge img]]/* [h3 \wedge p[b]] \triangleleft p?$	yes
$/body/table/tr[*[h3 \wedge p \wedge img]]/* [h3 \wedge p[b]] \equiv p?$	yes
$/body/table/tr[*/img]/* [h3 \wedge p/b] \equiv p?$	simple minimisation applied

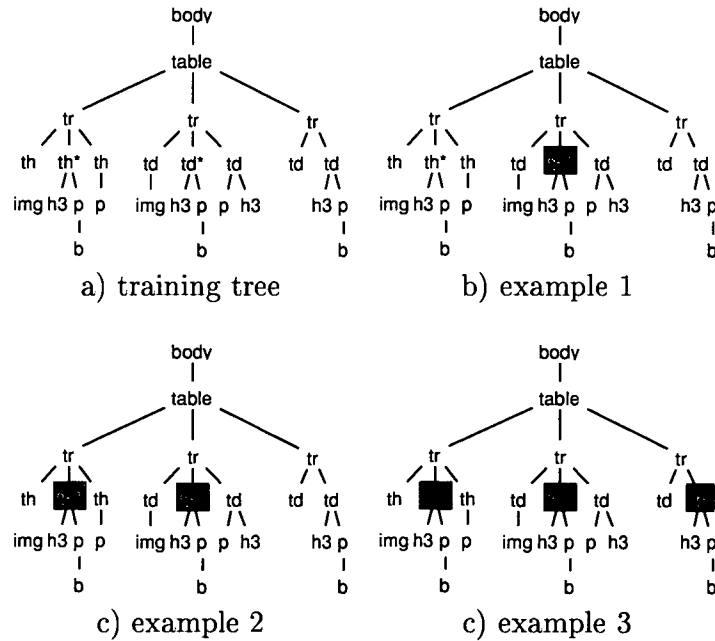


Figure 6.11: Learning of the expression $/body/table/tr[*/img]/* [h3 \wedge p/b]$ using tree-prefix queries

Theorem 6.5.2. *Algorithm 9 learns the $XP(/, [], *)$ fragment and runs in time $O(|p^?||t|)$ where $|p^?|$ is size of the target XPath expression and $|t|$ is size of the largest received example.*

Proof. Correctness implies from the property of partial ordering defined with the tree-prefix queries. Let p_1 and p_2 be two XPath expressions such that $p_1 \triangleleft p_2$. Let p be an arbitrary XPath expression (the hypothesis). Let p' be an expression constructed from p by replacing its part p_1 with expression p_2 . Then holds $p \triangleleft p'$.

The algorithm systematically searches the hypotheses space with respect to this ordering. Therefore, each target XPath expression is once eventually reached.

The for loops run at most $O(|t|)$ times. A node is appended to the queue only if a filter condition f from hypothesis is replaced with a more specialised form $f[*]$ or $f[\wedge x]$ which is a prefix of the corresponding filter condition in $p^?$. Therefore, the while loop runs at most $O(|p^?|)$ times. \square

Chapter 7

Classification of DOM Attributes

In the previous chapter has been studied learning of tree structures. So far, we did not discuss handling of the DOM attributes. Of course, one straightforward solution is to handle the attributes with the DOM Level1 approach that means to view them as regular DOM nodes and then apply the algorithms of the previous chapter.

In this chapter we study another approach that can be both applied as an extension of the previous query-based algorithm and used separately. It becomes more flexible, especially in context of the HTML DOM trees, which have many types of attributes with diverse values. This allows us to build into the learning algorithm various heuristics using knowledge of the HTML standard. Moreover, we can easily add to the algorithm any additional features which are already used in the existing wrapper systems such as syntactic and semantic concepts as defined in [BFG01].

Knowledge representation using decision trees has been already used in many applications. Decision trees are a type of symbolic machine learning and are based on the divide and conquer method. The training data are gradually divided into smaller and smaller subsets. At the beginning of the learning using decision trees is one big set with all training data, at the end there is a partition of the training data into a system of subsets.

In this chapter we present a solution for classification of extraction instances that is based on decision trees using the so called “entropy” measure. Let us note that there are also other types of measures such as the Gini index or χ^2 . But our experiments did not show noticeable advantage when using those measures.

The entropy based queries will show us how to undermine the problem of choice between restricting the probabilistic distribution of examples and the need of an additional channel on information about the target concept, which is known from the trade-offs between the active and passive learning models

described in the Chapter 1. Of course, this approach must trade strong assumptions for a simple algorithm performing well in some applications. Types of constraints that we are going to assume are restricted (finite) number of question, small (finite) hypotheses space, independence of certain random events and ability to estimate some probabilities.

7.1 Entropy

Suppose that we have a teacher T and learner L . The learner is willing to discover the unknown target concept $c^?$. Let $Q = \{q_1, q_2, \dots, q_n\}$ be the set of all possible questions the learner L can ask before applying the divide and conquer step. Let $A_{q_i} = \{a_1^i, a_2^i, \dots, a_n^i\}$ be the set of answers the teacher T can answer on learner's question q_i .

We denote $P(\text{answer}(q_i) = a_j^i)$ as probability of the random event that the teacher answers the question q_i with answer a_j^i . Note, we implicitly assume here that answering answers a_k and a_l for arbitrary k and l are two independent events. That is

$$P(\text{answer}(q_i) = \{a_k^i, a_l^i\}) = P(\text{answer}(q_i) = a_k^i) + P(\text{answer}(q_i) = a_l^i)$$

Now, we would like to help the learner to identify the most valuable question. Ideally, we would use a metric for measuring how good each of the questions is and choose the question with the highest value. From an usable measure of the informativeness of a question we would expect the following natural properties:

1. I is continuous.

That is, we expect the measure of the question informativeness to be a continuous function of the probabilities of its answers.

2. If the set of all possible answers $A = \{a_1, a_2, \dots, a_n\}$ is splitted for a particular question q into m disjoint subsets A_1, \dots, A_m such that $A = \bigcup_{i=1}^m A_i$; and $A_i \cap A_j = \emptyset$ for all i and j , then informativeness of the question q with the original answer set A should be equal to the weighted sum of informativenesses of the question q with the reduced answer subsets. That is

$$I(A) = \sum_{i=1}^m P(A_i) \cdot I(A_i) \quad (7.1)$$

where

- $I(A)$ - informativeness of question q with set of the possible answers A
- $I(A_i)$ - informativeness of question q with set of the possible answers A_i
- $P(A_i)$ - probability that answer returned for question q will be from the set A_i

3. If all answers $A = \{a_1, a_2, \dots, a_n\}$ for a question q have equal probability ($= \frac{1}{n}$), then the informativeness $I(q)$ of the question q should grow with n .

That is, the more answers are possible for a question, then with receiving one particular answer from this set we can exclude larger part of the hypotheses space. So the question has larger informativeness about the target concept.

Due to Shannon's seminal work on information theory [Sha48] we know that there exists only one function fulfilling the above criteria. That function is called entropy.

Definition 7.1.1. Let $A = \{a_1, a_2, \dots, a_n\}$ be a set of *independent* random events and p_1, p_2, \dots, p_n their probabilities, $\sum_{i=1}^m p_i = 1$. Then the only continuous function of probabilities fulfilling the above three criteria is called entropy and has the form of:

$$I(p_1, \dots, p_n) = \sum_{i=1}^m p_i \log \frac{1}{p_i} \quad (7.2)$$

Using the equation $\ln x \leq x - 1$ with equality only if $x = 1$ one can show that

$$I(p_1, \dots, p_n) \leq \log(n) \quad (7.3)$$

The upper bound of the equation 7.3 is reached only if $p_i = \frac{1}{n}$. Therefore, informativeness of a question q is maximal if all of its answers a_1, a_2, \dots, a_n have equal probabilities.

7.2 Building Decision Trees

Now let us return to the problem of the learner with the identification of the best question. Let $Q = \{q_1, q_2, \dots, q_m\}$ be the set of questions the learner may ask in the current situation. Her interest is to choose the most valuable question among them using the entropy function.

The informativeness of question q_i we express using (7.1) as:

$$I(q_i) = \sum_{q_i \in Q} P(\text{answer}(q_i) = a_{ij}) \cdot I(\text{answer}(q_i) = a_{ij})$$

where $P(\text{answer}(q_i) = a_{ij})$ is probability of answer a_{ij} on question q_i and $I(\text{answer}(q_i) = a_{ij})$ is informativeness of answer a_{ij} to question q_i . The second value is expressed by (7.2) as:

$$I(\text{answer}(q_i) = a_{ij}) =$$

$$\sum_{c \in C} P(c \equiv c^? \mid \text{answer}(q_i) = a_{ij}) \cdot \log \frac{1}{P(c \equiv c^? \mid \text{answer}(q_i) = a_{ij})}$$

where $P(c \equiv c^? \mid \text{answer}(q_i) = a_{ij})$ is conditional probability that a hypothesis c is the correct target concept, if answer to the question q_i is a_{ij} . Its value is usually estimated as the ratio between number of example instances consistent with the hypothesis c and answer a_{ij} on question q_j versus the number of examples consistent with the answer a_{ij} on question q_j .

Being able to choose the best local question, the decision tree is built recursively using Algorithm 10.

7.3 Wrapping with Attribute Classification

7.3.1 Clustering

Having all background prepared, we can proceed with an explanation of the attribute classifying algorithm. We will demonstrate the algorithm on the following, rather artificial, Web page displayed in Figure 7.1 which allows us to show the interesting properties of this algorithm. Inside of this example Web page, we are interested to extract the city names and the contact links with the black text colour. Assume that in the current situation we have from the user the following positive and negative example instances, marked with green respectively red background.

Algorithm 10 building decision tree

```

# start with the set of all hypotheses
function chooseQuestion( $C$ )

  if  $|C| \geq 2$  :
    for each question  $q_i$  from available questions  $\{q_1, q_2, \dots, q_n\}$ 
      compute  $I(q_i)$ 

    # choose question  $q_{\max}$  with highest informativeness value
     $q_{\max} = \max_{I(q_1), \dots, I(q_n)} \{q_1, q_2, \dots, q_n\}$ 
    ask question  $q_{\max}$  and receive answer  $a_{\max}$ 

    # partition the hypotheses into two subsets  $C^+, C^-$ 
    # according their consistence with the answer
     $C^+ = \text{getConsistentHypotheses}(C, q_{\max}, a_{\max})$ 
     $C^- = C - C^+$ 

    # recursively build the decision tree
    chooseQuestion( $C^+$ )
    chooseQuestion( $C^-$ )

```

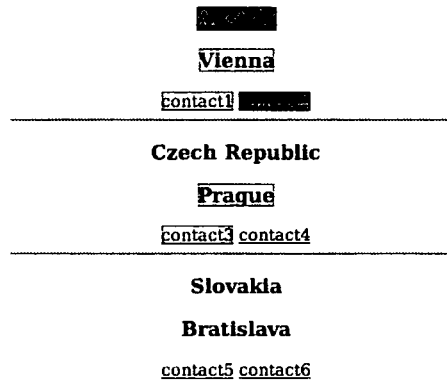


Figure 7.1: Example Web page for attribute classification

During the training process Algorithm 11 proceeds with the following steps:

1. Sort example instances into clusters with respect to similarity of their tree structure.
2. For each cluster build the list of features used for classification. Here the feature list is a collection of possible values for each attribute of the training examples.
3. Build the training dataset.
4. Build the decision tree based classifier.

We have implemented the clustering of example instances according to the similarity of the tree structure using the nearest neighbourhood grouping. As the distance metrics is used the number of different nodes between two DOM trees. During the clustering process a DOM tree becomes pivot of a new cluster, if its distance from pivot DOM trees of the existing clusters oversteps above a chosen threshold. In that case, a new cluster is created and the existing instances are regrouped to their closest cluster.

Additionally, each cluster is divided into blocks. Essentially each DOM element forms its own block, just for some types of the DOM elements, we use the built-in HTML knowledge and merge several elements into a single block. Examples of elements merged into one block are `<table>/<tbody>` or `<p>/<center>`.

The following figure contains a clustering of the instances from our example. Let us note that the cluster might not be just flat tree branches, but also full tree-like structures.

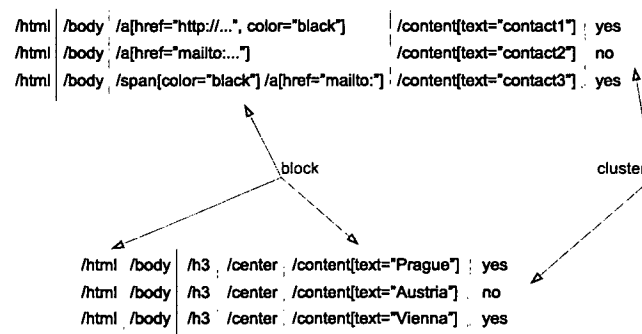


Figure 7.2: Clustering of instances

When using this clustering and attribute classification approach, the wrapper is a set of extraction rules. Each cluster has associated its own extraction rule which is a pair consisting of a CoreXPath expression [GKP02]

Algorithm 11 learning of the wrapper and the extraction process

```

# learn the wrapper
clustering = buildClustering(USER_EXAMPLE_INSTANCES)
for cluster in clustering
    # each cluster contains already the cluster.xpath
    # discovered during the clustering process

    #build the attribute classifier
    cluster.featuresDef = buildClusterFeatures(cluster)
    trainData = buildClusterTrainData(cluster)
    cluster.classifier = buildDecisionTree(featuresDef, trainData)

# extract the matched instances
for cluster in clustering
    instances = evaluateXPath(cluster.xpath, USER_INPUT_DOCUMENT)
    for instance in instances:
        testExample = buildTrainOrTestExample(cluster.featuresDef, instance)
        if cluster.classifier.classify(testExample)=='yes'
            # add 'instance' to matched instances

```

and an attribute classifier. The XPath expression is used to find DOM nodes (instances) inside of the input DOM tree matching the particular tree shape. Subsequently, the attribute classifier is used to sort out the instances with incorrect attributes and properties.

7.3.2 Building Features

Building of a decision requires to know which features are available and what are the possible values for these features. Therefore, before training of the DOM attribute classifier we need to decide what are the interesting features being considered by the classifier.

The list of the features is built from the set of DOM attributes and text node values in the cluster. The same DOM attribute may repeat in several blocks of a cluster with different values, for example the 'bgcolor' attribute in the instance `/table[bgcolor='red']/tr/td[bgcolor='green']`. Therefore, each feature has associated also an index of the block it belongs to. DOM attributes inside of one block are treated as non-repeating that means, their values are unified into a single set.

Clearly, not all HTML attributes have equal semantics. Therefore, it

does not make sense to map all DOM attributes one-to-one into training features. For example, our feature selection considers only attributes that have rendering effect. Because the user highlights the interested instances inside of a rendered Web page, she can not distinguish DOM nodes that differ only with an attribute that does not influent rendering of the DOM node. Therefore, we sorted the HTML attributes into three categories, depending on their rendering effect:

- attributes with no rendering effect
- attributes which affect rendering always the same way, independent of their value
- attributes which affect rendering with each their value differently

Examples of the attributes with different rendering influence are listed in the following table.

influence of rendering:		
none	by presence	by each value
alt	multiple	align
codebase	readonly	border
id	checked	color
method	disabled	style

The list of possible values for a feature is constructed in the following way. Values of a feature constructed from a DOM attribute with rendering influence by presence are 'present' and 'absent'. Values of a feature constructed from a DOM attribute with rendering changed by each value is union of all values of this attribute collected in the current block over all example instances.

Using only the syntactical properties of the DOM trees is quite successful method, but not always sufficient. The beauty of our approach is that we can easily include any other type of features. Therefore, we extended our list for example also with ontology-based features and we include syntactic concepts such as isDate, isYear, isNumber, isCurrency and semantic concepts such as isCity, isCountry, isContinent. List of values for those features is 'yes' and 'no'.

Each feature has two additional values with special meaning. The '!missing' value is used during building of the training data to express that the current example instance does not contain a particular attribute in a block,

while the other instances contain this feature in that block. The value '!other' is used during the extraction process to express that an instance has a particular attribute, but with a value that has not been known during the training process. The Algorithm 12 outlines building of the feature list for a cluster.

For our example, these are the features chosen for the training:

```
BL3_href_val = {http://..., mailto:..., !missing, !other}
BL3_href_protocol = {http, mailto, !missing, !other}
BL3_href_color = {black, !missing, !other}
BL4_text_val = {contact1, contact2, contact3, !missing, !other}

BL5_text_val = {Prague, Austria, Vienna, !missing, !other}
BL5_text_isCity = {yes, no, !missing, !other}
```

7.3.3 Training

Building of the training data is straightforward. We iterate over all instances in the cluster and compute the values of features, as outlined in Algorithms 13 and 14. As indicated already earlier, there is a case possible, when value of a feature can not be computed for some example instance, most probably because the DOM element does not contain the needed DOM attribute. In that case the special value '!missing' is used.

Moreover, a special target feature is added to each training example. The values of that feature are 'yes' or 'no', depending whether the processed example instance is a positive or negative extraction example. Prediction of this feature is then trained by the decision tree classifier.

For our example, the following training set is constructed:

BL3_href_val	BL3_href_protocol	BL3_bgcolor_val	BL4_text_val	extract*
http://...	http	black	contact1	yes
mailto:...	mailto	!missing	contact2	no
http://...	http	black	contact3	yes

BL5_text_val	BL5_text_isCity	extract*
Prague	yes	yes
Austria	no	no
Vienna	yes	yes

Finally, the constructed list of features and the data set are feed into the training algorithm that builds the classifier. Our implementation is illustrated in Figures 7.3-7.7. For implementation of the learning algorithm we

Algorithm 12 building features

```

function buildClusterFeatures(cluster)
    # at the beginning the cluster has no features
    featuresDef = {}

    # build the feature definitions from DOM attributes
    # in the example instances
    for block in cluster.blocks()
        blockIndex = cluster.indexOfBlock(block)
        for domElement in block
            for domAttName in domElement.attributes()
                if not getRenderingEffect(domAttName)==NONE
                    domAttValue = domElem.getAttributeValue(domAttName)
                    addFeature(featuresDef, blockIndex,
                               domAttName, domAttValue)

    return featuresDef

function addFeature(featuresDef, blockIndex, domAttName, domAttValue)
    # construct name for the feature
    featureName = 'BL'+blockIndex+'_'+domAttName+'_val'

    # add the current dom value to the feature domain
    featureDomain = featuresDef.get(featureName)
    if featureDomain==none
        featureDomain = ['!missing', '!other']
        featuresDef.put(featureName, featureDomain)
        if getRenderingEffect(domAttName)==BY_PRESENCE
            featureDomain.add('present')
            featureDomain.add('absent')

    if getRenderingEffect(domAttName)==BY_EACH_VALUE
        featureDomain.add(domAttValue)

    # construct the additional features such as the href_proto, text_isCity
    # using the built-in HTML knowledge or ontologies

```

Algorithm 13 building training data (1)

```

function buildClusterTrainData(cluster)
    #build training example for each instance
    trainData = []
    for instance in cluster.instances():
        example = buildTrainOrTestExample(cluster.featuresDef, instance)
        trainData.add(example)

    return trainData

function buildTrainOrTestExample(featuresDef, instance)
    # at the beginning the training example has no feature values
    trainOrTestExample = {}

    # find value for each feature
    for featureName in featuresDef.attributes()
        blockID, domAttName, valueType = parseFeatureName(featureName)

        domAttValue = none
        blockDOMElements =
            getBlockDomElements(exampleInstance, blockID)
        for domElem in blockDOMElements
            if domElem.hasAttribute(domAttName)
                domAttValue = domElem.getAttributeValue(domAttName)
                break
        featureValue = buildTrainValue(valueType, domAttValue)
        trainOrTestExample.put(featureName, featureValue)

    # set value for the target feature being learned
    if instance.isPositive()
        trainOrTestExample.put('extract*', 'yes')
    else if instance.isNegative()
        trainOrTestExample.put('extract*', 'no')
    else
        # used during the extraction process,
        # this value is left unfilled here and then
        # it is computed using the decision tree
        trainOrTestExample.put('extract*', '?')

    return trainOrTestExample

```

Algorithm 14 building training data (2)

```
function buildTrainValue(valueType, domAttValue)
  if domAttValue==none
    return '!missing'

  switch (valueType)
    case VALUE:
      trainValue = domAttValue
    case PROTOCOL:
      trainValue = 'unknown'
      for proto in ['http','mailto']
        if domAttValue.startsWith(proto)
          trainValue = proto
    case ISCITY:
      if database.isConcept('city', domAttValue)
        trainValue = 'yes'
      else
        trainValue = 'no'
  ....

return trainValue
```

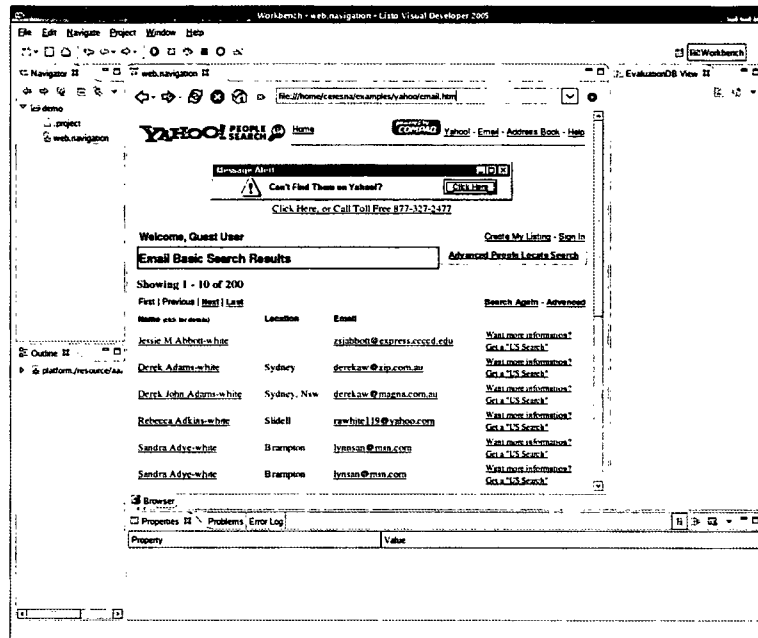


Figure 7.3: Start of the wrapper learning

used the Weka package [WF00] which allowed us to compare performance of various classifying algorithms. Using the ID3 decision tree we get the following classifier:

Cluster1 ID3:
 BL3_bg_color != 'missing' : yes

Cluster2 ID3 :
 BL5_text_isCity = 'yes' : yes

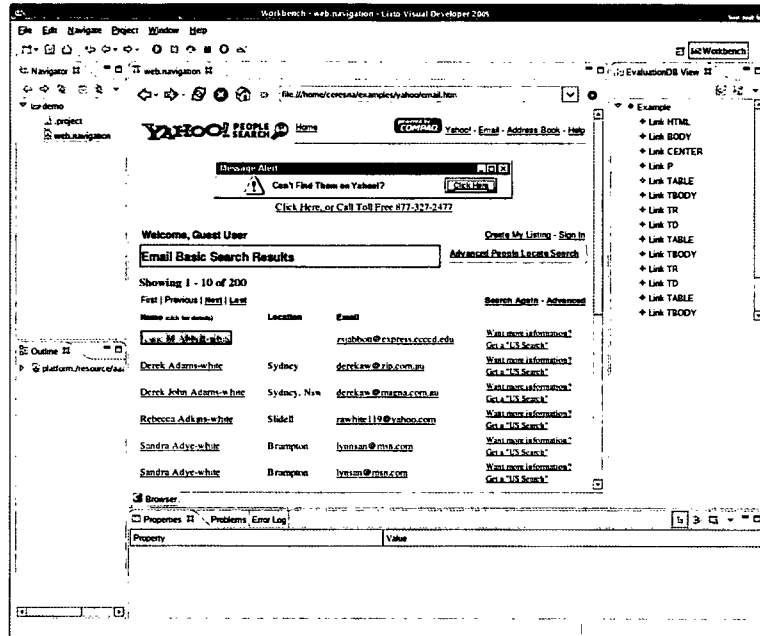


Figure 7.4: Positive example from the user

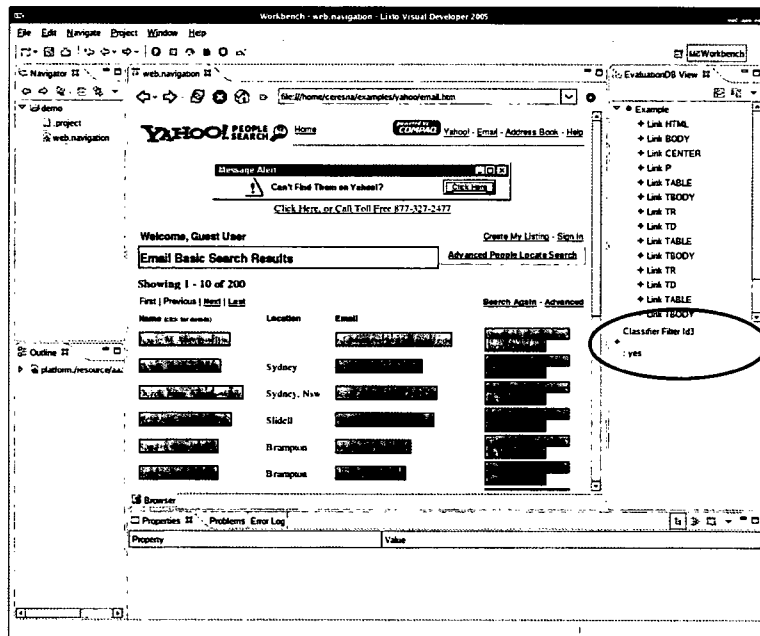
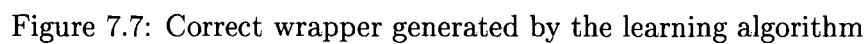
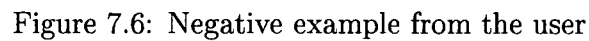


Figure 7.5: Hypothesis generated by the learning algorithm



Conclusion

There are many approaches to Web information extraction. When characterised by their usage of learning techniques, these approaches fall into two distinct categories; fully automated systems using data mining principles and natural language processing and systems whereby information extraction procedures are created by human wrapper designers. The former category of systems reach relatively low precision and recall, but also have low maintenance costs. The latter category carries a much higher maintenance cost, but the precision and recall of the generated wrappers are very high, often reaching almost one hundred per cent.

The successful operation of data mining techniques is achieved under one important assumption: the information is numerously repeating on the Web. Thus, it is not important if the information extraction procedure fails on one particular Web page, as the same information can be still retrieved from plenty of other available Web pages.

An example of applications built with this assumption is the collection of records about corporations and their managers from news sites. The same news item, interview or statement is posted on many news portals and, therefore, failure to extract the data on one particular portal can be salvaged by extracting the information from another portal with a different structure and layout that happens to be more friendly to the information extraction algorithms being used.

However, there are many other applications where the user needs to extract data from just one particular Web site with a concrete structure and layout. These are the cases where wrappers built by human designers are most suitable. Examples of applications built with this approach include collecting items sold in a small set of monitored Web shops, monitoring balance and transactions for an account in Internet banking and collecting job openings on particular portals.

Most previous research in information extraction has made use of data mining techniques to create fully automated systems. In comparison, there has not been much previous effort to apply learning algorithms to acceler-

ate the creation of wrappers by human designers. In this thesis we have attempted to narrow this gap, and have studied active forms of learning and explored their application to HTML wrapper generation.

The first major part of this thesis studied existing active learning theory. At the beginning we presented several learning models related to active forms of learning, including identification from characteristic sets, teaching and query based learning. We concluded that the query based model had the most reasonable assumptions about the learner and the environment, and was closest to the paradigm of interactive wrapper generation. It was therefore chosen as the most suitable model for a practical implementation and we further analysed its properties.

The most common approach of active learning is query based learning using membership and equivalence queries. Therefore we further studied the properties of learning algorithms using membership and equivalence queries. We explored the lower and upper bounds for the number of required queries and presented two generic learning algorithms that work on any hypothesis space - the halving-XEQ algorithm and the standard optimal algorithm.

Next we focused on properties of the hypothesis spaces that measure the hardness of learnability - the teaching, exclusion, approximate fingerprint and Vapnik-Chervonenkis dimensions. Some of these dimensions were also used later when proving negative learnability results.

At the end of the first part we presented the L^* algorithm, a well known positive result of active learning theory. The algorithm is capable of query-based learning of DFA using membership and equivalence queries.

In the second major part of this thesis we applied the knowledge from the active learning theory to interactive wrapper generation.

At the beginning we overviewed existing approaches to the building of HTML wrappers: from hand-coded programs to interactive wrapper induction and data mining techniques. Additionally, we also discussed some of the most known practical implementations.

After the introduction we focused on wrapping techniques that operate on DOM trees and create wrappers by interacting with a human designer. This allowed us to view HTML wrapping as identification of relevant parts in the input DOM tree, such as internal nodes, text node values or attribute values.

We continued by formalising the interaction between the user and the wrapper induction system. We gave a game-theoretic characterisation of the interaction protocol, defined game space and winning conditions. Then we studied the existence of a winning strategy for the wrapper system and proved

that, interestingly, the interactive highlighting of example instances weakens the learning power of the wrapping system. That is, the active interaction inside of a single document, resembling the example based queries, did not lead to tractable learnability results. Tractability was found to come from the possibility to pose the example based queries across a set of similarly structured documents.

Because the XPath language is nowadays a standard for DOM tree navigation and also an integral part of many other XML related standards, we chose it as the formalism for expressing our wrappers. We therefore deeply explored learnability of XPath and its sub-fragments using example-based queries. First we started with the queries used by the minimal adequate learner - equivalence and membership queries. We proved several non-learnability results when using each of these queries separately; even for surprisingly simple XPath fragments in case of equivalence queries.

Interestingly, wrapper induction turned out to be the first known problem where equivalence queries had a natural application and visualisation, but use of the membership queries was artificial and led to obstacles in their visualisation. We therefore proposed a replacement for the membership queries - the tree-prefix queries that did not suffer the visualisation problems. Afterwards we presented a polynomial learning algorithm for several XPath fragments using the combination of equivalence and tree-prefix queries.

The Figure 7.8 summarises the results of query-based learnability for XPath fragments which have been proved in this thesis. Let us note that the negative results for these fragments can not be automatically lifted up to their super-fragments. This is due the fact that the non-learnability proofs, as usual for query-based learning, are representation dependent. In this thesis we used the default representation of the XPath expressions defined in the W3C standard [Rec99].

When we implemented the XPath learning algorithm in practice, we discovered that the proposed learning algorithm is applicable for learning of tree shapes, but has difficulty handling DOM attributes. For HTML wrapper induction we needed to improved the handling of DOM attributes, making use of HTML semantics and text values. The last chapter therefore presents our method for classification of DOM attributes based on decision trees. We showed an algorithm that encodes DOM attributes into a dataset and how a decision tree is built for this dataset. Additionally, we showed how to use the trained attribute classifier for HTML information extraction.

From our experience, we conclude that the learning of wrappers - and information extraction in general - is a challenge combining many areas of artificial intelligence such as machine learning, data mining, statistics, knowl-

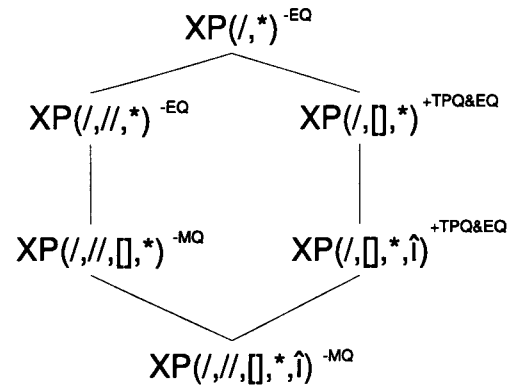


Figure 7.8: Polynomial query-based learnability of XPath fragments

edge representation and natural language processing into a unified theory and implementation leveraging a synergy of all the participating techniques. And in this thesis we have deeply explored some of the pieces in this puzzle.

Bibliography

- [AHHP98] Howard Aizenstein, Tibor Hegedus, Lisa Hellerstein, and Leonard Pitt. Complexity theoretic hardness results for query learning. *Computational Complexity*, 7(1):19–53, 1998.
- [AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 444–454, New York, NY, USA, 1991. ACM Press.
- [Ang78] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, pages 39:337–350, 1978.
- [Ang82] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.
- [Ang87] 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.
- [Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990.
- [Ang04] Dana Angluin. Queries revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.
- [Ant94] Martin Anthony. Probabilistic analysis of learning in artificial neural networks: The PAC model and its variants. Technical Report NC-TR-94-3, London, UK, 1994.
- [BDGW97] Jose Balcazar, Josep Diaz, Ricard Gavaldà, and Osamu Watanabe. Algorithms for learning finite automata from queries:

- A unified view. Technical report, Neural and Computational Learning, NeuroCOLT 8556, 1997.
- [BEHW89] Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
- [BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web information extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.
- [BFK03] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. In *Proceedings of the 9th International Conference on Database Theory*, 2003.
- [BGGM99] Nader H. Bshouty, Paul W. Goldberg, Sally A. Goldman, and H. David Mathias. Exact learning of discretized geometric concepts. *SIAM Journal on Computing*, 28(2):674–699, 1999.
- [Bsh95] Nader H. Bshouty. Exact learning boolean functions via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.
- [CGL97] Jorge Castro, David Guijarro, and Victor Lavin. Learning nearly monotone k -term DNF. In *EuroCOLT*, pages 162–170, 1997.
- [CLN04] Julien Carme, Aurelien Lemay, and Joachim Niehren. Learning node selecting tree transducer from completely annotated examples. In *ICGI*, pages 91–102, 2004.
- [CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [Cor05] Microsoft Corporation. Internet Explorer – Web browser, 1995–2005. Homepage on <http://www.microsoft.com/windows/ie/>.
- [ea04] Andy Quick et al. Jtidy - HTML syntax checker and pretty printer, 2004. Homepage on <http://jtidy.sourceforge.net>.
- [Fou05] Mozilla Foundation. Mozilla – Web browser, 1998–2005. Homepage on <http://www.mozilla.org>.

- [GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.
- [GK95] Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.
- [GKP02] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, 2002.
- [GLR97] David Guijarro, Victor Lavin, and Vijay Raghavan. Learning monotone term decision lists. In *EuroCOLT*, pages 16–26, 1997.
- [GM96] Sally A. Goldman and H. David Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2):255–267, 1996.
- [GMTT05] Remi Gilleron, Patric Marty, Marc Tommasi, and Fabien Torre. Statistical classification for wrapper induction. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
- [GO92] Pedro Garcia and Jose Oncina. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, 1:49–61, 1992.
- [GO93] Pedro Garcia and Jose Oncina. Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informaticos y Computacio, Universidad Politnica de Valencia, 1993. Technical Report DSIC II/47/1993.
- [Goe05] Brian Goetz. Easy screen-scraping with XQuery, 2005. Published on <http://www-128.ibm.com/developerworks/java/library/j-jtp03225.html>.
- [Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
- [Gol78] E.M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

- [Han89] Thomas R. Hancock. Identifying decision trees with equivalence queries. Technical report, Harvard University, 1989.
- [HFAN98] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. Jedi: Extracting and synthesizing information from the Web. In *Conference on Cooperative Information Systems*, pages 32–43, 1998.
- [HGMC⁺97] Joachim Hammer, Hector Garcia-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management for Semistructured Data*, 1997.
- [Hig97] Colin De La Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.
- [HL98] Rainer Himmeroder and Bertram Ludascher. Querying the Web with FLORID. In *Grundlagen von Datenbanken*, pages 47–51, 1998.
- [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.
- [IS93] Y. Ishigami and S. Tani. *The VC-dimensions of finite automata with n states*, volume Lecture Notes on Artificial Intelligence 744, pages 328–341. Springer Verlag, 1993.
- [KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- [KWD97] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
- [Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.
- [Lix05] Lixto Software GmbH. Lixto Visual Wrapper software, 2000–2005. Homepage on <http://www.lixt.com>.

- [LPH00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for Web information sources. In *ICDE*, pages 611–621, 2000.
- [LSPS05] Martin Labsky, Vojtech Svatek, Pavel Praks, and Ondrej Svab. Information extraction from HTML product catalogues: Coupling quantitative and knowledge-based approaches. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
- [Mic04] Sun Microsystems. Java Swing library, 2004. Documentation published on <http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/text/html/package-summary.html>.
- [MJ05] Andrew McCallum and David Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
- [MK05] Matthew Michelson and Craig A. Knoblock. Semantic annotation of unstructured and ungrammatical text. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
- [MS02] Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Symposium on Principles of Database Systems*, pages 65–76, 2002.
- [MT92] Wolfgang Maass and Gyorgy Turan. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9(2-3):107–145, 1992.
- [PW89] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 421–432. ACM Press, 1989.
- [Rec98] W3C Recommendation. Document Object Model (DOM) Level1 specification, 1998. Published on <http://www.w3.org/TR/REC-DOM-Level1>.
- [Rec99] W3C Recommendation. XML Path Language (XPath), 1999. Published on <http://www.w3.org/TR/xpath/>.

- [Rec03] W3C Recommendation. Document Object Model (DOM) Level2 HTML specification, 2003. Published on <http://www.w3.org/TR/DOM-Level-2-HTML>.
- [RS93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.
- [Sak90] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.
- [Sha48] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.
- [Tec05a] Fetch Technologies. Fetch Agent platform, 1999-2005. Homepage on <http://www.fetch.com>.
- [Tec05b] Kapow Technologies. Kapowtech Robosuite software, 2000-2005. Homepage on <http://www.kapowtech.com>.
- [WF00] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

CURRICULUM VITAE



Mgr. Michal Ceresna

Zelena 12

91501 Nove Mesto nad Vahom

Slovakia

Email: ceresna@dbai.tuwien.ac.at

Date of Birth: August 1st, 1978

Education

- **Mgr.** (the Slovak equivalent of B.S. + M.S.), **Computer Science**, Comenius University, Bratislava, Slovak Republic, June 11th, 2002.
- Student of Computer Science at Comenius University, Bratislava, Slovak Republic, (Sept. 1996 – Jun. 2002).
 - Specialization in artificial intelligence and mathematical methods of computer science (cryptography, formal logic).
 - Diploma thesis: “Computational model for analysis of natural language” supervised by Prof. Jan Sefranek, submitted 31st March, 2002.

Non-native natural languages spoken: English, German.

Professional experience

Dec 1999 – present day Technical University Vienna, Institute for Information Systems,
Database and Artificial Intelligence Group.

Research associate.

- Research in the areas of machine learning and information extraction, implementation of a visually guided HTML wrapper induction system.