ICOT Technical Report: TR-724

TR-724

Algorithmic Learning of Formal Languages and Decision Trees

by
Y. Sakakibara (Fujitsu)

January, 1992

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Algorithmic Learning of Formal Languages and Decision Trees

Yasubumi Sakakibara [1]

A dissertation submitted to the Department of Information Science
in partial satisfaction of the requirements for the degree of

Doctor of Science

at the

TOKYO INSTITUTE OF TECHNOLOGY

October 1991

[1] International Institute for Advanced Study of Social Information Science (IIAS-SIS)
FUJITSU LABORATORIES LTD.
140, Miyamoto, Numazu, Shizuoka 410-03, Japan
E-mail : yasu%iias.flab.fujitsu.co.jp@uunet.uu.net

Abstract

We investigate the problem of designing efficient learning algorithms that identify certain concepts from examples. In the first half of this thesis, formal languages are considered as target concepts. We begin with the traditional learning model introduced by Gold, and show that in learning context-free languages, it is useful to have information on the grammatical structure of the unknown language. Indeed we show that while Gold has shown that the class of context-free languages cannot be identified from positive presentations of strings (i.e., strings in the unknown languages), there is a subclass of context-free grammars that can be identified from positive presentations of strings with grammatical structure in polynomial time and can generate all of the context-free languages, where a string with grammatical structure, called *a structured string*, is a string with some parentheses inserted to indicate the shape of the derivation tree.

Next we study a learning situation in which the learning algorithm is allowed to make queries to a teacher on the grammatical structure of the unknown language. We present a learning algorithm that can exactly learn the whole class of context-free grammars in polynomial time by making membership queries for structured strings and structural equivalence queries. We also claim the importance of representations for the problem of learning formal languages. We introduce a new class of representations for formal languages in the framework of Smullyan's elementary formal systems and show that by employing the representations, a larger class of formal languages than context-free languages is efficiently learnable from some reasonable queries.

In the second half, we focus on learning decision trees in the presence of noise. We employ a probabilistic learning model due to Valiant and consider the noise model called *classification noise process* introduced by Angluin and Laird. First we present a polynomial-time algorithm for learning decision lists in the presence of noise, where the decision list is a restricted kind of decision tree introduced by Rivest to represent Boolean functions. Next we extend the algorithm to a polynomial-time algorithm for learning decision trees in the presence of noise. In the course of this study, we develop a technique of building efficient robust learning algorithms, called *noise-tolerant Occam algorithms*, and show that using them, one can construct a polynomial-time algorithm for learning a class of Boolean functions in the presence of noise. Then we present a noise-tolerant Occam algorithm for decision lists and extend it to a noise-tolerant Occam algorithm for decision trees.

Acknowledgements

First I am deeply grateful to my advisor, Masako Takahashi of Tokyo Institute of Technology. She was willing to serve as my thesis supervisor, wade through poorly written drafts, and offer numerous improvements and suggestions. The presentation of this thesis has greatly improved thanks to her comments. I would like to thank other members of my judging committee, Izumi Kimura, Kojiro Kobayashi, Makoto Haraguchi, Osamu Watanabe for their many valuable comments.

Special thanks to Manfred Warmuth who read the initial draft, made useful comments on related papers, and suggested many improvements to the draft. Thanks to many people who are working on Computational Learning Theory research in the world. Especially I would like to thank Robert Schapire for reading the initial draft and pointing out an error in Lemma 6.3 in the draft. Helpful comments were provided by Setsuo Arikawa, Avrim Blum, Jyrki Kivinen, Philip Laird, Takashi Yokomori.

Thanks to the members of IIAS-SIS, Fujitsu Laboratories Ltd.. I would like to thank Tosio Kitagawa, the former president of IIAS-SIS, Hajime Enomoto, Bun-ichi Oguchi, the former directors of IIAS-SIS, Shigeru Sato, the director of IIAS-SIS, Mitsuhiko Toda, Kozo Sugiyama, Kaname Kobayashi, for giving me the opportunity to pursue this work and their warm encouragement. I am also grateful to the members of Computational Learning Group at IIAS-SIS, Yuji Takada, Kunihiko Hiraishi, Masahiro Matsuoka, Hiroki Ishizaka (ICOT), for their enjoyable discussions and friendship.

Last but not least, I would like to thank my parents, my wife, my daughter, and my son for all of the love and support they have given.

The part of this work was done as part of the R&D activities of the Fifth Generation Computer Project, conducted under program set up by MITI of Japan.

Contents

List of Figures

Chapter 1

Introduction

Human beings have an ability to learn new concepts without explicit programming. In the daily life, they often extract the general rules from observed instances. For example, the child can learn his mother language and its grammar from sentences in his parents' daily conversations, and the scientist tries to find the general principle from observations in his experience. In the machine learning literature, this ability is called *inductive inference* or *learning from examples*. Many researchers have worked on this subject to find its mechanisms and make the machine have such ability. The purpose of this thesis is to contribute the progress on the theoretical front for learning from examples.

1.1 Algorithmic Learning

Suppose that you are trying to learn a foreign language (suppose "Russian" for example) with a *very good* teacher. The teacher will try to teach you the Russian grammar by giving many Russian sentences rather than by telling the grammar itself explicitly. Thus you can receive many grammatically correct sentences and incorrect sentences, together with the indication whether they are correct or not. Sometimes you can ask some questions to the teacher like "Is this sentence right?" or "Is this grammar correct?", but you can never ask the question "What is the correct Russian grammar?". The teacher gives you the answer of "Yes" or "No" or a counter-example in the case that the grammar that you conjecture is not correct. As the learning process proceeds, you can distinguish correct sentences from incorrect ones more and more using the grammar that you have learned at that point. With the teacher who never tells "Your grammar is correct" when you find it, it is difficult for you to know which grammar is correct and when you have it so that your learning continues infinitely or you may be satisfied with having an almost correct

grammar. With the teacher who kindly tells "Your grammar is exactly correct", you may finally find the correct grammar.

We are interested in studying such a learning problem formally. The subject of this thesis is called *algorithmic learning theory* or *computational learning theory*. Algorithmic learning theory is the theoretical portion of artificial intelligence that is concerned with machine learning and represents a major theoretical component of machine learning work. To study the subject on machine learning rigorously and precisely, the formal treatment of it is necessary. Rivest suggests in [Slo89] the following seven questions that must be answered to specify a learning problem well.

1. What is being learned ?
2. From what is it learned ?
3. What a priori knowledge does the learner begin with ?
4. How is what is learned represented ?
5. By what method is it learned ?
6. How well is it learned ?
7. How efficiently is it learned ?

In line with these questions, we define a mathematical model for the problem of learning from examples.

What is being learned We assume a domain U on which the target class of concepts to be learned is defined. In this thesis, we examine exclusively abstract mathematical domains. Thus U may be the set of all finite strings over a finite alphabet Σ or the set of all truth assignments over n Boolean variables $\{0,1\}^n$ or the set of all ground atoms over a first order language, i.e. the Herbrand base. We study these domains both because they are interesting in their own right, and in the belief that they are rich enough to form good mathematical models of real world problems.

In this thesis we focus on the learning problem where what is learned is just a subset of U. These subsets of U are called *concepts* and a learning problem for concepts is called *concept learning*. Concepts may be formal languages or Boolean functions or Herbrand models. We initially assume a target class of concepts $\mathcal{C} = \{c_1, c_2, \ldots\}$ of the domain U

and assume that the unknown concept c_* to be learned is chosen from $\mathcal{C}$. A target class of concepts may be the class of regular languages or context-free languages or a class of Boolean functions.

How is what is learned represented Those concepts are typically represented by formal grammars, e.g. finite state automata or context-free grammars, Boolean formulae, decision trees, or logic programs. These representations will be rigorously defined in later sections by using their alphabets to describe representations for concepts.

What a priori knowledge does the learner begin with An initial knowledge that a learning algorithm may have ahead of time is that the unknown concept is chosen from some particular class of concepts. More formally, we design learning algorithms that are only guaranteed to work on the assumption that the unknown concept is chosen from some particular class. For example, we will design a learning algorithm which can work only for the unknown language chosen from the class of context-free languages.

From what is it learned, How well is it learned Within algorithmic learning theory there are three major established formal models for learning from examples or inductive inference. They are the model by Gold [Gol67], the one by Angluin [Ang88], and the one by Valiant [Val84]. Each model provides learning protocol (This corresponds to "From what is it learned ?".) and criterion of the success of learning. (This corresponds to "How well is it learned ?".) We follow these models and their philosophies rather than make a new learning model and work on it because they have already been established well and many interesting results have been given so that we can easily compare our new results with them. These models will be briefly introduced in the following sections and formally defined in the main chapters.

How efficiently is it learned An important aspect of algorithmic learning theory is to analyze the computational cost of a learning algorithm. One criterion of the efficiency of a learning algorithm is whether its running time can be bounded by a polynomial in the relevant parameters. For example, the relevant parameters are the number of states of the minimum deterministic finite automaton for the unknown language, the size of the unknown grammar, the maximum length of counter-examples, the number of Boolean variables, and so on. Thus an efficient learning algorithm is required to use only polynomially bounded examples and computation resources.

By what method is it learned The goal of algorithmic learning theory is to pose interesting learning problems in line with the above questions, and to design algorithms that solve those problems. In the search for polynomial-time learning algorithms for two domains, formal languages (especially context-free languages) and Boolean functions, we will show several results.

1.2 Learning Formal Languages from Queries

In this thesis, we will focus on two major learning problems, *learning formal languages from queries* and *learning decision trees from large data in noisy environment*. A learning problem is a pair (R, C), where R is a class of representations given some representation language, and C is a concept mapping from R to the target class of concepts $\mathcal{C}$, that is, for each r in R, $C(r)$ denotes the concept represented by r. In the first half, we take formal languages as the class $\mathcal{C}$ of learning objects, ask the learning algorithm to return phrase-structure grammars (as R) to generate the language, and consider the situation in which a teacher is available and a learner can make questions about the unknown concept (language) to the teacher.

A problem of inductive inference for formal languages has been long discussed in the context of *grammatical inference* problem. The grammatical inference problem is the problem of learning a "correct" grammar for the unknown language from finite examples in the language. Gold [Gol67] originated this study and introduced the notion of *identification in the limit*. His motivation studying the problem is to construct a formal model of human language acquisition. Identification in the limit views learning as an infinite process and provides a learning model where an infinite sequence of examples of the unknown language is presented to a learning algorithm and the eventual or limiting behavior of the algorithm may be used as the criterion of its success. The criterion requires the algorithm to produce a correct grammar in a finite time during the presentation of an infinite sequence of examples and stay with this grammar from that point on. Although the framework of identification in the limit have succeeded to get many interesting results (Angluin and Smith [AS83] have provided an excellent survey of these studies.), its computational complexity has prohibited itself from further development. We employ a problem setup in which besides given examples, various types of information on the unknown language are available. First we show that in learning context-free languages by using context-free grammars as the representations, it is useful to have information on the grammatical structure of the unknown language.

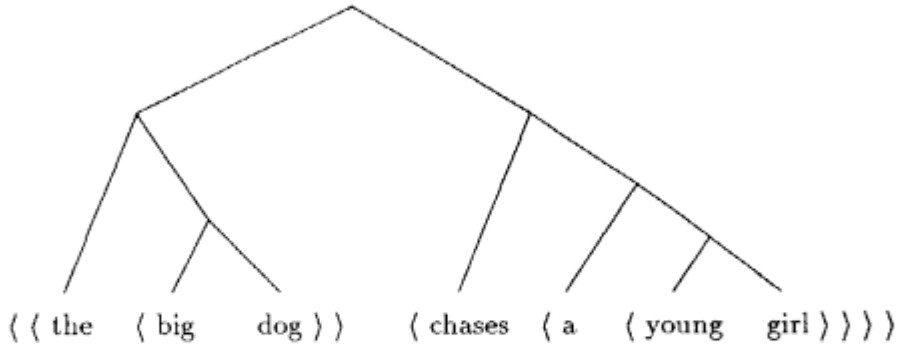

Figure 1.1: A structural description for "the big dog chases a young girl"

In Gold's criterion of identification in the limit for successful learning of a formal language, Gold [Gol67] has shown that there is a fundamental, important difference in what could be learned from *positive* versus *complete* presentations. Examples of formal languages are usually in the form of strings. A string that is in the unknown language is called a *positive example* and a string that is not is a *negative example*. A positive presentation enumerates all and only strings in the unknown language to a learning algorithm, while a complete presentation enumerates all strings, together with the information of whether they belongs to the unknown language. Gold [Gol67] has shown that learning from positive presentations is strictly weaker than learning from complete presentations. Intuitively, an inherent difficulty in trying to learn from positive rather than complete presentations depends on the problem of *overgeneralization*. Gold has shown that any class of languages containing all the finite languages and at least one infinite language cannot be identified in the limit from positive presentations. According to this result, the class of context-free languages (even the class of regular languages) cannot be learned from positive presentations.

To overcome this essential difficulty of learning from positive presentations, we consider the problem of learning from strings with grammatical structure, that is, we assume example presentations in the form of strings with grammatical structure. A string with grammatical structure, called *a structured string* or *a structural description (of string)*, is a string with some parentheses inserted to indicate the shape of the derivation tree of a grammar, or equivalently an unlabelled derivation tree of the grammar, that is, a

derivation tree whose internal nodes have no labels. (See Figure 1.1.) Thus we consider the problem where the domain U is the set of all structured strings, the class of representations R is the class of context-free grammars, and $C(r)$ denotes the set of all unlabelled derivation trees of the grammar $r \in R$, while in learning regular languages, U is the set of all strings, R is the class of finite automata, and for $r \in R$, $C(r)$ denotes the regular language accepted by the automaton r. Hence this problem setting assumes that more than the information of strings in the unknown language is available, that is, information on the grammatical structure of strings is available to the learning algorithm. This setting is also necessary to identify a grammar having the intended structure, that is, structurally equivalent to the unknown grammar. Levy and Joshi [LJ78] have already suggested the possibility of efficient grammatical inferences in terms of structured strings.

The problem is to identify context-free grammars in the limit from positive presentations of structured strings, called *positive structural presentations*, that is, all and only unlabelled derivation trees of the unknown grammar. We show that there is a class of context-free grammars, called *reversible context-free grammars*, which can be identified from positive structural presentations. We also show that the reversible context-free grammar is a normal form for context-free grammars, that is, reversible context-free grammars can generate all of the context-free languages. We present a polynomial-time algorithm which identifies them in the limit from positive structural presentations. This implies that the whole class of context-free languages can be learned efficiently from positive presentations of structured strings of reversible context-free grammars. Note that this does not imply that the whole class of context-free grammars can be learned from positive structural presentations. The class of reversible context-free grammars is a *restricted* and *proper* subclass of context-free grammars.

Next we show that another useful information in the formal language learning is the answer given by making *queries* to a *teacher*. We consider a learning situation in which a teacher is available to answer some queries on the unknown concept. Angluin [Ang88] has devised an elegant formulation of such a teacher and learner paradigm, illustrated in Figure 1.2. In this setup, we can expect the learning algorithm be the *exact learning*, which means the algorithm outputs a correct grammar (i.e., a grammar that exactly generates the unknown language) in a certain finite time. This is no longer a limiting criterion of learning. In the exact learning model, a teacher is a fixed set of *oracles* that can answer specific kinds of queries made by the learning algorithm on the unknown concept c_*. Angluin [Ang88] has considered several types of queries and examined their effects for efficient exact learning. For example, the following two types of queries are

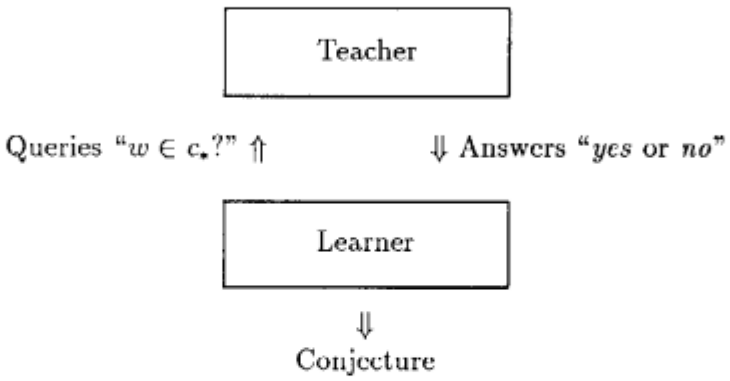

Figure 1.2: A teacher and learner paradigm

typical:

1. *Membership*. The input is an element $w \in U$ and the output is *yes* if $w \in c_*$ and *no* if $w \notin c_*$.

2. *Equivalence*. The input is a representation $r \in R$ and the output is *yes* if $C(r) = c_*$ (i.e., the representation r exactly represents the unknown concept c_*) and *no* otherwise. If the answer is *no*, an element $w \in (C(r) - c_*) \cup (c_* - C(r))$ is returned.

For the equivalence query, the returned element w is called a *counter-example*. A teacher that can answer queries about 1 and 2 is called a *minimally adequate teacher* (MAT, for short), and is considered by Angluin to be a reasonable (not too strong and not too weak) teacher.

A membership query returns one bit of information: whether or not the queried element is a member of the unknown concept c_*. Nevertheless it often plays an important role in efficient exact learning. For example, certain membership queries can avoid a lengthy computation in learning regular languages. The problem of learning regular languages from examples has been studied quite extensively [AS83, Pit89]. Gold [Gol78] has shown a hardness result for learning regular languages that the problem of finding a deterministic finite automaton of a minimum number of states consistent with a given finite set of positive and negative examples is NP-hard. This result is generally interpreted as indicating that even a very simple case of grammatical inference, identifying deterministic finite automata from positive and negative examples, is computationally intractable. Further recently Pitt and Warmuth [PW89] have shown the stronger result

that it is NP-hard to find a deterministic finite automaton of at most $n^{\log n}$ states consistent with a given finite set of positive and negative examples for any constant $\epsilon > 0$, where n is the number of states of a minimum deterministic finite automaton consistent with the given set. Nevertheless Angluin [Ang87b] has shown that regular languages can be learned by an algorithm if one is allowed to make queries above mentioned in time polynomial both in the number of states of the minimum deterministic finite automaton for the unknown language and in the maximum length of counter-examples returned for equivalence queries.

The question of whether there is an analogous result for the whole class of context-free languages is very interesting and important, because the problem itself is theoretically interesting and the class of context-free grammars is an important class from the practical point of view. The syntax of a programming language constitutes a context-free language in principle and many tools for programming languages like a parser or a compiler are designed by means of context-free grammars. Recently Angluin and Kharitonov [AK91] have investigated cryptographic limitations on learning context-free grammars from membership and equivalence queries. They have shown that the problem of learning the whole class of context-free grammars from membership and equivalence queries is computationally as hard as the cryptographic problems such as quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, for which there is currently no known polynomial-time algorithm.

To break this intractability of learning context-free grammars from queries, we again assume that information on the grammatical structure of strings is available to the learning algorithm. That is, we consider the problem where the domain U is the set of all structured strings, the class of representations R is the class of context-free grammars, and $C(r)$ denotes the set of all unlabelled derivation trees of the grammar $r \in R$. In this setup, a membership query for a structured string $w \in U$, called *structural membership query*, returns *yes* if $w \in C(r_*)$ for the unknown grammar r_* and an equivalence query for a grammar $r \in R$, called *structural equivalence query*, returns *yes* if $C(r) = C(r_*)$, that is, r is structurally equivalent to r_*. We show that the whole class of context-free grammars can be learned from structural membership queries and structural equivalence queries in a polynomial computation time. It is known that the set of derivation trees of a context-free grammar constitutes a rational set of trees, where a *rational set* of trees is a set of trees which can be recognized by some tree automaton. Further the set of unlabelled derivation trees of a context-free grammar also constitutes a rational set of trees. Based on these observations, the problem of learning a context-free grammar from

structured strings is reduced to the problem of learning a tree automaton. Then by extending Angluin's efficient learning algorithm for finite automata [Ang87b] to the one for tree automata, we present an efficient learning algorithm for context-free grammars using structural membership and structural equivalence queries. Notice that the learning algorithm can exactly and efficiently learn the whole class of context-free grammars by making structural membership and structural equivalence queries while only a restricted class of context-free grammars can be identified in the limit from positive structural presentations.

We also claim the importance of representations (the class of representations R) for what is learned. In the context of grammatical inference, formal languages are typically represented as regular expressions, finite-state automata, context-free grammars, or phrase-structure grammars. Sometimes these classical grammatical representations are not adequate to design an efficient learning algorithm. We introduce a new class of representations for formal languages in the framework of Smullyan's elementary formal systems [Smu61] for the problem of learning formal languages. The new class of representations is a natural extension of context-free grammars, and the languages defined by the representations lie between context-free languages and context-sensitive languages. We demonstrate a polynomial-time algorithm for learning these representations using some reasonable queries to a teacher. This implies that there exists a larger class of formal languages than the class of context-free languages that is efficiently learnable by using some reasonable queries.

1.3 Learning Decision Trees from Large Data in Noisy Environment

So far we have considered learning problems of formal languages in two kinds of setting. One is an *on-line* setting where an infinite sequence of examples of the unknown language is presented to a learning algorithm and after receiving each example in the sequence, the algorithm updates its conjecture. The other is an *interactive* setting where a teacher is available to answer some queries made by a learner about the unknown language.

In the second half, we consider the learning problem in a *batch* setting from a *large* amount of examples which may contain some noise. A batch setting means a learning situation in which a learning algorithm is not required to produce any output until the end of a sequence of examples, at which time the algorithm outputs a representation as conjecture. The study of learning from large data is motivated by the following obser-

vation. Nowadays by surprising developments of the computer hardware technology, the huge amount of computer resources can be provided for the user. In such circumstances, the user will find some computer software useful if there exists a large amount of data in the problem domain that cannot be dealt with by hand and the software can compute those data very efficiently by using rich computer resources. For example, programs for numerical calculations or the transaction of large data base are such softwares. We would like to consider those problems concerned with large data and to develop some techniques by which one can efficiently extract the general rules of the data

As an instance of such problems of learning from large data, we focus on the learning problem of Boolean functions represented by decision trees. Learning decision trees [BFOS84, Pag90] have been studied long with such a motivation and has the most successful counterpart of practical applications. One famous and practical example of such systems is ID3 by Quinlan [Qui86b]. Decision trees are often used for classification tasks and as the representation of acquired knowledge in a learning system. A classification task is to assign an element of the domain to one of a specified number of disjoint classes. For example, the diagnosis of a medical condition from symptoms is a classification task, in which the classes could be either the various disease states or the possible therapies. ID3 induces such decision trees from examples. Numerous applications based on ID3 have also been investigated. In the decision tree learning problem, concepts are defined on a set of objects in which the objects are described in terms of a set of attribute-value pairs. In the case where each attribute is a Boolean variable (i.e., the value is 0 or 1), the problem of learning decision trees can be formulated as the problem of learning Boolean functions.

When we study the problem of learning from large data, it is practical to assume that the data contain some noise. Therefore we consider the problem of learning decision trees in the presence of noise. Since the presence of noise prevents a learning algorithm from exactly identifying the unknown concept, we employ a probabilistic learning model for this problem.

More precisely, we consider the distribution-independent model introduced by Valiant [Val84], which is called *probably approximately correct learning* (PAC learning, for short). In the PAC learning model, we assume that random samples are drawn independently from the domain U whose probability distribution D may be arbitrary and unknown. A learning algorithm takes a sample as input and produces as output a representation in R. The success of learning is measured by two parameters, the accuracy parameter ϵ and the confidence parameter δ, which are given as inputs to the learning algorithm. We define a notion of the difference between two concepts c and c' with respect to the probability

distribution D as

$$d(c, c') = \sum_{w \in c \oplus c'} \Pr_D(w),$$

where $c \oplus c'$ denotes the symmetric difference of c and c' and $\Pr_D(w)$ denotes the probability of element $w \in U$ with respect to D. The *error* of a representation r with respect to the unknown concept c_* is defined to be $d(C(r), c_*)$. A successful learning algorithm is one that *with high probability* (at least $1 - \delta$) finds a representation *whose error is small* (less than ϵ) and its running time is bounded by a *polynomial* in the relevant parameters (n, $1/\epsilon$, and $1/\delta$). Thus in the PAC learning model, a learning algorithm is not required to learn the unknown concept exactly, but only to find a representation that is a good approximation of it with high probability. This model is naturally applied in our batch setting from a large amount of examples in noisy environment.

The PAC learning model has been applied to the problem of designing and analyzing algorithms for learning Boolean functions and some other functions. Several interesting classes of Boolean functions have been proved to be or not to be polynomially learnable in the PAC learning model [BEHW89, KLPV87, PV88].

The problem of learning decision trees in the PAC learning model has also been studied [Riv87, EH89]. Rivest [Riv87] has introduced a class of representations, called *decision lists*, for representing Boolean functions and shown that k-DL (the class of decision lists with conjunctive clauses of size at most k at each decision) is polynomially learnable in the PAC learning model. The decision list is a useful way of representing Boolean functions, and in fact the decision lists are an important class because k-DL properly includes other well-known techniques for representing Boolean functions such as k-CNF (formulae in conjunctive normal form with at most k literals per term), and k-DNF (formulae in disjunctive normal form with at most k literals per clause). Ehrenfeucht and Haussler [EH89] have introduced the notion of the *rank* of a decision tree and shown that for any fixed r, the class of decision trees of rank at most r, denoted r-DT, is polynomially learnable in the PAC learning model and Rivest's result for decision lists can be interpreted as a special case of their result for rank 1.

However, those works depend strongly on the assumption of perfect, noise-less examples. This assumption is generally unrealistic and in many situations of the real world, we are not always so fortune, our observations will often be afflicted by noise and hence there is always some chance that a noisy example is given to the learning algorithm. Few works have suggested any way to make their learning algorithms noise tolerant and two formal models of noise have been studied so far in the PAC learning model for concept

learning. One is the *malicious error model* initiated by Valiant [Val85] and investigated by Kearns and Li [KL88]:

> Independently for each example, the example is replaced, with some small probability, by an arbitrary example classified perhaps incorrectly.

The goal of this model is to capture the worst possible case of noise process by the adversary. This model is also called *adversarial noise process* in [AL88]. The other is the *classification noise process* introduced by Angluin and Laird [AL88]:

> Independently for each example, the label of the example is reversed with some small probability.

The goal of this model is to study the question of how to compensate for randomly introduced errors, or "noise", in classifying the example data. We consider the classification noise process to study the effect on the polynomial learnability of decision trees. In the classification noise process, we assume that the rate of noise η is strictly less than $1/2$ and there is some information about the noise rate η available to a learning algorithm, namely an upper bound η_b such that $\eta \leq \eta_b < 1/2$.

We begin with the decision lists for the problem of learning decision trees in the presence of classification noise. We present a polynomial-time algorithm for learning k-DL in the presence of classification noise. Next we extend the algorithm to a polynomial-time algorithm for learning r-DT in the presence of classification noise. More precisely, we first develop a technique for building efficient robust learning algorithms in the presence of classification noise. That is the technique of, rather than finding a Boolean function consistent with the given sample, which is the well known technique used in learning in the absence of noise, finding a Boolean function consistent with a large fraction of the sample. We call a polynomial-time algorithm to find such a Boolean function a *noise-tolerant Occam algorithm*, and show that using a noise-tolerant Occam algorithm for a class of Boolean functions, one can construct a polynomial-time algorithm for learning the class in the presence of classification noise. Next we present a noise-tolerant Occam algorithm for k-DL and hence conclude that k-DL is polynomially learnable in the presence of classification noise. This strictly increases the class of Boolean functions that are known to be polynomially learnable in the presence of classification noise: the only example of a class of Boolean functions is k-CNF that has been shown to be polynomially learnable in the presence of classification noise [AL88] and k-DL properly includes k-CNF. Further we extend the noise-tolerant Occam algorithm for k-DL to the one for r-DT and conclude

that r-DT are polynomially learnable in the presence of classification noise. Both results can hold at a noise rate even close to $1/2$.

1.4 Outline of Thesis

The remainder of this thesis is organized as follows.

In Chapter 2, we present a polynomial-time algorithm which identifies in the limit a class of context-free grammars, called *reversible context-free grammars*, from positive structural presentations. We describe a learning model introduced by Gold, *identification in the limit*, and introduce its variant that is a model for identifying a grammar from examples of structured strings. We state the relationship between context-free grammars and tree automata, and show that the problem of learning a context-free grammar from structured strings can be reduced to the problem of learning a tree automaton. Then by extending the efficient algorithm of Angluin [Ang82] which identifies finite automata from positive presentations to the one for tree automata, we present a polynomial-time algorithm which identifies reversible context-free grammars in the limit from positive structural presentations. We also demonstrate several examples to show the learning process of our learning algorithm and to emphasize how successfully and efficiently our learning algorithm identifies primary examples of grammars given in the previous papers for the grammatical inference problem.

In Chapter 3, we describe a learning model due to Angluin, *exact learning*, in which a teacher is available to answer some queries about the unknown concept. In this learning model we show that the whole class of context-free grammars can be learned from structural membership queries and structural equivalence queries in a polynomial computation time. We also demonstrate that this algorithm can be applied to learning a class of logic programs, called *linear monadic logic programs*, from membership and equivalence queries in a polynomial time.

In Chapter 4, we address a problem of exactly learning a larger class of formal languages than the class of context-free languages by using some "reasonable" queries. We introduce a new class of representations for formal languages in the framework of Smullyan's elementary formal systems and demonstrate a polynomial-time algorithm for learning these representations using some "reasonable" queries to a teacher.

In Chapter 5, we describe a learning model due to Valiant, *probably approximately correct learning*, that is a distribution-independent probabilistic model of concept learning from random examples and consider a formal model of noise, *classification noise process*,

in the PAC learning model introduced by Angluin and Laird. We develop a technique of building efficient robust learning algorithms, called *noise-tolerant Occam algorithm*, and show that using a noise-tolerant Occam algorithm for a class of concepts, one can construct a polynomial-time algorithm for learning the class in the presence of classification noise.

In Chapter 6, we define a class of representations for Boolean functions, *decision lists*, introduced by Rivest. We present a noise-tolerant Occam algorithm for k-DL and hence conclude that k-DL is polynomially learnable in the presence of classification noise.

In Chapter 7, we define a class of representations, *decision trees*, and the notion of the *rank* of a decision tree introduced by Ehrenfeucht and Haussler. We extend the noise-tolerant Occam algorithm for decision lists to the one for decision trees and conclude that the class of decision trees of rank at most r is polynomially learnable in the presence of classification noise.

Finally, in Chapter 8 we conclude by summarizing the results presented in this thesis and stating some future research.

Parts of this thesis have appeared else-where. The contents of Chapter 2 appeared in [Sak89, Sak88a]. All sections of Chapter 3 except Section 3.5 appeared in [Sak90c, Sak88b]. Section 3.5 appeared in [Sak90b]. The contents of Chapter 4, 5, 6 appeared in [Sak90e], [Sak90d], [Sak90a], respectively.

Part I

Learning from Queries

Chapter 2

Learning Context-Free Grammars from Positive Structural Examples

In this chapter, we introduce a subclass of context-free grammars, called *reversible context-free grammars*, which is a normal form for context-free grammars and has a good property for learning from positive presentations. We show that the class of reversible context-free grammars can be identified in the limit from (positive) presentations of structured strings and indeed we present an efficient algorithm for it. This implies that the whole class of context-free languages can be learned efficiently from positive presentations of structured strings of reversible context-free grammars.

2.1 Identification in the Limit

Gold's theoretical study [Gol67] of language learning introduces a fundamental concept that is very important in inductive inference : *identification in the limit*. Formally the model of identification in the limit is defined as follows. An infinite sequence of examples of the unknown language L is presented to the learning algorithm M that is attempting to identify L. Examples of formal languages are usually in the form of strings. A *positive presentation* of L is an infinite sequence giving all and only the elements of L. A *complete presentation* of L is an infinite sequence of ordered pairs $\langle w, l \rangle$ from $\Sigma^* \times \{0, 1\}$ such that $l = 1$ if and only if w is a member of L, and such that every element w of Σ^* appears as the first component of some pair in the sequence, where Σ is the alphabet which the unknown language L is defined over. A positive presentation eventually includes every member of L, whereas a complete presentation eventually classifies every element of Σ^* as to its membership in L. After receiving each pair of the presentation, M outputs next conjecture. In the case of formal language learning, the conjecture is usually in the form

of a grammar. If after some finite number of steps in a positive (complete) presentation of L, M guesses a correct grammar for the unknown language L (i.e., a grammar G such that $L(G) = L$) and never changes its guess after this, then M is said to *identify L in the limit from positive (complete) presentations.*

Gold [Gol67] has shown that there is a fundamental, important difference in what could be learned from *positive* versus *complete* presentations and that learning from positive presentations is strictly weaker than learning from complete presentations. Gold has also shown that any class of languages containing all the finite languages and at least one infinite language cannot be identified in the limit from positive presentations. According to this result, the class of context-free languages (even the class of regular sets) cannot be learned from positive presentations. In order to learn formal languages from positive presentations in Gold's criterion of identification in the limit, we must avoid the problem of "overgeneralization", which means guessing a language that is a strict superset of the unknown language. Nevertheless Angluin [Ang80b] has shown various conditions for correct identification of formal languages from positive presentations that avoids overgeneralization and presented nontrivial classes of formal languages that can be learned from positive presentations. Recently Shinohara [Shi90] has shown that larger classes of formal languages can be learned from positive presentations and those classes form a different hierarchy of formal languages from Chomsky hierarchy. However no efficient algorithm has yet been presented for learning those classes. In this chapter, we take a different approach to overcome this essential difficulty of learning from positive presentations. That is learning from structured strings.

2.2 Learning from Structured Strings and Related Works

In learning from structured strings, examples are presented in the form of structured strings, where a structured string is a string with some parentheses inserted to indicate the shape of the derivation tree of a grammar. Thus the learning algorithm can enjoy information on the grammatical structure of the unknown language. The primal concern in this and next chapters is to see how the setting of learning from structured strings will overcome essential difficulties in the problem of learning formal languages.

In this setup, the domain U is the set of all structured strings, the set of representations R is the class of context-free grammars, and $C(r)$ denotes the set of all unlabelled derivation trees of the grammar r in R.

In a practical use of formal language learning (e.g. designing a parser), the above assumption on availabilities of information on the grammatical structure of strings is quite natural. The traditional grammatical inference problem is defined to identify a grammar G from examples of the unknown language L such that G correctly generates the language L, i.e., $L = L(G)$. However for any context-free language L there exist infinitely many grammars G such that $L = L(G)$. Furthermore, those grammars may have different structures. Consider the following example. The grammar G_1 below describes the set of all valid arithmetic expressions involving a variable "v" and the operations of multiplication "$\times$" and addition "$+$".

$$S \to v \mid Av$$
$$A \to v+ \mid v\times \mid v + A \mid v \times A$$
(the grammar G_1)

However the structure assigned by the grammar G_1 to sentences is semantically meaningless. The same language can be specified by the grammar G_2 below in a meaningful manner.

$$S \to E$$
$$E \to F \mid F + E$$
$$F \to v \mid v \times F$$
(the grammar G_2)

Here the phrases are all significant in terms of the rules of arithmetic. Although G_1 and G_2 are equivalent (i.e. $L(G_1) = L(G_2)$), this fact is not very relevant from a practical point of view since it would be unusual to consider such a grammar as G_1. Thus in a practical use like designing a parser, the structure of the learned grammar is more significant because the learned grammar is intended for use in a practical situation entailing the translation or interpretation of sentences in a compiler. However in the framework of the usual grammatical inference, it is impossible to compel to identify such a grammar (e.g. not the grammar G_1 but G_2) which has the correct (intended) structure. To do so, it is necessary for us to assume that information on the grammatical structure of strings is available to the learning algorithm. This hypothesis is in agreement with studies on natural languages by Chomsky in terms of the theory of phrase structure grammars which claim that the availability of the grammatical structure of the unknown language is prerequisite for language description, since there must be a partially semantic basis in syntax acquisition.

We state some related works on learning from structured strings and compare them with our works described below. A related early work is Crespi-Reghizzi's [CR72]. He has described a constructive method for learning context-free grammars from positive samples of structured strings. His algorithm uses a completely different method from ours and learns a different class of context-free grammars. The class of context-free grammars that our algorithm learns can generate all of the context-free languages, while his class of context-free grammars defines a subclass of context-free languages, called *noncounting context-free languages* [CRGM78]. Since our formalization is based on tree automata, one of merits of our method is the simplicity of the theoretical analysis and the ease of understanding the algorithm, whereas the time efficiency of his algorithm [CR72] is still not clear. Levy and Joshi [LJ78] have already suggested a theoretical framework for grammatical inference and the possibility of efficient grammatical inferences in terms of structured strings. Fass [Fas83] has given a theoretical basis for grammatical inference problem of context-free languages from their structured strings based on the theory of Levy and Joshi. However she has not given any algorithmic solution and any analysis of its computational complexity for the problem.

2.3 Basic Definitions of Tree

In this and next sections, we give formal definitions of trees, tree automata, context-free grammars, and some results about their properties and their relations.

Let $\mathbf{N}$ be the set of positive integers and $\mathbf{N}^*$ be the free monoid generated by $\mathbf{N}$. For $y, x \in \mathbf{N}^*$, we write $y \leq x$ if and only if there is a $z \in \mathbf{N}^*$ such that $x = y \cdot z$, and $y < x$ if and only if $y \leq x$ and $y \neq x$.

A *ranked alphabet* V is a finite set of symbols associated with a finite relation called the *rank* relation $r_V \subseteq V \times \mathbf{N}$. V_n denotes the subset $\{f \in V \mid (f,n) \in r_V\}$ of V. Let $m = max\{n \mid V_n \neq \emptyset\}$, i.e., $m = min\{n \mid r_V \subseteq V \times \{0,1,\ldots,n\}\}$. In many cases the symbols in V_n are considered as *function symbols*. We say that a function symbol f has an *arity* n if $f \in V_n$ and a symbol of arity 0 is called a *constant symbol*.

A *tree* over V is a mapping t from Dom_t into V where (1) the domain Dom_t is a finite nonempty subset of $\mathbf{N}^*$; (2) if $x \in Dom_t$ and $y < x$, then $y \in Dom_t$; (3) if $y \cdot i \in Dom_t$ and $i \in \mathbf{N}$, then $y \cdot j \in Dom_t$ for $1 \leq j \leq i$, $j \in \mathbf{N}$; (4) $t(x) \in V_n$, whenever for $i \in \mathbf{N}$, $x \cdot i \in Dom_t$ if and only if $1 \leq i \leq n$. An element of the tree domain Dom_t is called a *node* of t. If $t(x) = A$, then we say that A is the *label* of the node x of t. V^T denotes the set of all trees over V. $|Dom_t|$ denotes the cardinality of Dom_t, that is, the number of

nodes in t.

Intuitively, trees are rooted, directed, connected acyclic finite graphs in which the direct successors of any node are linearly-ordered from left to right. If we consider V as a set of function symbols, the trees over V can be identified with *well-formed terms* over V and written linearly with commas and parentheses. In particular, we identify a single node tree $t : \{\epsilon\} \rightarrow a(\in V_0)$. Within a proof or a theorem, we shall write down only well-formed terms to represent trees. Hence when declaring "let t be of the form $f(t_1, \ldots, t_n) \ldots$" we also declare that f is of arity n

Let t be a tree over V. A node y in t is called a *terminal node* if and only if for all $x \in Dom_t$, $y \not< x$. A node y in t is an *internal node* if and only if y is not a terminal node. The *frontier* of Dom_t, denoted $frontier(Dom_t)$, is the set of all terminal nodes in Dom_t. The *interior* of Dom_t, denoted $interior(Dom_t)$, is $Dom_t - frontier(Dom_t)$. The *depth* of $x \in Dom_t$, denoted $depth(x)$, is the length of x. For a tree t, the *depth* of t is defined as $depth(t) = max\{depth(x) \mid x \in Dom_t\}$. The *size* of t is the number of nodes in t.

Let \$ be a new symbol (i.e., $\$ \notin V$) of rank 0. $V_{\$}^T$ denotes the set of all trees in $(V \cup \{\$\})^T$ which contain exactly one \$-symbol. For trees $s \in V_{\$}^T$ and $t \in (V^T \cup V_{\$}^T)$, we define an operation "#" to replace the terminal node labelled \$ of s with t by

$$s\#t(x) = \begin{cases} s(x) & \text{if } x \in Dom_s \text{ and } s(x) \neq \$, \\ t(y) & \text{if } x = z \cdot y,\ s(z) = \$ \text{ and } y \in Dom_t. \end{cases}$$

For subsets $S \subseteq V_{\$}^T$ and $T \subseteq (V^T \cup V_{\$}^T)$, $S\#T$ is defined to be the set $\{s\#t \mid s \in S \text{ and } t \in T\}$.

Let $t \in V^T$ and $x \in Dom_t$. The *subtree* t/x of t at x is a tree such that $Dom_{t/x} = \{y \mid x \cdot y \in Dom_t\}$ and $t/x(y) = t(x \cdot y)$ for any $y \in Dom_{t/x}$. The *co-subtree* $t\backslash x$ of t at x is a tree in $V_{\$}^T$ such that $Dom_{t\backslash x} = \{y \mid y \in Dom_t \text{ and } x \not< y\}$ and

$$t\backslash x(y) = \begin{cases} t(y) & \text{for } y \in Dom_{t\backslash x} - \{x\}, \\ \$ & \text{for } y = x. \end{cases}$$

Let T be a set of trees. We define the set $Sc(T)$ of co-subtrees of elements of T by

$$Sc(T) = \{t\backslash x \mid t \in T \text{ and } x \in Dom_t\},$$

and the set $Sub(T)$ of subtrees of elements of T by

$$Sub(T) = \{t/x \mid t \in T \text{ and } x \in Dom_t\}.$$

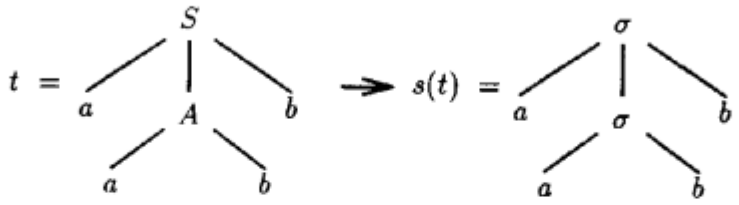

Figure 2.1: The skeletal (or structural) description of a tree t.

A *skeletal alphabet* Sk is a ranked alphabet consisting of only the special symbol σ with the rank relation $r_{Sk} \subseteq \{\sigma\} \times \{1, 2, 3, \ldots, m\}$ for some m. A tree defined over $Sk \cup V_0$ is called a *skeleton*. Let $t \in V^T$. The *skeletal description* of t, denoted $s(t)$, is a skeleton with $Dom_{s(t)} = Dom_t$ such that

$$s(t)(x) = \begin{cases} t(x) & \text{if } x \in frontier(Dom_t), \\ \sigma & \text{if } x \in interior(Dom_t). \end{cases}$$

Let T be a set of trees. The *corresponding skeletal set*, denoted $s(T)$, is $\{s(t) \mid t \in T\}$.

Thus the skeleton is a tree which has a special label σ for the interior nodes. It tells us only the shape and terminal nodes of the tree.

2.4 Tree Automaton and Context-Free Grammar

Definition A *nondeterministic (frontier-to-root) tree automaton* over V is a quadruple $A = (Q, V, \delta, F)$ such that Q is a finite set $(Q \cap V_0 = \emptyset)$, F is a subset of Q, and $\delta = (\delta_1, \delta_2, \ldots, \delta_m)$ consists of the following maps:

$$\delta_k : V_k \times (Q \cup V_0)^k \to 2^Q \qquad (k = 1, 2, \ldots, m).$$

Q is the set of *states*, F is the set of *final states* of A, and δ is the set of *state transition functions* of A. δ is defined on V^T by letting :

$$\delta(f(t_1, \ldots, t_k)) = \begin{cases} \bigcup_{q_1 \in \delta(t_1), \ldots, q_k \in \delta(t_k)} \delta_k(f, q_1, \ldots, q_k) & \text{if } k > 0, \\ \{f\} & \text{if } k = 0. \end{cases}$$

The tree t is *accepted* by A if and only if $\delta(t) \cap F \neq \emptyset$. The set of trees accepted by A is denoted by $T(A) = \{t \in V^T \mid \delta(t) \cap F \neq \phi\}$.

In our definition, the terminal symbols on the frontier are taken as "initial" states. Note that in our definition the tree automaton A cannot accept any tree of depth 0.

Definition A tree automaton is *deterministic* if and only if for each k-tuple $q_1, \ldots, q_k \in Q \cup V_0$ and each symbol $f \in V_k$, there is at most one element in $\delta_k(f, q_1, \ldots, q_k)$.

Note that we allow undefined state transitions in deterministic tree automata.

Let Sk be a skeletal alphabet. A (deterministic or nondeterministic) tree automaton over $Sk \cup V_0$ is called a *skeletal tree automaton.*

Theorem 2.1 ([TW68]) *Nondeterministic tree automata are no more powerful than deterministic tree automata. That is, the set of trees accepted by a nondeterministic tree automaton is accepted by a deterministic tree automaton.*

Note that the deterministic tree automaton may have exponentially many more states than the nondeterministic one accepting the same set.

In this chapter, unless otherwise stated, we will mean a "nondeterministic tree automaton" by simply saying "tree automaton".

Lemma 2.2 (replacement lemma) *Let $A = (Q, V, \delta, F)$ be a deterministic tree automaton. For $s, s' \in V^T$ and $t \in V_{\$}^T$, if $\delta(s) = \delta(s')$, then $\delta(t\#s) = \delta(t\#s')$.*

Proof. It can straightforwardly be proved by induction on the depth of the node labelled \$ in t. □

An *alphabet* is a finite non-empty set of symbols. The set of all finite strings of symbols in an alphabet Σ is denoted Σ^*. The empty string is denoted ϵ. The length of the string w is denoted $|w|$. If X is a finite set, $|X|$ denotes the cardinality of X.

Definition A *phrase-structure grammar* is defined as $G = (N, \Sigma, P, S)$ where N and Σ are alphabets (of *nonterminal symbols* and *terminal symbols* respectively) such that $N \cap \Sigma = \emptyset$, P is a finite set of productions of the form $\alpha \to \beta$, where $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$ and $\beta \in (N \cup \Sigma)^*$, and S is a special nonterminal called the *start symbol.*

If $\alpha \to \beta$ is a production of P, then for any strings γ and δ in $(N \cup \Sigma)^*$, we define $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ and we say that $\gamma\alpha\delta$ *directly derives* $\gamma\beta\delta$ in G. Suppose that $\alpha_1, \alpha_2, \ldots, \alpha_m$ are strings in $(N \cup \Sigma)^*$, $m \geq 1$, and

$$\alpha_1 \Rightarrow \alpha_2,\ \alpha_2 \Rightarrow \alpha_3,\ \ldots,\ \alpha_{m-1} \Rightarrow \alpha_m.$$

Then we say α_1 *derives* α_m in G and write $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_m$. That is, $\stackrel{*}{\Rightarrow}$ is the reflexive and transitive closure of $\Rightarrow$. The finite sequence of strings $\alpha_1, \alpha_2, \ldots, \alpha_m$ is said to be a *derivation* of α_m from α_1 in G and is also written

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \cdots \Rightarrow \alpha_m.$$

The *language generated* by G, denoted $L(G)$, is $\{w \mid w \text{ is in } \Sigma^* \text{ and } S \stackrel{*}{\Rightarrow} w\}$.

A phrase-structure grammar $G = (N, \Sigma, P, S)$ is *context-sensitive* if each production is of the form $\alpha A \gamma \rightarrow \alpha \beta \gamma$, where $A \in N$, $\alpha, \gamma \in (N \cup \Sigma)^*$, and $\beta \in (N \cup \Sigma)^+$. A phrase-structure grammar $G = (N, \Sigma, P, S)$ is *context-free* if each production is of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$.

Two context-free grammars G and G' are said to be *equivalent* if and only if $L(G) = L(G')$. Two context-free grammars $G = (N, \Sigma, P, S)$ and $G' = (N', \Sigma, P', S')$ are said to be *isomorphic*, that is, differ only by the names of nonterminals, if and only if there exists a bijection φ of N onto N' such that $\varphi(S) = S'$ and for every $A, B_1, \ldots, B_k \in N \cup \Sigma$, $A \rightarrow B_1 \cdots B_k \in P$ if and only if $\varphi(A) \rightarrow B_1' \cdots B_k' \in P'$ where $B_i' = \varphi(B_i)$ if $B_i \in N$ and $B_i' = B_i$ if $B_i \in \Sigma$ for $1 \leq i \leq k$.

Definition Let $G = (N, \Sigma, P, S)$ be a context-free grammar. For A in $N \cup \Sigma$, the set $D_A(G)$ of trees over $N \cup \Sigma$ is recursively defined as :

$$D_A(G) = \begin{cases} \{a\} & \text{if } A = a \in \Sigma, \\ \{A(t_1, \ldots, t_k) \mid A \rightarrow B_1 \cdots B_k,\ t_i \in D_{B_i}(G)\ (1 \leq i \leq k)\} & \text{if } A \in N. \end{cases}$$

A tree in $D_A(G)$ is called a *derivation tree* of G from A.

For the set $D_S(G)$ of derivation trees of G from the start symbol S, the S-subscript will be deleted. Then $s(D(G))$ is the set of skeletal descriptions of derivation trees of G.

Two context-free grammars G_1 and G_2 are said to be *structurally equivalent* if $s(D(G_1)) = s(D(G_2))$. Note that if G_1 and G_2 are structurally equivalent, they are equivalent, too.

Given a context-free grammar G, we can get the skeletal alphabet which $s(D(G))$ is defined over. Let r be the set of the lengths of the right-hand sides of all the productions in G. Then the skeletal alphabet Sk for $s(D(G))$ consists of the singleton set $\{\sigma\}$ with $r_{Sk} = \{\sigma\} \times r$. A *structured string* is a skeleton over $Sk \cup \Sigma$. For a skeletal tree automaton A over $Sk \cup \Sigma$, we denote it by $A = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ without confusion.

Next we show two important theorems which connect a context-free grammar with a tree automaton. By a coding of the derivation process of a context-free grammar in the formalism of a tree automaton, we can get the following result.

Definition Let $G = (N, \Sigma, P, S)$ be a context-free grammar. The corresponding skeletal tree automaton $A(G) = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ is defined as follows:

$$\begin{aligned} Q &= N, \\ F &= \{S\}, \\ \delta_k(\sigma, B_1, \ldots, B_k) &= \{A \in N \mid A \to B_1 \cdots B_k \text{ is in } P\}. \end{aligned}$$

Theorem 2.3 *Let G be a context-free grammar. Then $T(A(G)) = s(D(G))$. That is, the set of skeletons accepted by $A(G)$ is equal to the set of skeletal descriptions of derivation trees of G.*

Proof. We prove that $s \in s(D_A(G))$ if and only if $\delta(s) \ni A$ for any skeleton s and $A \in N \cup \Sigma$. Then it immediately follows that $s \in s(D(G))$ if and only if $\delta(s) \ni S$. Hence $s(D(G)) = T(A(G))$.

We prove it by induction on the depth of s. Suppose first that the depth of s is 0, i.e. $s = a \in \Sigma$. By the definition of $D_A(G)$ and $A(G)$, $a \in D_A(G)$ if and only if $A = a$ if and only if $\delta(a) = \{a\} \ni A$. Hence $a \in s(D_A(G))$ if and only if $\delta(a) \ni A$.

Next suppose that the result holds for all skeletons with depth at most h. Let s be a skeleton of depth $h+1$, so that $s = \sigma(u_1, \ldots, u_k)$ for some skeletons $u_1, \ldots, u_k$ with depth at most h. Assume that $u_i \in s(D_{B_i}(G))$ for $1 \le i \le k$. Then

$\sigma(u_1, \ldots, u_k) \in s(D_A(G))$
if and only if there is the production of the form $A \to B_1 \cdots B_k$ in P, by the definition of $D_A(G)$,
if and only if $\delta_k(\sigma, B_1, \ldots, B_k) \ni A$, by the definition of $A(G)$,
if and only if $\delta_k(\sigma, B_1, \ldots, B_k) \ni A$ and $B_1 \in \delta(u_1), \ldots, B_k \in \delta(u_k)$, by the induction hypothesis,
if and only if $\delta(\sigma(u_1, \ldots, u_k)) \ni A$.

This completes the induction and the proof of the above theorem. □

Conversely, by a coding of the recognizing process of a tree automaton in the formalism of a context-free grammar, we can get the following result.

Definition Let $A = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ be a deterministic skeletal tree automaton. The corresponding context-free grammar $G(A) = (N, \Sigma, P, S)$ is defined as follows:

$$N = Q \cup \{S\},$$

$$\begin{aligned} P = \{&\delta_k(\sigma, q_1, \ldots, q_k) \to q_1 \cdots q_k \\ &\mid q_1, \ldots, q_k \in Q \cup \Sigma \text{ and } \delta_k(\sigma, q_1, \ldots, q_k) \text{ is defined}\} \\ \cup\{&S \to q_1 \cdots q_k \mid \delta_k(\sigma, q_1, \ldots, q_k) \in F\}. \end{aligned}$$

Theorem 2.4 *Let $A = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ be a deterministic skeletal tree automaton. Then $s(D(G(A))) = T(A)$. That is, the set of skeletal descriptions of derivation trees of $G(A)$ is equal to the set of skeletons accepted by A.*

Proof. First we prove that (i) $\delta(s) = q$ if and only if $s \in s(D_q(G(A)))$ for $q \in Q \cup \Sigma$. We prove it by induction on the depth of s. Suppose first that the depth of s is 0, i.e. $s = a \in \Sigma$. By the definition of $G(A)$ and $D_A(G)$, $\delta(a) = q$ if and only if $q = a$ if and only if $a \in D_q(G(A))$. Hence $\delta(a) = q$ if and only if $a \in s(D_q(G(A)))$.

Next suppose that the result holds for all skeletons with depth at most h. Let s be a skeleton of depth $h+1$, so that $s = \sigma(u_1, \ldots, u_k)$ for some skeletons $u_1, \ldots, u_k$ with depth at most h. Assume that $\delta(u_i) = q_i$ for $1 \leq i \leq k$. Then

$\delta(\sigma(u_1, \ldots, u_k)) = q$
if and only if $\delta_k(\sigma, \delta(u_1), \ldots, \delta(u_k)) = q$
if and only if $\delta_k(\sigma, q_1, \ldots, q_k) = q$
if and only if there is the production of the form $q \to q_1 \cdots q_k$ in $G(A)$, by the definition of $G(A)$,
if and only if $q \to q_1 \cdots q_k$ in $G(A)$ and $u_1 \in s(D_{q_1}(G(A))), \ldots, u_k \in s(D_{q_k}(G(A)))$, by the induction hypothesis,
if and only if $\sigma(u_1, \ldots, u_k) \in s(D_q(G(A)))$, by the definition of $D_A(G)$.

This completes the induction and the proof of (i).

Secondly we prove that (ii) $s \in s(D_S(G(A)))$ if and only if $s \in s(D_q(G(A)))$ for some $q \in F$. Let s be a skeleton of the form $\sigma(u_1, \ldots, u_k)$ for some skeletons $u_1, \ldots, u_k$. If $s \in s(D_S(G(A)))$, then since if $u_i \in s(D_{q_i}(G(A)))$, then $q_i = \delta(s_i)$ for $1 \leq i \leq k$ by (i), there is the production of the form $S \to \delta(u_1) \cdots \delta(u_k)$ in $G(A)$ and $\delta_k(\sigma, \delta(u_1), \ldots, \delta(u_k)) \in F$ by the definition of $G(A)$. Then $\delta(\sigma(u_1, \ldots, u_k)) \in F$ and so $\delta(s) \in F$. Hence by (i), $s \in s(D_q(G(A)))$ for some $q \in F$.

Conversely if $s \in s(D_q(G(A)))$ for some $q \in F$, then $\delta(s) = \delta_k(\sigma, \delta(u_1), \ldots, \delta(u_k)) \in F$ by (i). By the definition of $G(A)$, there is the production of the form $S \to \delta(u_1) \cdots \delta(u_k)$ in $G(A)$. Since $u_i \in s(D_{\delta(u_i)}(G(A)))$ for $1 \leq i \leq k$ by (i), $\delta(u_1, \ldots, u_k) \in s(D_S(G(A)))$. Hence $s \in s(D_S(G(A)))$.

Lastly it immediately follows from (i) and (ii) that $\delta(s) \in F$ if and only if $s \in s(D(G(A)))$. Hence $T(A) = s(D(G(A)))$. □

Therefore the problem of learning a context-free grammar from structured strings can be reduced to the problem of learning a tree automaton.

2.5 Basic Idea of Learning

Angluin [Ang82] has introduced a subclass of finite automata and the corresponding class of languages, called *zero-reversible automata* and *zero-reversible languages* respectively, that can be identified in the limit from positive presentations. She has also presented an efficient algorithm for it.

A zero-reversible automaton is a deterministic finite automaton with at most one final state such that no two arrows entering any state are labelled with the same symbol. The class of zero-reversible languages defined by zero-reversible automata is a proper subclass of regular languages.

We extend this notion to skeletal tree automata, and further to context-free grammars. We call such a skeletal tree automaton a *reversible skeletal tree automaton* and the corresponding context-free grammar a *reversible context-free grammar*, which will be formally defined in the next section. The class of reversible context-free grammars is a proper subclass of context-free grammars, but can generate all of the context-free languages. In Section 2.7, we will extend Angluin's efficient algorithm for identifying zero-reversible automata to the one for identifying reversible skeletal tree automaton in the limit from positive presentations and hence for identifying reversible context-free grammars in the limit from positive presentations of structured strings.

In this section, we give an informal example of learning process by our algorithm. A context-free grammar $G = (N, \Sigma, P, S)$ is said to be *reversible* if (1) $A \to \alpha$ and $B \to \alpha$ in P implies $A = B$ and (2) $A \to \alpha B \beta$ and $A \to \alpha C \beta$ in P implies $B = C$. The learning algorithm takes as input a finite set Sa of structured strings. First the algorithm constructs a context-free grammar G_0 that precisely generates the set Sa, that is, $s(D(G_0)) = Sa$. Next the algorithm merges nonterminals and generalizes it to get a reversible context-free grammar G such that $s(D(G)) = \min\{s(D(G')) \mid Sa \subseteq s(D(G')), \text{and } G' \text{ is a reversible context-free grammar}\}$.

Suppose that the following set Sa of structured strings is given to the learning algorithm.

$$Sa = \{ \langle \langle a\ b \rangle \langle c \rangle \rangle, \langle \langle a \langle a\ b \rangle\ b \rangle \langle c \langle c \rangle \rangle \rangle, \langle \langle a\ b \rangle \langle c \langle c \rangle \rangle \rangle \}.$$

The algorithm constructs the following grammar G_0 such that $s(D(G_0)) = Sa$:

$$\begin{aligned}
&S \to A\ B\\
&A \to a\ b\\
&B \to c\\
&S \to C\ D\\
&C \to a\ C'\ b\\
&C' \to a\ b\\
&D \to c\ D'\\
&D' \to c\\
&S \to E\ F\\
&E \to a\ b\\
&F \to c\ F'\\
&F' \to c.
\end{aligned}$$

Then nonterminals A, C', and E in G_0 are merged and nonterminals B, D', and F' in G_0 are merged to satisfy the condition (1).

$$\begin{aligned}
&S \to A\ B\\
&A \to a\ b\\
&B \to c\\
&S \to C\ D\\
&C \to a\ A\ b\\
&D \to c\ B\\
&S \to A\ F\\
&F \to c\ B.
\end{aligned}$$

Again to satisfy the condition (1), nonterminals D and F are merged.

$$\begin{aligned}
&S \to A\ B\\
&A \to a\ b\\
&B \to c\\
&S \to C\ D\\
&C \to a\ A\ b\\
&D \to c\ B\\
&S \to A\ D.
\end{aligned}$$

To satisfy the condition (2), nonterminals B and D are merged and nonterminals A and C are merged. Finally the algorithm outputs the following reversible context-free grammar:

$$S \rightarrow A\ B$$
$$A \rightarrow a\ b$$
$$A \rightarrow a\ A\ b$$
$$B \rightarrow c$$
$$B \rightarrow c\ B.$$

2.6 Reversible Context-Free Grammars

We introduce a subclass of skeletal tree automata, called *reversible skeletal tree automata*, and the corresponding subclass of context-free grammars, called *reversible context-free grammars*, and show that it is a normal form for context-free grammars (i.e., any context-free grammar has an equivalent reversible context-free grammar) and has a good property for learning from positive presentations.

Definition A deterministic skeletal tree automaton $A = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ is *reset-free* if and only if for no two distinct states q_1 and q_2 in Q do there exist a state $q_3 \in Q$, positive integers k, i $(1 \leq i \leq k)$, and $k-1$-tuple $u_1, \ldots, u_{k-1} \in Q \cup \Sigma$ such that $\delta_k(\sigma, u_1, \ldots, u_{i-1}, q_1, u_i, \ldots, u_{k-1}) = q_3 = \delta_k(\sigma, u_1, \ldots, u_{i-1}, q_2, u_i, \ldots, u_{k-1})$. The skeletal tree automaton is said to be *reversible* if and only if it is deterministic, has at most one final state, and is reset-free.

The idea of the reversible skeletal tree automaton comes from the "reversible automaton" and the "reversible languages" in [Ang82]. Basically, the reversible skeletal tree automaton is the extension of the "zero-reversible automaton" in [Ang82].

Definition A context-free grammar $G = (N, \Sigma, P, S)$ is said to be *invertible* if and only if $A \rightarrow \alpha$ and $B \rightarrow \alpha$ in P implies $A = B$.

The motivation for studying invertible grammars comes from the theory of bottom-up parsing. Bottom-up parsing consists of (1) successively finding phrases and (2) reducing them to their parents. In a certain sense, each half of this process can be made simple but only at the expense of the other. Invertible grammars allow reduction decisions to be made simply. Invertible grammars have unique righthand sides of the productions so that the reduction phase of parsing becomes a matter of table lookup. Gray and Harrison [GH72] proved that for any context-free language L, there is an invertible grammar G such that $L(G) = L$. That is, the invertible grammar is a normal form for context-free grammars.

Theorem 2.5 (Gray & Harrison [GH72]) *For each context-free grammar G there is an invertible context-free grammar G' so that $L(G') = L(G)$. Moreover, if G is ϵ-free then so is G'.*

Note that this result is essentially the same one as the determinization of a frontier-to-root tree automaton, and suffers the same exponential blowup in the number of nonterminals in the grammer. It however preserves structural equivalence. (This needs a slight modification of the definition for context-free grammars. See also [McN67].)

Definition A context-free grammar $G = (N, \Sigma, P, S)$ is *reset-free* if and only if for any two nonterminals B, C and $\alpha, \beta \in (N \cup \Sigma)^*$, $A \rightarrow \alpha B \beta$ and $A \rightarrow \alpha C \beta$ in P implies $B = C$.

Definition A context-free grammar G is said to be *reversible* if and only if G is invertible and reset-free. A context-free language L is defined to be *reversible* if and only if there exists a reversible context-free grammar G such that $L = L(G)$.

Example 2.1 The following is a reversible context-free grammar for a subset of the syntax for a programming language Pascal.

$$
\begin{aligned}
&\mathit{Statement} \rightarrow \mathit{Ident} := \mathit{Expression}\\
&\mathit{Statement} \rightarrow \text{while } \mathit{Condition} \text{ do } \mathit{Statement}\\
&\mathit{Statement} \rightarrow \text{if } \mathit{Condition} \text{ then } \mathit{Statement}\\
&\mathit{Statement} \rightarrow \text{begin } \mathit{Statementlist} \text{ end}\\
&\mathit{Statementlist} \rightarrow \mathit{Statement} \ ; \ \mathit{Statementlist}\\
&\mathit{Statementlist} \rightarrow \mathit{Statement}\\
&\mathit{Condition} \rightarrow \mathit{Expression} > \mathit{Expression}\\
&\mathit{Expression} \rightarrow \mathit{Term} + \mathit{Expression}\\
&\mathit{Expression} \rightarrow \mathit{Term}\\
&\mathit{Term} \rightarrow \mathit{Factor}\\
&\mathit{Term} \rightarrow \mathit{Factor} \times \mathit{Term}\\
&\mathit{Factor} \rightarrow \mathit{Ident}\\
&\mathit{Factor} \rightarrow (\ \mathit{Expression}\ ).
\end{aligned}
$$

Even if we add the production "$\mathit{Expression} \rightarrow \mathit{Term} - \mathit{Expression}$" or "$\mathit{Term} \rightarrow \mathit{Factor} \ / \ \mathit{Term}$" to the above grammar, it is still reversible. However if the production "$\mathit{Factor} \rightarrow \mathit{Number}$" or "$\mathit{Factor} \rightarrow \mathit{Function}$" is added, it is no longer reversible.

Definition Let $A = (Q, \{\sigma\} \cup \Sigma, \delta, \{q_f\})$ be a reversible skeletal tree automaton for a skeletal set. The corresponding context-free grammar $G'(A) = (N, \Sigma, P, S)$ is defined as

follows.

$$\begin{aligned} N &= Q, \\ S &= q_f, \\ P &= \{\delta_k(\sigma, q_1, \ldots, q_k) \to q_1 \cdots q_k \\ &\qquad \mid q_1, \ldots, q_k \in Q \cup \Sigma \text{ and } \delta_k(\sigma, q_1, \ldots, q_k) \text{ is defined}\}. \end{aligned}$$

By the definitions of $A(G)$ and $G'(A)$, we can observe the following.

Theorem 2.6 *If G is a reversible context-free grammar, then $A(G)$ is a reversible skeletal tree automaton such that $T(A(G)) = s(D(G))$. Conversely if A is a reversible skeletal tree automaton, then $G'(A)$ is a reversible context-free grammar such that $s(D(G'(A)))$ $= T(A)$.*

Therefore the problem of learning reversible context-free grammars from structured strings is reduced to the problem of learning reversible skeletal tree automata.

Next we show some important theorems about the normal form property of reversible context-free grammars. We give two transformations of a context-free grammar into an equivalent reversible context-free grammar. The first transformation adds a number of copies of a nonterminal that derives only ϵ to the right-hand side of each production to make each production unique.

Theorem 2.7 *For any context-free language L, there is a reversible context-free grammar G such that $L(G) = L$.*

Proof. First we assume that L does not contain the empty string. Let $G' = (N', \Sigma, P', S')$ be an ϵ-free context-free grammar in Chomsky normal form (see [HU79] for the definition of *Chomsky normal form*) such that $L(G') = L$. Index the productions in P' by the integers $1, 2, \ldots, |P'|$. Let the index of $A \to \alpha \in P'$ be denoted $I(A \to \alpha)$. Let R be a new nonterminal symbol not in N' and construct $G = (N, \Sigma, P, S')$ as follows:

$$\begin{aligned} N &= N' \cup \{R\}, \\ P &= \{A \to \alpha R^i \mid A \to \alpha \in P' \text{ and } i = I(A \to \alpha)\} \\ &\quad \cup \{R \to \epsilon\} \end{aligned}$$

Clearly G is reversible and $L(G) = L$.

If $\epsilon \in L$, let $L' = L - \{\epsilon\}$ and $G' = (N, \Sigma, P, S')$ be the reversible context-free grammar constructed in the above way for L'. Then $G = (N \cup \{S\}, \Sigma, P \cup \{S \to S', S \to RR\}, S)$ is reversible and $L(G) = L$. □

The trivialization occurs in the previous proof because ϵ-productions are used to encode the index of the production. We prefer to allow ϵ-production only if absolutely necessary and prefer ϵ-free reversible context-free grammars if possible because ϵ-free grammars are important in practical applications such as efficient parsing. Unfortunately there are context-free languages for which there do not exist any ϵ-free reversible context-free grammar. An example of such a language is:

$$\{a^i \mid i \geq 1\} \cup \{b^j \mid j \geq 1\} \cup \{c\}$$

However if a context-free language does not contain the empty string and any terminal string of length one, then there is an ϵ-free reversible context-free grammar which generates the language. The second transformation achieves this result by means of chain rules with new nonterminals.

Theorem 2.8 *Let L be any context-free language in which all strings are of length at least two. Then there is an ϵ-free reversible context-free grammar G such that $L(G) = L$.*

Proof. We construct the reversible context-free grammar $G = (N, \Sigma, P, S)$ in the following steps.

First by the proof of Theorem 2.5 in [GH72], there is an invertible context-free grammar $G' = (N', \Sigma, P', S')$ such that $L(G') = L$ and each production in P' is of the form

1. $A \to BC$ with $A, B, C \in N' - \{S'\}$ or
2. $A \to a$ with $A \in N' - \{S'\}$ and $a \in \Sigma$ or
3. $S' \to A$ with $A \in N' - \{S'\}$.

Since all strings in L are of length at least two, P' has no production of the form $A \to a$ for $A \in N' - \{S'\}$ and $a \in \Sigma$ such that $S' \to A \in P'$.

Next for all productions in P', we make them reset-free while preserving invertibility. P is defined as follows:

1. For each $A \in N' - \{S'\}$, let

$$\{A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_n\}$$

be the set of all productions in P' whose left-hand side is A. P contains the set of productions

$$\{A \to \alpha_1, A \to X_{A_1}, X_{A_1} \to \alpha_2, X_{A_1} \to X_{A_2}, \ldots, X_{A_{n-1}} \to \alpha_n\},$$

where $X_{A_1}, X_{A_2}, \ldots, X_{A_{n-1}}$ are new distinct nonterminal symbols.

2. Let

$$I = \{S \to BY_C \mid A \to BC \in P' \text{ and } S' \to A \in P'\},$$

where S and each Y_C is new distinct nonterminal symbols. Let us denote $\{S \to \beta_i \mid 1 \leq i \leq n\}$ for the set I. P contains the set of productions

$$\{S \to \beta_1, S \to X_{S_1}, X_{S_1} \to \beta_2, X_{S_1} \to X_{S_2}, \ldots, X_{S_{n-1}} \to \beta_n\},$$

where $X_{S_1}, X_{S_2}, \ldots, X_{S_{n-1}}$ are new distinct nonterminal symbols.

3. P contains the set of productions $\{Y_C \to C \mid C \in N' - \{S'\}\}$.

Let $G = (N, \Sigma, P, S)$, where $N = (N' - \{S'\}) \cup \{X_{A_1}, X_{A_2}, \ldots, X_{A_{n-1}} \mid A \in N' - \{S'\}\} \cup \{Y_C \mid C \in N' - \{S'\}\} \cup \{X_{S_1}, X_{S_2}, \ldots, X_{S_{n-1}}\} \cup \{S\}$.

Now we begin the proof that G is reversible, ϵ-free, and $L(G) = L(G')$.

Claim 1 G is reversible.

Proof. Since G' is invertible, each production of the form $A \to BC$, $A \to a$, $X_{A_i} \to BC$ or $X_{A_i} \to a$ for $A, B, C \in N'$ and $a \in \Sigma$ in P has the unique righthand side by the construction 1 of P, and each production of the form $S \to BY_C$ or $X_{S_i} \to BY_C$ in P also has the unique righthand side by the construction 2 of P. By the constructions 1, 2 and 3 of P, each production of the form $A \to B$ for $A, B \in N$ in P has the unique righthand side. Hence G is invertible.

For each $A \in N$, by the constructions 1, 2 and 3 of P, there are at most two productions whose left-hand side is A in P. Furthermore they have different forms, that is, $A \to BC$ or $A \to a$ and $A \to B$, where $A, B, C \in N$ and $a \in \Sigma$. Hence G is reset-free. Therefore G is reversible.

Claim 2 $L(G') \subseteq L(G)$.

Proof. By the construction 1 of P, for each $A \in N' - \{S'\}$, $A \to \alpha$ in G' implies $A \stackrel{*}{\Rightarrow} \alpha$ in G. By the construction 2 and 3 of P, $S' \Rightarrow A \Rightarrow BC$ in G' implies $S \stackrel{*}{\Rightarrow} BY_C \Rightarrow BC$ in G. Hence for each $w \in \Sigma^*$, $S' \stackrel{*}{\Rightarrow} w$ in G' implies $S \stackrel{*}{\Rightarrow} w$ in G.

Claim 3 $L(G') \supseteq L(G)$.

Proof. First we prove by induction on the length of a derivation in G that for each $A \in N' - \{S'\}$ and each $w \in \Sigma^*$, $A \stackrel{*}{\Rightarrow} w$ or $X_{A_i} \stackrel{*}{\Rightarrow} w$ in G implies $A \stackrel{*}{\Rightarrow} w$ in G'. Suppose first that $A \Rightarrow w$ or $X_{A_i} \Rightarrow w$ in G. Then $A \rightarrow w$ or $X_{A_i} \rightarrow w$ is in P. By the construction 1 of P, $A \rightarrow w$ is in P'. Hence $A \Rightarrow w$ in G'.

Next suppose that the result holds for all derivations of the length at most m. Let $A \Rightarrow BC \stackrel{*}{\Rightarrow} w$ or $X_{A_i} \Rightarrow BC \stackrel{*}{\Rightarrow} w$ $(B, C \in N')$ be a derivation of length $m+1$ in G. This implies that $A \rightarrow BC$ or $X_{A_i} \rightarrow BC$ is in P and $BC \stackrel{*}{\Rightarrow} w$ is a derivation of length m in G. By the construction 1 of P and the induction hypothesis, $A \rightarrow BC$ is in P' and $BC \stackrel{*}{\Rightarrow} w$ in G'. Hence $A \stackrel{*}{\Rightarrow} w$ in G'. Let $A \Rightarrow X \stackrel{*}{\Rightarrow} w$ or $X_{A_i} \Rightarrow X \stackrel{*}{\Rightarrow} w$ $(X \in N)$ be a derivation of length $m+1$ in G. By the construction 1 of P, this implies that $X = X_{A_j}$, $A \rightarrow X_{A_j}$ or $X_{A_i} \rightarrow X_{A_j}$ is in P, and $X_{A_j} \stackrel{*}{\Rightarrow} w$ is a derivation of length m in G. By the induction hypothesis, $A \stackrel{*}{\Rightarrow} w$ in G'. This completes the induction.

Suppose that $S \stackrel{*}{\Rightarrow} w$ in G. By the constructions 2 and 3, this implies that $S \stackrel{*}{\Rightarrow} BY_C \Rightarrow BC$ in G, $S' \Rightarrow A \Rightarrow BC$ in G', and $BC \stackrel{*}{\Rightarrow} w$ in G for some $B, C \in N' - \{S'\}$. By the above result, $BC \stackrel{*}{\Rightarrow} w$ in G'. Hence $S' \stackrel{*}{\Rightarrow} w$ in G'. This completes the proof of Claim 3.

By Claim 2 and 3, $L(G') = L(G)$. To finish the proof, note that G is ϵ-free. □

We analyze how much the transformation used in Theorem 2.8 blows up the size of the grammar. Let $G' = (N', \Sigma, P', S')$ be any invertible context-free grammar such that each production in P' has the form given in Theorem 2.8 and $G = (N, \Sigma, P, S)$ be the resulting equivalent reversible context-free grammar by the transformation. Then $|N| \leq 2|N'| + 2|P'| - 3$ and $|P| \leq 4|P'| + |N'| - 3$. Thus this transformation polynomially blows up the size of the grammar. However the transformation of any context-free grammar into an equivalent one that is invertible suffers the exponential blowup in the number of nonterminals in the grammer.

Note that while the standard transformation to make a context-free grammar invertible preserves structural equivalence (see [McN67] for example), the transformations used in Theorem 2.7, 2.8 to achieve reset-freeness in general does not, and cannot always, preserve structural equivalence, although it preserves language equivalence. This is because some sets of skeletons accepted by skeletal tree automata are not accepted by any reversible skeletal tree automaton, which is the correct analog of the theory in the case of finite automata, where not all regular languages are reversible.

Definition A context-free grammar $G = (N, \Sigma, P, S)$ is said to be *extended reversible*

if and only if for $P' = P - \{S \to a \mid a \in \Sigma\}$, $G' = (N, \Sigma, P', S)$ is reversible.

By the above theorem, reversible context-free grammars can be easily extended so that for any context-free language not containing ϵ, we can find an extended reversible context-free grammar which is ϵ-free and generates the language.

Theorem 2.9 *Let L be any context-free language not containing ϵ. Then there is an ϵ-free extended reversible context-free grammar G such that $L(G) = L$.*

2.7 Learning Algorithms from Positive Presentations

In this section we first describe and analyze the algorithm *RTA* to learn reversible skeletal tree automata from positive presentations. Next we apply this algorithm to learning context-free grammars from positive presentations of structured strings. Essentially the algorithm *RTA* is an extension of Angluin's learning algorithm [Ang82] for zero-reversible automata. Without loss of generality, we restrict our consideration to only ϵ-free context-free grammars.

First we give several definitions about some basic operations of set and tree automata, which we will need to describe and analyze learning algorithms in the sequel.

A *partition* of some set S is a set of pairwise disjoint nonempty subsets of S whose union is S. If π is a partition of S, then for any element $s \in S$ there is a unique element of π containing s, which we denote $B(s, \pi)$ and call the *block* of π containing s. A partition π is said to *refine* another partition π', or π is *finer* than π', if and only if every block of π' is a union of blocks of π. If π is a partition of a set S and S' is a subset of S, then the *restriction* of π to S' is the partition $\pi' = \{B \cap S' \mid B \in \pi, B \cap S' \neq \emptyset\}$ of S'. The *trivial partition* of a set S is the class of all singleton sets $\{s\}$ such that $s \in S$.

Definition Let $A = (Q, V, \delta, F)$ be any tree automaton. If π is any partition of Q, we define the tree automaton $A/\pi = (Q', V, \delta', F')$ induced by π as follows: Q' is the set of blocks of π (i.e. $Q' = \pi$). F' is the set of all blocks of π that contain an element of F (i.e. $F' = \{B \in \pi \mid B \cap F \neq \emptyset\}$). δ' is a mapping from $V_k \times (\pi \cup V_0)^k$ to 2^π and for $B_1, \ldots, B_k \in Q' \cup V_0$ and $f \in V_k$, the block B is in $\delta'_k(f, B_1, \ldots, B_k)$ whenever there exist $q \in B$ and $q_i \in B_i \in \pi$ or $q_i = B_i \in V_0$ for $1 \leq i \leq k$ such that $q \in \delta_k(f, q_1, \ldots, q_k)$.

Lemma 2.10 *Let $A = (Q, V, \delta, F)$ be a tree automaton and π be a partition of Q. Then $T(A/\pi) \supseteq T(A)$, $T(A/\pi) = T(A)$ if π is the trivial partition of Q, and $T(A/\pi) \subseteq T(A/\pi')$ if π refines π'.*

Definition Let Sa be a finite set of trees of V^T. We define the *base tree automaton* for Sa, denoted $Bs(Sa) = (Q, V, \delta, F)$, as follows :

$$\begin{aligned} Q &= Sub(Sa) - V_0, \\ F &= Sa, \\ \delta_k(f, u_1, \ldots, u_k) &= f(u_1, \ldots, u_k) \\ &\qquad \text{whenever } u_1, \ldots, u_k \in Q \cup V_0 \text{ and } f(u_1, \ldots, u_k) \in Q. \end{aligned}$$

Note that $Bs(Sa)$ is a tree automaton that accepts precisely the set Sa.

Definition A *positive sample* of a tree automaton A is a finite subset of $T(A)$. A positive sample Sa of a reversible skeletal tree automaton A is a *characteristic sample* for A if and only if for any reversible skeletal tree automaton A', $Sa \subseteq T(A')$ implies $T(A) \subseteq T(A')$.

2.7.1 The Learning Algorithm *RTA* for Tree Automata

The input to *RTA* is a finite nonempty set Sa of skeletons. The output is a particular reversible skeletal tree automaton $A = RTA(Sa)$. The learning algorithm *RTA* begins with the base tree automaton for Sa and generalizes it by merging states. *RTA* finds a reversible skeletal tree automaton whose characteristic sample is equal to the input sample.

On input Sa, *RTA* first constructs $A = Bs(Sa)$, the base tree automaton for Sa. It then constructs the finest partition π_f of the set Q of states of A with the property that A/π_f is reversible, and outputs A/π_f.

To construct π_f, *RTA* begins with the trivial partition of Q and repeatedly merges any two distinct blocks B_1 and B_2 if any of the following conditions is satisfied.

1. B_1 and B_2 both contain final states of A.
2. There exist two states $q \in B_1$ and $q' \in B_2$ of the forms $q = \sigma(u_1, \ldots, u_k)$ and $q' = \sigma(u'_1, \ldots, u'_k)$ such that for $1 \leq j \leq k$, u_j and u'_j both are in the same block or the same terminal symbols.
3. There exist two states q, q' of the forms $q = \sigma(u_1, \ldots, u_k)$ and $q' = \sigma(u'_1, \ldots, u'_k)$ in the same block and an integer l $(1 \leq l \leq k)$ such that $u_l \in B_1$ and $u'_l \in B_2$ and for $1 \leq j \leq k$ and $j \neq l$, u_j and u'_j both are in the same block or the same terminal symbols.

When there no longer remains any such pair of blocks, the resulting partition is π_f.

To implement this merging process, *RTA* keeps track of the further merges immediately implied by each merge performed. The variable LIST contains a list of pairs of states whose corresponding blocks are to be merged. *RTA* initially selects some final state q of A and places on LIST all pairs (q, q') such that q' is a final state of A other than q. This ensures that all blocks containing a final state of A will eventually be merged.

After these initializations, *RTA* proceeds as follows. While the list LIST is nonempty, *RTA* removes the first pair of states (q_1, q_2). If q_1 and q_2 are already in the same block of the current partition, *RTA* goes on to the next pair of states in LIST. Otherwise, the blocks containing q_1 and q_2, call them B_1 and B_2, are merged to form a new block B_3. This action entails that LIST be updated as follows. For any two states $q, q' \in Q$ of the forms $q = \sigma(u_1, \ldots, u_k)$ and $q' = \sigma(u'_1, \ldots, u'_k)$, if q and q' are not in the same block and u_j and u'_j both are in the same block or the same terminal symbols for $1 \leq j \leq k$, then the pair (q, q') is added to LIST. Also for any $q \in B_1, q' \in B_2$ of the forms $q = \sigma(u_1, \ldots, u_k)$ and $q' = \sigma(u'_1, \ldots, u'_k)$ and an integer l $(1 \leq l \leq k)$, if u_l and u'_l are states of A and not in the same block and u_j and u'_j both are in the same block or the same terminal symbols for $1 \leq j \leq k$ and $j \neq l$, then the pair (u_l, u'_l) is added to LIST. After this updating, *RTA* goes on to the next pair of states from LIST.

When LIST becomes empty, the current partition is π_f. *RTA* outputs A/π_f and halts.

The learning algorithm *RTA* is illustrated in Figure 2.2. This completes the description of the algorithm *RTA*, and we next analyze its correctness.

2.7.2 Correctness of *RTA*

In this section, we show that *RTA* correctly finds a reversible skeletal tree automaton whose characteristic sample is equal to the input sample.

For any $t \in V^T$, we denote the *quotient* of T and t by

$$U_T(t) = \begin{cases} \{u \mid u \in V_\$^T \text{ and } u\#t \in T\} & \text{if } t \in V^T - V_0, \\ t & \text{if } t \in V_0. \end{cases}$$

Lemma 2.11 *Let A be a deterministic tree automaton. If $\delta(t_1) = \delta(t_2)$, then $U_{T(A)}(t_1) = U_{T(A)}(t_2)$.*

Proof. It is straightforward from the replacement lemma. □

Let T be a set of trees. We define the partition π_T of V^T associated with T by $B(t_1, \pi_T) = B(t_2, \pi_T)$ if and only if $U_T(t_1) = U_T(t_2)$.

ALGORITHM *RTA*

Input: A nonempty positive sample Sa.
Output: A reversible skeletal tree automaton A.
Procedure:

```
  %% Initialization
1 Let A = (Q, V, δ, F) be Bs(Sa);
2 Let π_0 be the trivial partition of Q;
3 Choose some q ∈ F;
4 Let LIST contain all pairs (q, q') such that q' ∈ F − {q};
5 Let i = 0;
  %% Main Routine
  %% Merging
6 While LIST≠ ∅ do
7       Begin
8             Remove first element (q_1, q_2) from LIST;
9             Let B_1 = B(q_1, π_i) and B_2 = B(q_2, π_i);
10            If B_1 ≠ B_2 then
11              Begin
12                    Let π_{i+1} be π_i with B_1 and B_2 merged;
13                    p-UPDATE(π_{i+1}) and s-UPDATE(π_{i+1}, B_1, B_2);
14                    Increase i by 1;
15              End
16      End
  %% Termination
17 Let f = i and output the tree automaton A/π_f.
  %% Sub-routine
18 where
19      p-UPDATE(π_{i+1}) is :
20        For all pairs of states σ(u_1, ..., u_k) and σ(u'_1, ..., u'_k) in Q with
              B(u_j, π_{i+1}) = B(u'_j, π_{i+1}) or u_j = u'_j ∈ Σ for 1 ≤ j ≤ k
              and B(σ(u_1, ..., u_k), π_{i+1}) ≠ B(σ(u'_1, ..., u'_k), π_{i+1})
21        do
22            Add the pair (σ(u_1, ..., u_k), σ(u'_1, ..., u'_k)) to LIST;
23      s-UPDATE(π_{i+1}, B_1, B_2) is :
24        For all pairs of states σ(u_1, ..., u_k) ∈ B_1 and σ(u'_1, ..., u'_k) ∈ B_2 with
            u_l, u'_l ∈ Q and B(u_l, π_{i+1}) ≠ B(u'_l, π_{i+1}) for some l (1 ≤ l ≤ k)
            and B(u_j, π_{i+1}) = B(u'_j, π_{i+1}) or u_j = u'_j ∈ Σ for 1 ≤ j ≤ k and j ≠ l
25        do
26            Add the pair (u_l, u'_l) to LIST.
```

Figure 2.2: The learning algorithm *RTA* for Reversible Tree Automata

Lemma 2.12 *Let T be a set of trees. For $t_i, u_i \in V^T (1 \leq i \leq k)$ and $f \in V_k$, $B(t_i, \pi_T) = B(u_i, \pi_T)$ implies $B(f(t_1, \ldots, t_k), \pi_T) = B(f(u_1, \ldots, u_k), \pi_T)$.*

Let $A = (Q, V, \delta, F)$ and $A' = (Q', V, \delta', F')$ be tree automata. A is *isomorphic* to A' if and only if there exists a bijection φ of Q onto Q' such that $\varphi(F) = F'$ and for every $q_1, \ldots, q_k \in Q \cup V_0$ and $f \in V_k$, $\varphi(\delta_k(f, q_1, \ldots, q_k)) = \delta'_k(f, q'_1, \ldots, q'_k)$ where $q'_i = \varphi(q_i)$ if $q_i \in Q$ and $q'_i = q_i$ if $q_i \in V_0$ for $1 \leq i \leq k$.

Let A be a deterministic tree automaton which accepts a set of trees T. A is *minimum* if and only if A has the minimum number of states among all deterministic tree automata which accept T. The minimum deterministic tree automaton is unique up to isomorphism [Bra68].

Definition Let $A = (Q, V, \delta, F)$ and $A' = (Q', V, \delta', F')$ be tree automata. A' is a *tree subautomaton* of A if and only if Q' and F' are subsets of Q and F respectively and for every $q'_1, \ldots, q'_k \in Q' \cup V_0$ and $f \in V_k$, $\delta'_k(f, q'_1, \ldots, q'_k) \subseteq \delta_k(f, q'_1, \ldots, q'_k)$.

Clearly $T(A') \subseteq T(A)$.

Definition Let $A = (Q, V, \delta, F)$ be a tree automaton. If Q'' is a subset of Q, then the *tree subautomaton of A induced by Q''* is the tree automaton (Q'', V, δ'', F''), where F'' is the intersection of Q'' and F, and $q'' \in \delta''_k(f, q''_1, \ldots, q''_k)$ if and only if $q'' \in Q''$, $q''_1, \ldots, q''_k \in Q'' \cup V_0$, and $q'' \in \delta_k(f, q''_1, \ldots, q''_k)$.

A state q of A is called *useful* if and only if there exist a tree t and some address $x \in Dom_t$ such that $q \in \delta(t/x)$ and $\delta(t) \cap F \neq \emptyset$. States that are not useful are called *useless*. A tree automaton that contains no useless states is called *stripped*.

Definition The *stripped tree subautomaton* of A is the tree subautomaton of A induced by the useful states of A.

The "stripped tree subautomaton" in fact contains no useless states, that is, stripped.

Definition Let T be a set of trees accepted by some tree automaton. We define the *canonical tree automaton* for T, denoted $C(T) = (Q, V, \delta, F)$, as follows :

$$
\begin{aligned}
Q &= \{U_T(u) \mid u \in Sub(T) - V_0\}, \\
F &= \{U_T(t) \mid t \in T\}, \\
\delta_k(f, U_T(u_1), \ldots, U_T(u_k)) &= U_T(f(u_1, \ldots, u_k)) \\
&\qquad \text{if } u_1, \ldots, u_k \text{ and } f(u_1, \ldots, u_k) \text{ are in } Sub(T).
\end{aligned}
$$

Since T is accepted by some tree automaton, the set $\{U_T(u) \mid u \in Sub(T) - V_0\}$ is finite by Lemma 2.11. Since $U_T(u_1) = U_T(u_2)$ implies $U_T(t\#u_1) = U_T(t\#u_2)$ for all trees t in $V_\$^T$, this state transition function is well defined and $C(T)$ is deterministic. $C(T)$ is stripped, that is, contains no useless states. $C(T)$ is the minimum deterministic tree automaton.

A tree automaton A is called *canonical* if and only if A is isomorphic to the canonical tree automaton for $T(A)$.

Lemma 2.13 *Let Sa be a positive sample of some tree automaton A. Let π be the partition $\pi_{T(A)}$ restricted to the set $Sub(Sa) - \Sigma$. Then $Bs(Sa)/\pi$ is isomorphic to a tree subautomaton of the canonical tree automaton $C(T(A))$. Furthermore, $T(Bs(Sa)/\pi)$ is contained in $T(A)$.*

Proof. The result holds trivially if $Sa = \emptyset$, so assume that $Sa \neq \emptyset$. Let $Bs(Sa)/\pi = (Q, V, \delta, F)$ and $C(T(A)) = (Q', V, \delta', F')$. The partition π is defined by $B(t_1, \pi) = B(t_2, \pi)$ if and only if $U_{T(A)}(t_1) = U_{T(A)}(t_2)$, for all $t_1, t_2 \in Sub(Sa) - \Sigma$. Hence $h(B(t, \pi)) = U_{T(A)}(t)$ is a well-defined and injective map from Q to Q'. If B_1 is a final state of $Bs(Sa)/\pi$, then $B_1 = B(t, \pi)$ for some t in Sa, and since $T(A)$ contains Sa, $U_{T(A)}(t)$ is a final state of $C(T(A))$. Hence h maps F to F'.

$Bs(Sa)/\pi$ is deterministic because for $f(t_1, \ldots, t_k)$ and $f(u_1, \ldots, u_k)$ in $Sub(Sa)$, $B(t_i, \pi) = B(u_i, \pi)$ if $t_i, u_i \in Sub(Sa) - \Sigma$ and $t_i = u_i$ if $t_i, u_i \in \Sigma$ $(1 \leq i \leq k)$ imply $B(f(t_1, \ldots, t_k), \pi) = B(f(u_1, \ldots, u_k), \pi)$ by Lemma 2.12. For $q_1, \ldots, q_k \in Q \cup \Sigma$ and $f \in V_k$,

$$\begin{aligned} h(\delta_k(f, q_1, \ldots, q_k)) &= h(B(f(t_1, \ldots, t_k), \pi)), \\ &\qquad \text{where } B(t_i, \pi) = q_i \text{ if } q_i \in Q \text{ and } t_i = q_i \text{ if } q_i \in \Sigma \ (1 \leq i \leq k), \\ &= U_{T(A)}(f(t_1, \ldots, t_k)) \\ &= \delta'_k(f, U_{T(A)}(t_1), \ldots, U_{T(A)}(t_k)). \end{aligned}$$

Thus h is an isomorphism between $Bs(Sa)/\pi$ and a tree subautomaton of $C(T(A))$. □

Lemma 2.14 *If A is a reversible skeletal tree automaton and A' is any tree subautomaton of A, then A' is a reversible skeletal tree automaton.*

Lemma 2.15 *Let $A = (Q, Sk \cup \Sigma, \delta, \{q_f\})$ be a reversible skeletal tree automaton. For $t \in (Sk \cup \Sigma)_\$^T$ and $u_1, u_2 \in (Sk \cup \Sigma)^T$, if A accepts both $t\#u_1$ and $t\#u_2$, then $\delta(u_1) = \delta(u_2)$.*

Proof. We prove it by induction on the depth of the node labelled \$ in t. Suppose first that $t = \$$. Since A has only one final state q_f, $\delta(u_1) = \delta(t\#u_1) = q_f = \delta(t\#u_2) = \delta(u_2)$. Next suppose that the result holds for all $t \in (Sk \cup \Sigma)^T_{\$}$ in which the depth of the node labelled \$ is at most h. Let t be an element of $(Sk \cup \Sigma)^T_{\$}$ in which the depth of the node labelled \$ is $h+1$, so that $t = t'\#\sigma(s_1, \ldots, s_{i-1}, \$, s_i, \ldots, s_{k-1})$ for some $s_1, \ldots, s_{k-1} \in (Sk \cup \Sigma)^T$, $i \in \mathbf{N}$ and $t' \in (Sk \cup \Sigma)^T_{\$}$ in which the depth of the node labelled \$ is h. If A accepts both $t\#u_1 = t'\#\sigma(s_1, \ldots, s_{i-1}, u_1, s_i, \ldots, s_{k-1})$ and $t\#u_2 = t'\#\sigma(s_1, \ldots, s_{i-1}, u_2, s_i, \ldots, s_{k-1})$, then $\delta(\sigma(s_1, \ldots, s_{i-1}, u_1, s_i, \ldots, s_{k-1})) = \delta(\sigma(s_1, \ldots, s_{i-1}, u_2, s_i, \ldots, s_{k-1}))$ by the induction hypothesis. So

$$\begin{aligned} &\delta_k(\sigma, \delta(s_1), \ldots, \delta(s_{i-1}), \delta(u_1), \delta(s_i), \ldots, \delta(s_{k-1})) \\ &= \delta_k(\sigma, \delta(s_1), \ldots, \delta(s_{i-1}), \delta(u_2), \delta(s_i), \ldots, \delta(s_{k-1})). \end{aligned}$$

Since A is reset-free, $\delta(u_1) = \delta(u_2)$, which completes the induction and the proof of Lemma 2.15. □

Lemma 2.16 *Suppose A is a reversible skeletal tree automaton. Then the stripped tree subautomaton A' of A is canonical.*

Proof. By Lemma 2.14, A' is a reversible skeletal tree automaton, and accepts $T = T(A)$. If $T = \emptyset$, then A' is the tree automaton with the empty set of states and therefore canonical. So suppose that $T \neq \emptyset$. Let $C(T) = (Q, Sk \cup \Sigma, \delta, \{q_f\})$ and $A' = (Q', Sk \cup \Sigma, \delta', \{q'_f\})$. We define $h(q') = U_T(u)$ if $\delta'(u) = q'$ for $q' \in Q'$. By Lemma 2.11, h is a well-defined and surjective map from Q' to Q. Let q'_1 and q'_2 be states of A', and suppose that $U_T(u_1) = U_T(u_2)$ for u_1 and u_2 such that $\delta'(u_1) = q'_1$ and $\delta'(u_2) = q'_2$. Since A' is stripped, this implies that there exists a tree $t \in (Sk \cup \Sigma)^T_{\$}$ such that $t\#u_1$ and $t\#u_2$ are in T. Thus, by Lemma 2.15, $q'_1 = q'_2$. Hence h is injective. Since $\delta'(u) = q'_f$ for any $u \in T$, h maps $\{q'_f\}$ to $\{q_f\}$. For $q'_1, \ldots, q'_k \in Q' \cup \Sigma$,

$$\begin{aligned} h(\delta'_k(\sigma, q'_1, \ldots, q'_k)) &= h(\delta'(\sigma(u_1, \ldots, u_k))), \\ &\qquad \text{where } \delta'(u_i) = q'_i \text{ for } 1 \leq i \leq k, \\ &= U_T(\sigma(u_1, \ldots, u_k)) \\ &= \delta_k(\sigma, U_T(u_1), \ldots, U_T(u_k)). \end{aligned}$$

Thus h is an isomorphism between $C(T)$ and A'. Hence A' is canonical. □

Lemma 2.17 *Suppose that A is a reversible skeletal tree automaton. Then the canonical tree automaton $C(T(A))$ is reversible.*

Proof. By the above lemma and Lemma 2.14, the stripped tree subautomaton A' of A is canonical, reversible, and accepts $T(A)$. Thus, since $C(T(A))$ is isomorphic to A', $C(T(A))$ is reversible. □

Lemma 2.18 *Let Sa be any nonempty positive sample of skeletons, and π_f be the final partition found by RTA on input Sa. Then π_f is the finest partition such that $Bs(Sa)/\pi_f$ is reversible.*

Proof. Let $A = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ be $Bs(Sa)$. If the pair (q_1, q_2) is ever placed on LIST, then q_1 and q_2 must be in the same block of the final partition, that is, $B(q_1, \pi_f) = B(q_2, \pi_f)$. Therefore, the initialization guarantees that all the final states of A are in the same block of π_f, so A/π_f has exactly one final state. For any $B_1, \ldots, B_k \in \pi_f \cup \Sigma$, all the elements of $\delta_k(\sigma, B_1, \ldots, B_k)$ are contained in one block of π_f. Thus A/π_f is deterministic. Also, for any block B of π_f, any pair of states $q_1, q_2 \in B$ of the forms $q_1 = \sigma(u_1, \ldots, u_k)$ and $q_2 = \sigma(u'_1, \ldots, u'_k)$ and any integer l $(1 \leq l \leq k)$, if $B(u_j, \pi_f) = B(u'_j, \pi_f)$ or $u_j = u'_j \in \Sigma$ for $1 \leq j \leq k$ and $j \neq l$, then both u_l and u'_l are in the same block or the same terminal symbols. Thus A/π_f is reset-free. Hence A/π_f is reversible.

Next we show that if π is any partition of Q such that A/π is reversible, then π_f refines π. We prove by induction that π_i refines π for $i = 0, 1, \ldots, f$. Clearly π_0, the trivial partition of Q, refines π. Suppose that $\pi_0, \pi_1, \ldots, \pi_i$ all refines π and π_{i+1} is obtained from π_i by merging the blocks $B(q_1, \pi_i)$ and $B(q_2, \pi_i)$ in the course of processing entry (q_1, q_2) from LIST. Since π_i refines π, $B(q_1, \pi_i)$ is a subset of $B(q_1, \pi)$ and $B(q_2, \pi_i)$ is a subset of $B(q_2, \pi)$. So in order to show that π_{i+1} refines π, it is sufficient to show that $B(q_1, \pi) = B(q_2, \pi)$.

If (q_1, q_2) was first placed on LIST during the initialization stage, then q_1 and q_2 are both final states, and since A/π is reversible, it has only one final state, and so $B(q_1, \pi) = B(q_2, \pi)$. Otherwise, (q_1, q_2) was first placed on LIST in consequence of some previous merge, say the merge to produce π_m from π_{m-1}, where $0 < m \leq i$. Then either q_1 and q_2 are of the forms $\sigma(u_1, \ldots, u_k)$ and $\sigma(u'_1, \ldots, u'_k)$ respectively and $B(u_j, \pi_m) = B(u'_j, \pi_m)$ or $u_j = u'_j \in \Sigma$ for $1 \leq j \leq k$, or there exist two states q'_1 in the block B_1 and q'_2 in the block B_2 of the forms $\sigma(u_1, \ldots, u_{l-1}, q_1, u_l, \ldots, u_{k-1})$ and $\sigma(u'_1, \ldots, u'_{l-1}, q_2, u'_l, \ldots, u'_{k-1})$ respectively for some l $(1 \leq l \leq k)$ such that $B(u_j, \pi_m) = B(u'_j, \pi_m)$ or $u_j = u'_j \in \Sigma$ for $1 \leq j \leq k-1$, where B_1 and B_2 are the blocks of π_{m-1} merged in forming π_m. Since π_m refines π by the induction hypothesis and A/π is reversible, $B(q_1, \pi) = B(q_2, \pi)$. Thus in either case π_{i+1} refines π. Hence by finite induction we conclude that π_f refines π. □

Theorem 2.19 *Let Sa be a nonempty positive sample of skeletons, and A_f be the skeletal tree automaton output by the algorithm RTA on input Sa. Then for any reversible skeletal tree automaton A, $Sa \subseteq T(A)$ implies $T(A_f) \subseteq T(A)$.*

Proof. The preceding lemma shows that A_f is a reversible skeletal tree automaton such that $T(A_f) \supseteq Sa$. Let A be any reversible skeletal tree automaton such that $T(A) \supseteq Sa$, and π be the restriction of the partition $\pi_{T(A)}$ to the set $Sub(Sa) - \Sigma$. Lemma 2.13 shows that $Bs(Sa)/\pi$ is isomorphic to a tree subautomaton of $C(T(A))$ and $T(Bs(Sa)/\pi)$ is contained in $T(A)$. Lemma 2.17 shows that $C(T(A))$ is reversible, and therefore by Lemma 2.14, $Bs(Sa)/\pi$ is reversible. Let π_f be the final partition found by *RTA*. By the above lemma, π_f refines π, so $T(Bs(Sa)/\pi_f) = T(A_f)$ is contained in $T(Bs(Sa)/\pi)$ by Lemma 2.10. Hence, $T(A_f)$ is contained in $T(A)$. □

2.7.3 Time Complexity of *RTA*

Theorem 2.20 *The algorithm RTA may be implemented to run in time polynomial in the sum of the sizes of the input skeletons, where the size of a skeleton (or tree) t is the number of nodes in t, i.e. $|Dom_t|$.*

Proof. Let Sa be the set of input skeletons, n be the sum of the sizes of the skeletons in Sa, and d be the maximum rank of the symbol σ in Sk. The base tree automaton $A = Bs(Sa)$ may be constructed in time $O(n)$ and contains at most n states. Similarly, the time to output the final tree automaton is $O(n)$. The partitions π_i of the states of A may be queried and updated using the simple MERGE and FIND operations described by Aho, Hopcroft and Ullman [AHU83]. Processing each pair of states from LIST entails two FIND operations to determine the blocks containing the two states. If the blocks are distinct, which can happen at most $n-1$ times, they are merged with a MERGE operation, and p-UPDATE and s-UPDATE procedures process $2(d+1)n(n-1)$ and at most $2dn(n-1)$ FIND operations respectively. Further at most $n-1$ new pairs may be placed on LIST. Thus a total of at most $2n(n-1)+(n-1)$ pairs must be placed on LIST. Thus at most $2((2d+1)n(n-1)+2n+1)(n-1)$ FIND operations and $n-1$ MERGE operations are required. The operation MERGE takes $O(n)$ time and the operation FIND takes constant time, so *RTA* requires a total time of $O(n^3)$. □

2.7.4 Identification in the Limit of Reversible Tree Automata

Next we show that the algorithm *RTA* may be used at the finite stages of an infinite learning process to identify the reversible skeletal tree automata in the limit from positive

presentations. The idea is simply to run *RTA* on the sample at the nth stage and output the result as the nth guess.

Definition For an infinite sequence $s_1, s_2, s_3, \ldots$ of skeletons and an infinite sequence $A_1, A_2, A_3, \ldots$ of skeletal tree automata, if

$$A_i = RTA(\{s_1, s_2, \ldots, s_i\})$$

holds, then we write $RTA_\infty(\{s_1, s_2, s_3, \ldots\}) = \{A_1, A_2, A_3, \ldots\}$.

Definition An infinite sequence $s_1, s_2, s_3, \ldots$ of skeletons is defined to be a *positive presentation* of a skeletal tree automaton A if and only if the set $\{s_1, s_2, s_3, \ldots\}$ is precisely $T(A)$. An infinite sequence of skeletal tree automata $A_1, A_2, A_3, \ldots$ is said to *converge to* a skeletal tree automaton A if and only if there exists an integer N such that for all $i \geq N$, A_i is isomorphic to A.

We will show that the output of RTA_∞ on a positive presentation converges to the correct guess after a finite number of stages. The following result is necessary for the proof of correct identification in the limit of the reversible skeletal tree automata from positive presentation. We extend δ to $(V \cup Q)^T$ by letting $\delta(q) = q$ for $q \in Q$, where Q is considered as a set of terminal symbols. In this definition, if $q = \delta(u)$ for $q \in Q$ and $u \in V^T$, then $\delta(t\#q) = \delta(t\#u)$ for $t \in V_\$^T$.

Theorem 2.21 *For any reversible skeletal tree automaton $A = (Q, \{\sigma\} \cup \Sigma, \delta, \{q_f\})$, there effectively exists a characteristic sample.*

Proof. Clearly, if $T(A) = \emptyset$, then $CS = \emptyset$ is a characteristic sample for A. Suppose $T(A) \neq \emptyset$. For each state $q \in Q$, let $u(q)$ be a tree of the minimum size in $Sub(T(A))$ such that $\delta(u(q)) = q$, and $v(q)$ be a tree of the minimum size in $Sc(T(A))$ such that $\delta(v(q)\#q) = q_f$. For each $a \in \Sigma$, let $u(a) = a$. Let CS consist of all skeletons of the form $v(q)\#u(q)$ such that $q \in Q$ and all skeletons of the form $v(q)\#\sigma(u(q_1), \ldots, u(q_k))$ such that $q_1, \ldots, q_k \in Q \cup \Sigma$ and $q = \delta_k(\sigma, q_1, \ldots, q_k)$. It is clear that $CS \subseteq T(A)$. We show that CS is a characteristic sample for A.

Let A' be any reversible skeletal tree automaton such that $T(A') \supseteq CS$. We show that $U_{T(A')}(t) = U_{T(A')}(u(q))$ for all skeletons $t \in Sub(T(A))$, where $q = \delta(t)$. We prove it by induction on the depth of t. Suppose first that the depth of t is 0, i.e. $t = a \in \Sigma$. Since $u(a) = a$, it holds for the depth 0. Next suppose that this holds for all skeletons of depth at most h, for some $h \geq 0$. Let t be a skeleton of depth $h + 1$ from

$Sub(T(A))$, so that $t = \sigma(s_1, \ldots, s_k)$ for some skeletons $s_1, \ldots, s_k \in Sub(T(A))$ with depth at most h. By the induction hypothesis, $U_{T(A')}(s_i) = U_{T(A')}(u(q_i))$, where $q_i = \delta(s_i)$ for $1 \leq i \leq k$. Thus, $U_{T(A')}(t) = U_{T(A')}(\sigma(s_1, \ldots, s_k)) = U_{T(A')}(\sigma(u(q_1), s_2, \ldots, s_k)) = \cdots = U_{T(A')}(\sigma(u(q_1), \ldots, u(q_{k-1}), s_k)) = U_{T(A')}(\sigma(u(q_1), \ldots, u(q_k)))$. If $q' = \delta_k(\sigma, q_1, \ldots, q_k) = \delta(t)$, then $v(q')\#u(q')$ and $v(q')\#\sigma(u(q_1), \ldots, u(q_k))$ are both elements of CS. So $v(q')\#u(q') \in T(A')$ and $v(q')\#\sigma(u(q_1), \ldots, u(q_k)) \in T(A')$. By Lemma 2.15, $U_{T(A')}(\sigma(u(q_1), \ldots, u(q_k))) = U_{T(A')}(u(q'))$. Hence $U_{T(A')}(t) = U_{T(A')}(u(q'))$, which completes the induction.

Thus for every $t \in T(A)$, $U_{T(A')}(t) = U_{T(A')}(u(q_f))$. Since $v(q_f) = \$$, $u(q_f) \in CS$ and so $u(q_f) \in T(A')$. This implies that $\$ \in U_{T(A')}(u(q_f)) = U_{T(A')}(t)$. Thus $t = \$\#t \in T(A')$. Hence $T(A)$ is contained in $T(A')$. Therefore CS is a characteristic sample for A. □

Then we conclude the following result.

Theorem 2.22 *Let A be a reversible skeletal tree automaton, $s_1, s_2, s_3, \ldots$ be a positive presentation of A, and $A_1, A_2, A_3, \ldots$ be the output of RTA_∞ on this input. Then $A_1, A_2, A_3, \ldots$ converges to the canonical skeletal tree automaton A' for $T(A)$.*

Proof. By Theorem 2.21, there exists a characteristic sample for A. Let N be sufficiently large that the set $\{s_1, s_2, \ldots, s_N\}$ contains a characteristic sample for A. For any reversible skeletal tree automaton A', $\{s_1, s_2, \ldots, s_i\} \subseteq T(A')$ implies $T(A_i) \subseteq T(A')$, by the definition of RTA_∞ and Theorem 2.19. Thus for $i \geq N$, $T(A_i) = T(A)$, by the definition of a characteristic sample. Moreover it is easily checked that the skeletal tree automaton output by RTA is stripped, and therefore canonical, by Lemma 2.16. Hence A_i is isomorphic to $C(T(A))$ for all $i \geq N$, so $A_1, A_2, A_3, \ldots$ converges to $C(T(A))$. □

We may modify RTA by a simple updating scheme to have good incremental behavior so that A_{i+1} may be obtained from A_i and s_{i+1}.

2.7.5 The Learning Algorithm *RCFG* for Context-Free Grammars

In this section, we describe and analyze the algorithm $RCFG$ using the algorithm RTA to learn reversible context-free grammars from positive presentations of structured strings.

A *positive structural sample* of a context-free grammar G is a finite subset Sa of $s(D(G))$. A positive structural sample Sa of a reversible context-free grammar G is a

ALGORITHM *RCFG*

Input: A nonempty positive structural sample Sa.
Output: A reversible context-free grammar G.
Procedure:
1 Run *RTA* on the sample Sa;
2 Let $G = G'(RTA(Sa))$ and output the grammar G.

Figure 2.3: The learning algorithm *RCFG* for Reversible Grammars

characteristic structural sample for G if and only if for any reversible context-free grammar G', $Sa \subseteq s(D(G'))$ implies $s(D(G)) \subseteq s(D(G'))$.

The input to *RCFG* is a finite nonempty set of skeletons Sa. The output is a particular reversible context-free grammar $G = RCFG(Sa)$ whose characteristic structural sample is equal to Sa. The learning algorithm *RCFG* is illustrated in Figure 2.3.

The following theorems of the correctness, time complexity and correct structural identification in the limit of the algorithm *RCFG* are straightforwardly derived by using Theorem 2.6 from the corresponding results for the algorithm *RTA* described in Sections 2.7.2, 2.7.3 and 2.7.4.

Theorem 2.23 *Let Sa be a nonempty positive structural sample of skeletons, and G_f be the output of the context-free grammar by the algorithm RCFG on input Sa. Then G_f is reversible and for any reversible context-free grammar G, $Sa \subseteq s(D(G))$ implies $s(D(G_f)) \subseteq s(D(G))$.*

Theorem 2.24 *The algorithm RCFG may be implemented to run in time polynomial in the sum of the sizes of the input skeletons.*

Define an operator $RCFG_\infty$ from infinite sequences $s_1, s_2, s_3, \ldots$ of skeletons to infinite sequences $G_1, G_2, G_3, \ldots$ of context-free grammars by

$$G_i = RCFG(\{s_1, s_2, \ldots, s_i\}) \qquad \text{for all } i \geq 1.$$

An infinite sequence $s_1, s_2, s_3, \ldots$ of skeletons is defined to be a *positive structural presentation* of a context-free grammar G if and only if the set $\{s_1, s_2, s_3, \ldots\}$ is precisely $s(D(G))$. An infinite sequence $G_1, G_2, G_3, \ldots$ of context-free grammars is said to *converge to* a context-free grammar G if and only if there exists an integer N such that for all $i \geq N$, G_i is isomorphic to G.

ALGORITHM *ERCFG*

Input: A nonempty positive structural sample Sa.
Output: An extended reversible context-free grammar G.
Procedure:
1 Let $Sa' = Sa - \{\sigma(a) \mid a \in \Sigma\}$;
2 Let $Uni = Sa \cap \{\sigma(a) \mid a \in \Sigma\}$;
3 Run *RCFG* on the sample Sa' and let $G' = (N, \Sigma, P, S)$ be $RCFG(Sa')$;
4 Let $P' = \{S \to a \mid \sigma(a) \in Uni\}$;
5 Let $G = (N, \Sigma, P \cup P', S)$ and output the grammar G.

Figure 2.4: The learning algorithm *ERCFG* for Extended Reversible Grammars

Theorem 2.25 *For any reversible context-free grammar G, there effectively exists a characteristic structural sample.*

Now we have the following.

Theorem 2.26 *Let G be a reversible context-free grammar, $s_1, s_2, s_3, \ldots$ be a positive structural presentation of G, and $G_1, G_2, G_3, \ldots$ be the output of $RCFG_\infty$ on this input. Then $G_1, G_2, G_3, \ldots$ converges to a reversible context-free grammar G' such that $s(D(G')) = s(D(G))$.*

We modify the algorithm *RCFG* to learn extended reversible context-free grammars from positive structural samples. We can easily verify that given a positive structural presentation of an extended reversible context-free grammar G, the algorithm *ERCFG*, illustrated in Figure 2.4, converges to an extended reversible context-free grammar which is structurally equivalent to G and runs in time polynomial in the sum of the sizes of the input skeletons. This implies that the whole class of context-free languages can be learned efficiently from positive presentations of structured strings of extended reversible context-free grammars.

Note that this result does not imply that all context-free grammars can be identified in the limit from positive structural presentations, because even if a source of structural examples is available for some context-free grammar, that context-free grammar may not have any structurally equivalent reversible context-free grammar, and the proposed algorithms *RCFG*, *ERCFG* may fail. We illustrate this point by the following example

in which the positive structural presentation is drawn from a non-reversible context-free grammar and the algorithm *RCFG* fails.

Example 2.2 Consider the following context-free grammar G:

$$S \rightarrow a\ b$$
$$S \rightarrow a\ A\ b$$
$$A \rightarrow a\ b.$$

Then $L(G) = \{ab, aabb\}$, $s(D(G)) = \{\sigma(a,b),\ \sigma(a,\sigma(a,b),b)\}$, G is not reversible, and there is no reversible context-free grammar structurally equivalent to G.

Given the positive structural sample $\{\sigma(a,b),\ \sigma(a,\sigma(a,b),b)\}$ of G, the algorithm *RCFG* outputs the following reversible context-free grammar G':

$$S \rightarrow a\ b$$
$$S \rightarrow a\ S\ b.$$

However $L(G') \neq L(G)$ $(s(D(G')) \neq s(D(G)))$ and hence the algorithm *RCFG* fails to identify G.

2.8 Example Runs

In the process of learning context-free grammars from structured strings, the problem is to reconstruct the nonterminal labels because the set of derivation trees of the unknown context-free grammar is given with all nonterminal labels erased.

The skeletal descriptions of derivation trees of a context-free grammar can be equivalently represented by means of the parenthesis grammar. For example, the structural description in Figure 1.1 can be represented as the following sentence of the parenthesis grammar (see [McN67] for the definition of *parenthesis grammar*):

$$\langle\ \langle\ \text{the}\ \langle\ \text{big dog}\ \rangle\ \rangle\ \langle\ \text{chases}\ \langle\ \text{a}\ \langle\ \text{young girl}\ \rangle\ \rangle\ \rangle\ \rangle$$

In the following, we demonstrate three examples to show the learning process of the algorithm *RCFG*. Three kinds of grammars will be learned, the first is a context-free grammar for a simple natural language, the second is a context-free grammar for a subset of the syntax for a programming language Pascal, and the third is an inherently ambiguous context-free grammar.

2.8.1 Simple Natural Language

Now suppose that the learning algorithm *RCFG* is going to learn the following unknown context-free grammar G_U for a simple natural language:

$Sentence \rightarrow Noun\_phrase\ Verb\_phrase$
$Noun\_phrase \rightarrow Determiner\ Noun\_phrase2$
$Noun\_phrase2 \rightarrow Noun$
$Noun\_phrase2 \rightarrow Adjective\ Noun\_phrase2$
$Verb\_phrase \rightarrow Verb\ Noun\_phrase$
$Determiner \rightarrow$ the
$Determiner \rightarrow$ a
$Noun \rightarrow$ girl
$Noun \rightarrow$ cat
$Noun \rightarrow$ dog
$Adjective \rightarrow$ young
$Verb \rightarrow$ likes
$Verb \rightarrow$ chases.

First suppose that the learning algorithm *RCFG* is given the sample:

⟨ ⟨ ⟨the⟩ ⟨ ⟨girl⟩ ⟩ ⟩ ⟨ ⟨likes⟩ ⟨ ⟨a⟩ ⟨ ⟨cat⟩ ⟩ ⟩ ⟩ ⟩
⟨ ⟨ ⟨the⟩ ⟨ ⟨girl⟩ ⟩ ⟩ ⟨ ⟨likes⟩ ⟨ ⟨a⟩ ⟨ ⟨dog⟩ ⟩ ⟩ ⟩ ⟩

RCFG first constructs the base context-free grammar for them. However it is not reversible. So *RCFG* merges distinct nonterminals repeatedly and outputs the following reversible context-free grammar:

$S \rightarrow NT1\ NT2$
$NT1 \rightarrow NT3\ NT4$
$NT4 \rightarrow NT5$
$NT2 \rightarrow NT6\ NT7$
$NT7 \rightarrow NT8\ NT9$
$NT9 \rightarrow NT10$
$NT3 \rightarrow$ the
$NT5 \rightarrow$ girl
$NT6 \rightarrow$ likes
$NT8 \rightarrow$ a
$NT10 \rightarrow$ cat
$NT10 \rightarrow$ dog.

RCFG has learned that "cat" and "dog" belong to the same syntactic category. However *RCFG* has not learned that "girl" belongs to the same syntactic category (*noun*) as

"cat" and "dog", and "a" and "the" belong to the same syntactic category (*determiner*). Suppose that in the next stage the following examples are added to the sample:

⟨ ⟨ ⟨a⟩ ⟨ ⟨dog⟩ ⟩ ⟩ ⟨ ⟨chases⟩ ⟨ ⟨the⟩ ⟨ ⟨girl⟩ ⟩ ⟩ ⟩ ⟩

⟨ ⟨ ⟨a⟩ ⟨ ⟨dog⟩ ⟩ ⟩ ⟨ ⟨chases⟩ ⟨ ⟨a⟩ ⟨ ⟨cat⟩ ⟩ ⟩ ⟩ ⟩

Then *RCFG* outputs the reversible context-free grammar:

$S \rightarrow NT1\ NT2$
$NT1 \rightarrow NT3\ NT4$
$NT4 \rightarrow NT5$
$NT2 \rightarrow NT6\ NT1$
$NT1 \rightarrow NT7\ NT8$
$NT8 \rightarrow NT9$
$NT3 \rightarrow$ the
$NT5 \rightarrow$ girl
$NT6 \rightarrow$ likes
$NT6 \rightarrow$ chases
$NT7 \rightarrow$ a
$NT9 \rightarrow$ cat
$NT9 \rightarrow$ dog.

RCFG has learned that "likes" and "chases" belong to the same syntactic category (*verb*) and "the girl", "a dog" and "a cat" are identified as the same phrase (*noun_phrase*). However *RCFG* has not learned yet that "a" and "the" belong to the same syntactic category. Suppose that in the further stage the following examples are added to the sample:

⟨ ⟨ ⟨a⟩ ⟨ ⟨dog⟩ ⟩ ⟩ ⟨ ⟨chases⟩ ⟨ ⟨a⟩ ⟨ ⟨girl⟩ ⟩ ⟩ ⟩ ⟩

⟨ ⟨ ⟨the⟩ ⟨ ⟨dog⟩ ⟩ ⟩ ⟨ ⟨chases⟩ ⟨ ⟨a⟩ ⟨ ⟨young⟩ ⟨ ⟨girl⟩ ⟩ ⟩ ⟩ ⟩ ⟩

RCFG outputs the reversible context-free grammar:

$S \rightarrow NT1\ NT2$
$NT1 \rightarrow NT3\ NT4$
$NT4 \rightarrow NT5$
$NT4 \rightarrow NT6\ NT4$
$NT2 \rightarrow NT7\ NT1$
$NT3 \rightarrow$ the
$NT3 \rightarrow$ a
$NT5 \rightarrow$ girl
$NT5 \rightarrow$ cat
$NT5 \rightarrow$ dog
$NT6 \rightarrow$ young
$NT7 \rightarrow$ likes
$NT7 \rightarrow$ chases.

This grammar is isomorphic to the unknown grammar G_U.

2.8.2 Programming Language

Suppose that the learning algorithm *RCFG* is going to learn the following unknown context-free grammar G_U for a subset of the syntax for a programming language Pascal:

$$
\begin{aligned}
&\mathit{Statement} \rightarrow v \ := \ \mathit{Expression}\\
&\mathit{Statement} \rightarrow \text{while } \mathit{Condition} \text{ do } \mathit{Statement}\\
&\mathit{Statement} \rightarrow \text{if } \mathit{Condition} \text{ then } \mathit{Statement}\\
&\mathit{Statement} \rightarrow \text{begin } \mathit{Statementlist} \text{ end}\\
&\mathit{Statementlist} \rightarrow \mathit{Statement} \ ; \ \mathit{Statementlist}\\
&\mathit{Statementlist} \rightarrow \mathit{Statement}\\
&\mathit{Condition} \rightarrow \mathit{Expression} \ > \ \mathit{Expression}\\
&\mathit{Expression} \rightarrow \mathit{Term} \ + \ \mathit{Expression}\\
&\mathit{Expression} \rightarrow \mathit{Term}\\
&\mathit{Term} \rightarrow \mathit{Factor}\\
&\mathit{Term} \rightarrow \mathit{Factor} \ \times \ \mathit{Term}\\
&\mathit{Factor} \rightarrow v\\
&\mathit{Factor} \rightarrow (\ \mathit{Expression}\ ).
\end{aligned}
$$

First suppose that *RCFG* is given the sample:

$$
\begin{aligned}
&\langle\ v \ := \ \langle\ \langle\langle v\rangle\rangle \ + \ \langle\langle\langle v\rangle\rangle\rangle\ \rangle\ \rangle\\
&\langle\ v \ := \ \langle\langle\ \langle v\rangle \ \times \ \langle\langle v\rangle\rangle\ \rangle\rangle\ \rangle\\
&\langle\ v \ := \ \langle\ \langle\langle v\rangle\rangle \ + \ \langle\langle\ \langle v\rangle \ \times \ \langle\langle v\rangle\rangle\ \rangle\rangle\ \rangle\ \rangle\\
&\langle\ v \ := \ \langle\langle\ \langle\ '('\ \langle\ \langle\langle v\rangle\rangle \ + \ \langle\langle\langle v\rangle\rangle\rangle\ \rangle\ ')'\ \rangle \ \times \ \langle\langle v\rangle\rangle\ \rangle\rangle\ \rangle
\end{aligned}
$$

RCFG outputs the following reversible context-free grammar which generates the set of all assignment statements whose right-hand sides are arithmetic expressions consisting of a variable "v", the operations of addition "+" and multiplication "×" and the pair of parentheses "(" and ")":

$$
\begin{aligned}
&S \rightarrow v \ := NT1\\
&NT1 \rightarrow NT2\\
&NT1 \rightarrow NT2 \ + \ NT1\\
&NT2 \rightarrow NT3\\
&NT2 \rightarrow NT3 \ \times \ NT2\\
&NT3 \rightarrow v\\
&NT3 \rightarrow (\ NT1\ ).
\end{aligned}
$$

Next suppose that *RCFG* is given four more examples:

$$\langle \text{ while } \langle \langle\langle v\rangle\rangle\rangle > \langle\langle\langle v\rangle \times \langle\langle v\rangle\rangle\rangle\rangle \text{ do } \langle v := \langle\langle\langle v\rangle\rangle + \langle\langle\langle v\rangle\rangle\rangle\rangle \rangle \rangle$$

$$\langle \text{ if } \langle \langle\langle v\rangle\rangle\rangle > \langle\langle\langle v\rangle \times \langle\langle v\rangle\rangle\rangle\rangle \text{ then } \langle v := \langle\langle\langle v\rangle\rangle + \langle\langle\langle v\rangle\rangle\rangle\rangle \rangle \rangle$$

$$\langle \text{ begin } \langle \langle v := \langle\langle\langle v\rangle\rangle + \langle\langle\langle v\rangle\rangle\rangle\rangle\rangle ; \langle \langle v := \langle\langle\langle v\rangle \times \langle\langle v\rangle\rangle\rangle\rangle\rangle \rangle \rangle \text{ end } \rangle$$

$$\langle \text{ begin } \langle \langle v := \langle\langle\langle v\rangle \times \langle\langle v\rangle\rangle\rangle\rangle\rangle \rangle \text{ end } \rangle$$

RCFG outputs the following reversible context-free grammar isomorphic to the unknown grammar G_U:

$$\begin{aligned}
&S \rightarrow v := NT1\\
&S \rightarrow \text{while } NT4 \text{ do } S\\
&S \rightarrow \text{if } NT4 \text{ then } S\\
&S \rightarrow \text{begin } NT5 \text{ end}\\
&NT1 \rightarrow NT2\\
&NT1 \rightarrow NT2 + NT1\\
&NT2 \rightarrow NT3\\
&NT2 \rightarrow NT3 \times NT2\\
&NT3 \rightarrow v\\
&NT3 \rightarrow (NT1)\\
&NT4 \rightarrow NT1 > NT1\\
&NT5 \rightarrow S\\
&NT5 \rightarrow S ; NT5.
\end{aligned}$$

2.8.3 Inherently Ambiguous Language

Suppose that the learning algorithm *RCFG* is going to learn the following unknown context-free grammar G_U for the language $\{a^m b^m c^n d^n \mid m \geq 1, n \geq 1\} \cup \{a^m b^n c^n d^m \mid m \geq 1, n \geq 1\}$ which is known to be an inherently ambiguous context-free language ([HU79]):

$$\begin{aligned}
&S \rightarrow A\ B\\
&S \rightarrow a\ C\ d\\
&A \rightarrow a\ b\\
&A \rightarrow a\ A\ b\\
&B \rightarrow c\ d\\
&B \rightarrow c\ B\ d\\
&C \rightarrow D\\
&D \rightarrow a\ D\ d\\
&D \rightarrow E\\
&E \rightarrow b\ c\\
&E \rightarrow b\ E\ c.
\end{aligned}$$

First suppose that *RCFG* is given the sample:

$$\langle\ \langle a\ b\rangle\ \langle c\ d\rangle\ \rangle$$
$$\langle\ \langle a\ \langle a\ b\rangle\ b\rangle\ \langle c\ \langle c\ d\rangle\ d\rangle\ \rangle$$
$$\langle\ \langle a\ b\rangle\ \langle c\ \langle c\ d\rangle\ d\rangle\ \rangle$$

RCFG outputs the following reversible context-free grammar which generates the language $\{a^m b^m c^n d^n \mid m \geq 1, n \geq 1\}$:

$$S \rightarrow NT1\ NT2$$
$$NT1 \rightarrow a\ b$$
$$NT1 \rightarrow a\ NT1\ b$$
$$NT2 \rightarrow c\ d$$
$$NT2 \rightarrow c\ NT2\ d.$$

Next suppose that *RCFG* is given three more examples:

$$\langle a\ \langle\ \langle\ \langle b\ c\rangle\ \rangle\ \rangle\ d\rangle$$
$$\langle a\ \langle\ \langle a\ \langle\ \langle b\ \langle b\ c\rangle\ c\rangle\ \rangle\ d\rangle\ \rangle\ d\rangle$$
$$\langle a\ \langle\ \langle\ \langle b\ \langle b\ c\rangle\ c\rangle\ \rangle\ \rangle\ d\rangle$$

RCFG outputs the following reversible context-free grammar isomorphic to the unknown grammar G_U:

$$S \rightarrow NT1\ NT2$$
$$S \rightarrow a\ NT3\ d$$
$$NT1 \rightarrow a\ b$$
$$NT1 \rightarrow a\ NT1\ b$$
$$NT2 \rightarrow c\ d$$
$$NT2 \rightarrow c\ NT2\ d$$
$$NT3 \rightarrow NT4$$
$$NT4 \rightarrow NT5$$
$$NT4 \rightarrow a\ NT4\ d$$
$$NT5 \rightarrow b\ c$$
$$NT5 \rightarrow b\ NT5\ c.$$

Chapter 3

Learning Context-Free Grammars from Structural Queries

In this chapter, we consider the problem of learning the whole class of context-free grammars from structured strings using queries. We present an efficient algorithm for it using two types of queries: structural membership queries and structural equivalence queries. The learning protocol is based on what is called "minimally adequate teacher". We show that a grammar learned by the algorithm is not only equivalent to the unknown grammar but also structurally equivalent to it. Furthermore, the algorithm runs in time polynomial both in the number of states of the minimum tree automaton for the set of skeletal descriptions of derivation trees of the unknown grammar and in the maximum size of counter-examples returned for structural equivalence queries.

We also demonstrate that this algorithm can be applied to learning a class of logic programs, called *linear monadic logic programs*.

3.1 Learning from Queries and Summary of Recent Results

Angluin [Ang88] has considered a learning situation in which a learning algorithm has access to a fixed set of *oracles* that answer specific kinds of queries about the unknown concept. Several efficient learning algorithms using such queries have been investigated. Especially finding interesting domains which can be learned from equivalence queries and membership queries in polynomial time is one of main interests in *Algorithmic Learning Theory*.

In this framework of learning from queries, Angluin [Ang87b] has shown that the regular languages can be learned by an algorithm using membership and equivalence queries

	class of languages	*learning protocol*
Angluin [Ang87b] (1987)	regular	MAT
Bermann & Roos [BR87] (1987)	one-counter	MAT
Takada [Tak88] (1988)	even linear	MAT
Ishizaka [Ish90] (1989)	simple deterministic	MAT
Angluin [Ang87a] (1987)	context-free	MAT + nonterminal membership query
Chapter 3	context-free	MAT + structured string
Chapter 4	a subclass of context-sensitive	MAT + α
Nishino [Nis90] (1990)	a subclass of context-sensitive	MAT + α
???	context-free	MAT

Figure 3.1: Summary of recent results for polynomial-time language learning.

in time polynomial both in the number of states of the minimum deterministic finite automaton for the unknown language and in the maximum length of counter-examples returned for equivalence queries. It is still an open question whether there is a polynomial-time algorithm using membership and equivalence queries for learning the full class of context-free languages (up to the equivalence of context-free grammars). Recently Angluin and Kharitonov [AK91] have investigated cryptographic limitations on the power of membership queries to help with concept learning. They have shown that assuming the intractability of quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, there is no polynomial-time algorithm for learning context-free grammars from membership and equivalence queries. That is, the problem of learning the whole class of context-free grammars from membership and equivalence queries is computationally as hard as those cryptographic problems, for which there is currently no known polynomial-time algorithm.

Some recent results for polynomial-time learning of formal languages using queries are summarized in Figure 3.1.

3.2 The Learning Algorithm using Queries

3.2.1 Basic Idea of Learning

Angluin [Ang87b] has presented an algorithm that learns deterministic finite automata using membership and equivalence queries and runs in time polynomial both in the number of states of the minimum deterministic finite automaton equivalent to the unknown automaton and in the maximum length of counter-examples returned for equivalence queries. We will extend it to the one for learning skeletal tree automata and apply it to learning the whole class of context-free grammars from structural membership and structural equivalence queries in polynomial time.

The important data structure used in Angluin's algorithm is called an *observation table*. An observation table is a two-dimensional matrix with rows and columns labelled by strings. The entry is 0 or 1, and the intended interpretation is that the entry for row s and column e is equal to 1 if and only if the string $s \cdot e$ is accepted by the unknown automaton. Two specific observation tables are defined, which are called *closed* and *consistent*. When we have a closed, consistent observation table, we can construct the minimum deterministic finite automaton consistent with the data contained in the table in time polynomial in the size of the table. The algorithm is going to find a closed, consistent observation table by asking membership queries to fill the entries. In the next section, we will extend the observation table to the one for skeletal tree automata. The extended observation table has rows labelled by skeletons and columns labelled by elements of $(Sk \cup \Sigma)^T_{\$}$. The intended interpretation is that the entry for row s and column e is equal to 1 if and only if the skeleton $s\#e$ is a structured string of the unknown grammar G.

The idea of the observation table is also related to the state characterization matrix by Gold [Gol78].

In this chapter, unless otherwise stated, we will mean a "deterministic tree automaton" by simply saying "tree automaton".

3.2.2 Observation Tables

We extend the observation table. An observation table is the data structure used in the learning algorithm to organize the information about a finite collection of skeletons with the indication whether they are structured strings of the unknown grammar or not.

Let B be a finite set of skeletons with depth at least 1 and C be a finite subset of $(Sk \cup \Sigma)^T_{\$}$. B is called *subtree-closed* if $s \in B$ implies that all subtrees with depth at

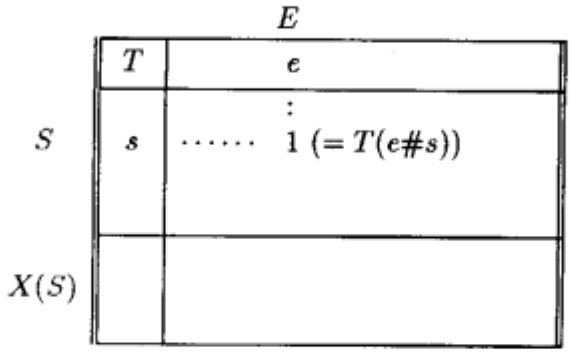

Figure 3.2: Observation table (S, E, T).

least 1 of s are elements of B. C is called *\$-prefix-closed with respect to B* if $e \in C - \{\$\}$ implies that there exists an e' in C such that $e = e'\#\sigma(s_1, \ldots, s_{i-1}, \$, s_i, \ldots, s_{k-1})$ for some $s_1, \ldots, s_{k-1} \in B \cup \Sigma$ and $i \in \mathbf{N}$.

Definition Let S be a nonempty finite subtree-closed set of skeletons with depth at least 1, $X(S) = \{\sigma(u_1, \ldots, u_k) \mid \sigma \in Sk_k,\ u_1, \ldots, u_k \in S \cup \Sigma \text{ and } \sigma(u_1, \ldots, u_k) \notin S \text{ for } k \geq 1\}$, E be a nonempty finite subset of $(Sk \cup \Sigma)^T_\$$ which is \$-prefix-closed with respect to S, and T be a function mapping $(E\#(S \cup X(S)))$ to $\{0, 1\}$. The interpretation of this is that $T(s)$ is 1 if and only if s is a structured string of the unknown grammar G. An *observation table*, denoted (S, E, T), is a two-dimensional matrix with rows labelled by elements of $(S \cup X(S))$, columns labelled by elements of E, and the entry for row s and column e equal to $T(e\#s)$.

An observation table can be visualized as in Figure 3.2. The learning algorithm uses the observation table to build a skeletal tree automaton. Rows labelled by elements of S are the candidates for states of the skeletal tree automaton being constructed, and columns labelled by elements of E are used to distinguish these states. Rows labelled by elements of $X(S)$ are used to construct the transition function.

If s is an element of $(S \cup X(S))$, $row(s)$ denotes the function f from E to $\{0, 1\}$ defined by $f(e) = T(e\#s)$.

Definition An observation table (S, E, T) is called *closed* if every $row(x)$ of $x \in X(S)$ is identical to some $row(s)$ of $s \in S$. An observation table is called *consistent* if whenever s_1 and s_2 are elements of S such that $row(s_1) = row(s_2)$, $row(\sigma(u_1, \ldots, u_{i-1}, s_1, u_i, \ldots, u_{k-1}))$ $= row(\sigma(u_1, \ldots, u_{i-1}, s_2, u_i, \ldots, u_{k-1}))$ for all $\sigma \in Sk$, $u_1, \ldots, u_{k-1} \in S \cup \Sigma$ and $1 \leq i \leq k$.

Definition Let (S, E, T) be a closed, consistent observation table. The *corresponding skeletal tree automaton* $A(S, E, T)$ over $Sk \cup \Sigma$ constructed from (S, E, T) is defined with the state set Q, the set of final states F, and the state transition function δ as follows.

$$Q = \{row(s) \mid s \in S\},$$
$$F = \{row(s) \mid s \in S \text{ and } T(s) = 1\},$$
$$\delta_k(\sigma, row(s_1), \ldots, row(s_k)) = row(\sigma(s_1, \ldots, s_k)) \quad \text{for } s_1, \ldots, s_k \in S \cup \Sigma,$$

where the function row is augmented to be $row(a) = a$ for $a \in \Sigma$.

We can see that this is a well-defined (deterministic) skeletal tree automaton. Let s_1 and s_2 be elements of S such that $row(s_1) = row(s_2)$. Then since E contains \$, $T(s_1) = T(\$\#s_1)$ and $T(s_2) = T(\$\#s_2)$ are defined and equal to each other. Hence F is well-defined. Since the observation table (S, E, T) is consistent, for $u_1, \ldots, u_{k-1} \in S \cup \Sigma$, $row(\sigma(u_1, \ldots, u_{i-1}, s_1, u_i, \ldots, u_{k-1})) = row(\sigma(u_1, \ldots, u_{i-1}, s_2, u_i, \ldots, u_{k-1}))$ $(0 \leq i \leq k)$, and since it is closed, this value is equal to $row(s)$ for some s in S. Hence δ is well-defined.

The lemmas and theorems that follow are analogous to Angluin's results.

Lemma 3.1 *Suppose that (S, E, T) is a closed, consistent observation table. For the transition function δ of the skeletal tree automaton $A(S, E, T)$ and for each s in $(S \cup X(S))$, we have $\delta(s) = row(s)$.*

Proof. It is clear from the definition of $A(S, E, T)$. □

Lemma 3.2 *Suppose that (S, E, T) is a closed, consistent observation table. Then the skeletal tree automaton $A(S, E, T)$ is consistent with the finite function T. That is, for every s in $(S \cup X(S))$ and e in E, $\delta(e\#s)$ is in F if and only if $T(e\#s) = 1$.*

Proof. We prove it by induction on the depth of the node labelled \$ in e. When e is \$ and s is any element of $(S \cup X(S))$, by Lemma 3.1, $\delta(e\#s) = \delta(s) = row(s)$. If s is in S, then by the definition of F, $row(s)$ is in F if and only if $T(s) = 1$. If s is in $X(S)$, then since (S, E, T) is closed, $row(s) = row(s')$ for some s' in S, and $row(s')$ is in F if and only if $T(s') = 1$, which is true if and only if $T(s) = 1$.

Next suppose that the result holds for all $e \in E$ in which the depth of the node labelled \$ is at most h. Let e be an element of E where the depth of the node labelled \$ is $h + 1$. Since E is \$-prefix-closed with respect to S, $e = e'\#\sigma(s_1, \ldots, s_{i-1}, \$, s_i, \ldots, s_{k-1})$ for some $s_1, \ldots, s_{k-1} \in S \cup \Sigma$, $i \in \mathbf{N}$ and $e' \in E$ in which the depth of the node labelled \$ is h. For

any element s of $(S \cup X(S))$, since (S, E, T) is closed, there is an element s' in S such that $\delta(s) = \delta(s')$. Therefore

$$\begin{aligned}\delta(e\#s) &= \delta(e'\#\sigma(s_1, \ldots, s_{i-1}, \$, s_i, \ldots, s_{k-1})\#s) \\ &= \delta(e'\#\sigma(s_1, \ldots, s_{i-1}, \$, s_i, \ldots, s_{k-1})\#s'), \\ &\quad \text{by the replacement lemma} \\ &= \delta(e'\#\sigma(s_1, \ldots, s_{i-1}, s', s_i, \ldots, s_{k-1})).\end{aligned}$$

By the induction hypothesis, $\delta(e'\#\sigma(s_1, \ldots, s_{i-1}, s', s_i, \ldots, s_{k-1}))$ is in F if and only if $T(e'\#\sigma(s_1, \ldots, s_{i-1}, s', s_i, \ldots, s_{k-1})) = T(e\#s') = 1$. Since $row(s) = row(s')$, $T(e\#s') = T(c\#s)$. Hence $\delta(e\#s)$ is in F if and only if $T(c\#s) = 1$. □

Lemma 3.3 *Suppose that (S, E, T) is a closed, consistent observation table, and the skeletal tree automaton $A(S, E, T) = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ has n states. If $A' = (Q', \{\sigma\} \cup \Sigma, \delta', F')$ is any skeletal tree automaton consistent with T that has n or fewer states, then A' is isomorphic to $A(S, E, T)$.*

Proof. We prove it by exhibiting an isomorphism φ from $A(S, E, T)$ to A'. First define for any $s \in S \cup X(S)$ $\varphi(row(s)) = \delta'(s)$. Since A' is consistent with T, φ is one-to-one mapping from Q to Q'. Hence A' has n states and φ is a bijection. We must verify that it preserves the transition function, and that it carries F to F'. For each $s_1, \ldots, s_k \in S \cup \Sigma$,

$$\begin{aligned}\varphi(\delta_k(\sigma, row(s_1), \ldots, row(s_k))) &= \varphi(row(\sigma(s_1, \ldots, s_k))) \\ &= \delta'(\sigma(s_1, \ldots, s_k)).\end{aligned}$$

Also for $q_i = \varphi(row(s_i))$ if $s_i \in S$ and $q_i = s_i$ if $s_i \in \Sigma$ $(1 \leq i \leq k)$,

$$\begin{aligned}\delta'_k(\sigma, q_1, \ldots, q_k) &= \delta'_k(\sigma, \delta'(s_1), \ldots, \delta'(s_k)) \\ &= \delta'(\sigma(s_1, \ldots, s_k)).\end{aligned}$$

Lastly since A' is consistent with T, for $s \in S$, $row(s)$ is in F if and only if $T(s) = 1$ if and only if $\delta'(s)$ is in F' if and only if $\varphi(row(s))$ is in F'. Thus φ maps F to F'. Hence we conclude that the mapping φ is an isomorphism from $A(S, E, T)$ to A'. □

3.2.3 The Learning Algorithm *CFGQ*

Now we describe and analyze the algorithm *CFGQ* to learn context-free grammars from structured strings using queries.

ALGORITHM *CFGQ*

Input: Oracles that answer s-m queries and s-e-queries about G_U.

Output: A context-free grammar G.

Procedure:

1 $S{:=}\phi$;
2 $E{:=}\{\$\}$;
3 Let $G = (\{S\}, \Sigma, \phi, S)$;
(i.e., G is the context-free grammar with the empty set of productions);
4 Make an s-e-query proposing G;
5 **While** the reply is *no*
6 add the counter-example t and all its subtrees with depth at least 1 to S;
7 extend T to $E\#(S \cup X(S))$ using s-m-queries;
8 **While** (S, E, T) is not closed or not consistent;
9 **If** (S, E, T) is not consistent **then**
10 find s_1 and s_2 in S, $e \in E$, $u_1, \ldots, u_{k-1} \in S \cup \Sigma$, and $i \in \mathbf{N}$ such that
$row(s_1)$ is equal to $row(s_2)$ and
$T(e\#\sigma(u_1, \ldots, u_{i-1}, s_1, u_i, \ldots, u_{k-1})) \neq T(e\#\sigma(u_1, \ldots, u_{i-1}, s_2, u_i, \ldots, u_{k-1}))$;
11 add $e\#\sigma(u_1, \ldots, u_{i-1}, \$, u_i, \ldots, u_{k-1}))$ to E;
12 extend T to $E\#(S \cup X(S))$ using s-m-queries;
13 **If** (S, E, T) is not closed **then**
14 find $s_1 \in X(S)$ such that $row(s_1)$ is different from $row(s)$ for all $s \in S$;
15 add s_1 to S;
16 extend T to $E\#(S \cup X(S))$ using s-m-queries;
17 **end**;
18 Once (S, E, T) is closed and consistent, let $G{:=}G(A(S, E, T))$;
19 Make an s-e-query proposing G;
20 **end**;
21 Halt and output G;

where

the operation of "extend T to $E\#(S \cup X(S))$ using s-m-queries" means the operation to extend T by asking s-m-queries for missing elements.

Figure 3.3: The learning algorithm *CFGQ* for Context-Free Grammars

Suppose G_U is the unknown grammar to be learned (up to structural equivalence). We assume that the terminal alphabet Σ and the skeletal alphabet Sk for G_U are known.

A *structural membership query* (*s-m-query*, for short) proposes a skeleton s and asks whether it is in $s(D(G_U))$. The answer is either *yes* or *no*. A *structural equivalence query* (*s-e-query*, for short) proposes a grammar G' and asks whether $s(D(G_U)) = s(D(G'))$. The answer is *yes* or *no*. If it is *no*, then it provides a *counter-example*, that is, a skeleton s in the symmetric difference of $s(D(G_U))$ and $s(D(G'))$.

Note that the problem of structural equivalence of context-free grammars is solvable, whereas the problem of equivalence of context-free grammars is unsolvable. Further the problem of testing two context-free grammars for structural equivalence may be solved by a computation polynomial in the sum of their sizes. So there is a computable implementation of a minimally adequate teacher for these two types of queries.

The learning algorithm *CFGQ* is illustrated in Figure 3.3.

Note that in the skeletal tree automaton, the transition function must specify the state assigned to each d-tuple of elements from $Q \cup V_0$, that is, at least n^d different d-tuples. In practice, the transition function is likely to be "sparse" in the sense that it assigns the "dead state" to most of these combinations. However the learning algorithm in its present form does not take advantage of sparsity. Perhaps it is possible to construct a more efficient version of the learning algorithm *CFGQ* to take advantage of the sparseness.

3.3 An Example Run

Suppose that the unknown context-free grammar is $G_U = (N, \Sigma, P, S)$ which generates the set of all valid arithmetic expressions involving a variable "v", the operations of multiplication "$\times$" and addition "$+$", and the parentheses "[" and "]" :

$$\begin{aligned} N &= \{S, E, F\}, \\ \Sigma &= \{v, \times, +, [,]\}, \\ P &= \{S \to E, \\ &\quad E \to F, \\ &\quad E \to F + E, \\ &\quad F \to v, \\ &\quad F \to v \times F, \\ &\quad F \to [E]\}. \end{aligned}$$

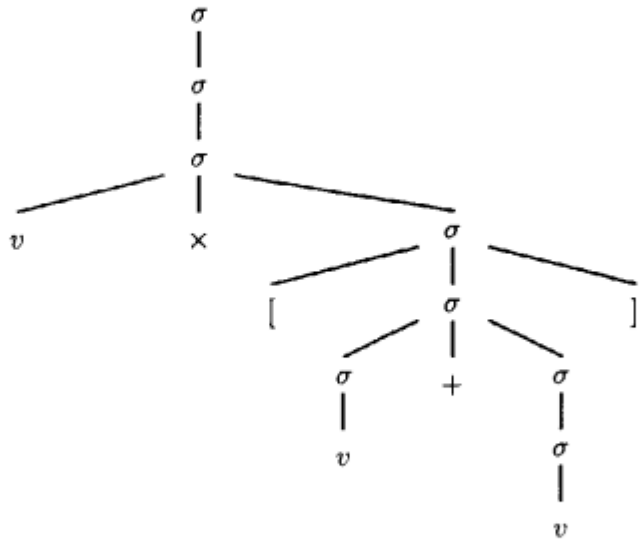

Figure 3.4: The structural description for "$v \times [v + v]$".

First the learning algorithm *CFGQ* proposes the context-free grammar $G = (\{S\}, \Sigma, \phi, S)$ and we assume that the counter-example shown in Figure 3.4 is returned for the s-e-query. This is the skeletal description of the derivation tree for the sentence "$v \times [v+v]$" assigned by G_U.

CFGQ adds all subtrees with depth at least 1 of it to S and divides them into two parts; one for those $s \in S$ with $row(s) = 0$ and the other for those s with $row(s') = 1$, by asking s-m-queries.

Next *CFGQ* tries to make a closed, consistent observation table by asking s-m-queries. In this process, *CFGQ* finds the observation table to be not consistent once, and hence it adds the element $\sigma(\$)$ to E. Then *CFGQ* eventually makes the closed, consistent observation table shown in Figure 3.5 and outputs the first conjecture of the context-free grammar G_1 shown in Figure 3.6.

The derivation tree for the sentence "$v \times [v + v]$" by G_1 is shown in Figure 3.7.

However G_1 is not structurally equivalent to G_U, and a counter-example is returned for the s-e-query. In this case, we assume that the counter-example $\sigma(\sigma(\sigma(\sigma(\sigma(\sigma(v))))))$ is selected; it is in $s(D(G_1))$ but not in $s(D(G_U))$. Then *CFGQ* eventually makes another closed, consistent observation table shown in Figure 3.8, and outputs the second conjecture whose reduced version G_2 is shown in Figure 3.9 by eliminating the meaning-

	T	\$	$\sigma(\$)$
S	$\sigma(v)$	0	0
	$\sigma(\sigma(v))$	0	1
	$\sigma(\sigma(v),+,\sigma(\sigma(v)))$	0	1
	$\sigma([,\sigma(\sigma(v),+,\sigma(\sigma(v))),])$	0	0
	$\sigma(v,\times,\sigma([,\sigma(\sigma(v),+,\sigma(\sigma(v))),]))$	0	0
	$\sigma(\sigma(v,\times,\sigma([,\sigma(\sigma(v),+,\sigma(\sigma(v))),])))$	0	1
	$\sigma(\sigma(\sigma(v,\times,\sigma([,\sigma(\sigma(v),+,\sigma(\sigma(v))),]))))$	1	0
$X(S)$	...	...	...

Figure 3.5: Observation table.

$$
\begin{aligned}
G_1 &= (N_1, \Sigma, P_1, S_1) \\
N_1 &= \{\langle 10\rangle, \langle 01\rangle, \langle 00\rangle\}, \\
P_1 &= \{\langle 00\rangle \to v, \\
&\quad \langle 01\rangle \to \langle 00\rangle, \\
&\quad \langle 01\rangle \to \langle 00\rangle + \langle 01\rangle, \\
&\quad \langle 00\rangle \to [\langle 01\rangle], \\
&\quad \langle 00\rangle \to v \times \langle 00\rangle, \\
&\quad \langle 10\rangle \to \langle 01\rangle, \\
&\quad S_1 \to \langle 01\rangle, \\
&\quad \langle 00\rangle \to \langle 10\rangle, \\
&\quad \langle 00\rangle \to v \times \langle 01\rangle, \cdots\}
\end{aligned}
$$

Figure 3.6: The first conjecture G_1.

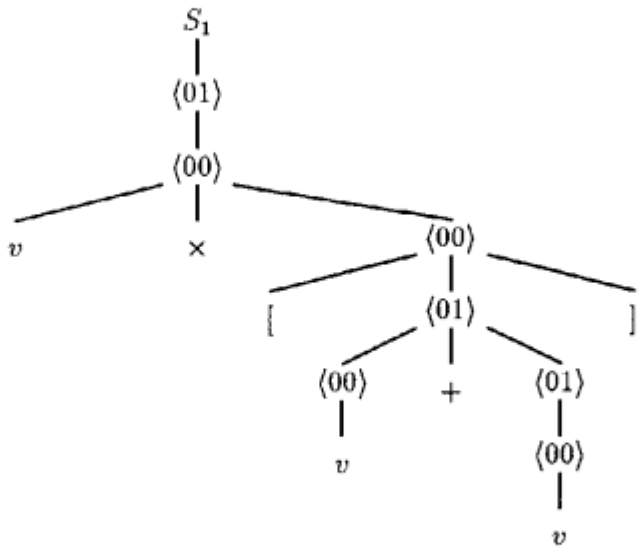

Figure 3.7: The derivation tree for "$v \times [v + v]$" by G_1.

less nonterminals and all productions including them. The grammar G_2 is structurally equivalent to G_U.

In this example run, the value $\sigma(\sigma(\$, +, \sigma(\sigma(v))))$ is taken as the third column of E. However this is one possible choice of a distinguishing environment. The simpler environment $\sigma(\sigma(\$))$ would also work.

The derivation tree for the sentence "$v \times [v + v]$" by G_2 is shown in Figure 3.10.

3.4 Correctness and Time Complexity of *CFGQ*

Now we will see that *CFGQ* eventually terminates and makes a correct conjecture, that is, outputs a grammar structurally equivalent to G_U. It is clear that if *CFGQ* ever terminates, its output is a grammar structurally equivalent to G_U. Let A_U be the minimum skeletal tree automaton for $s(D(G_U))$ and n be the number of states in it.

Lemma 3.4 *The conjectures $G(A(S, E, T))$ that the algorithm CFGQ makes are consistent with T. That is, for every s in $(S \cup X(S))$ and e in E, $e\#s \in s(D(G(A(S, E, T))))$ if and only if $T(e\#s) = 1$. (See Section 2.4 for the definition of $G(A)$ for a deterministic skeletal tree automaton A.)*

	T	\$	$\sigma(\$)$	$\sigma(\sigma(\$, +, \sigma(\sigma(v))))$
	$\sigma(v)$	0	0	1
	$\sigma(\sigma(v))$	0	1	0
	$\sigma(\sigma(v), +, \sigma(\sigma(v)))$	0	1	0
	$\sigma([, \sigma(\sigma(v), +, \sigma(\sigma(v))),])$	0	0	1
	$\sigma(v, \times, \sigma([, \sigma(\sigma(v), +, \sigma(\sigma(v))),]))$	0	0	1
S	$\sigma(\sigma(v, \times, \sigma([, \sigma(\sigma(v), +, \sigma(\sigma(v))),])))$	0	1	0
	$\sigma(\sigma(\sigma(v, \times, \sigma([, \sigma(\sigma(v), +, \sigma(\sigma(v))),])))))$	1	0	0
	$\sigma(\sigma(\sigma(v)))$	1	0	0
	$\sigma(\sigma(\sigma(\sigma(v))))$	0	0	0
	$\sigma(\sigma(\sigma(\sigma(\sigma(v)))))$	0	0	0
	$\sigma(\sigma(\sigma(\sigma(\sigma(v)))))$	0	0	0
$X(S)$	...	...	...	...

Figure 3.8: Observation table.

$$
\begin{aligned}
G_2 &= (N_2, \Sigma, P_2, S_2) \\
N_2 &= \{\langle 100\rangle, \langle 010\rangle, \langle 001\rangle\}, \\
P_2 &= \{\langle 001\rangle \to v, \\
&\quad \langle 010\rangle \to \langle 001\rangle, \\
&\quad \langle 010\rangle \to \langle 001\rangle + \langle 010\rangle, \\
&\quad \langle 001\rangle \to [\langle 010\rangle], \\
&\quad \langle 001\rangle \to v \times \langle 001\rangle, \\
&\quad S_2 \to \langle 010\rangle\}
\end{aligned}
$$

Figure 3.9: The second conjecture G_2.

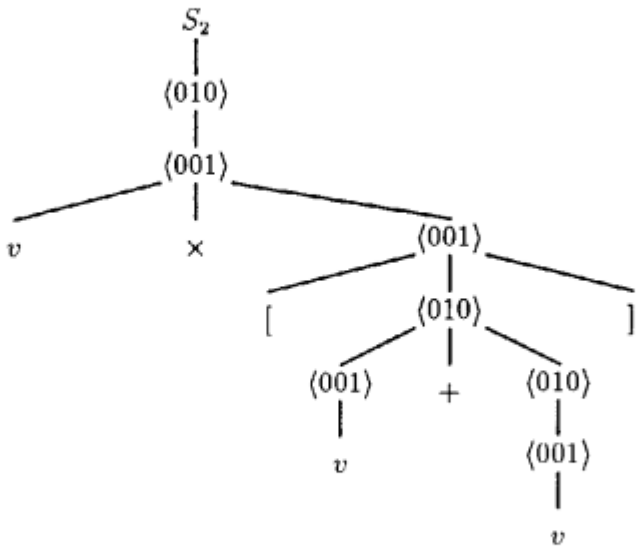

Figure 3.10: The derivation tree for "$v \times [v + v]$" by G_2.

Proof. Firstly we show that S is always subtree-closed and E is always \$-prefix-closed with respect to S. In *CFGQ*, there are three operations which extend S or E. When t and all its subtrees with depth at least 1 are added to S, S obviously remains subtree-closed. If (S, E, T) is not consistent, then for some $e \in E$, $u_1, \ldots, u_{k-1} \in S \cup \Sigma$ and $i \in \mathbf{N}$, $e\#\sigma(u_1, \ldots, u_{i-1}, \$, u_i, \ldots, u_{k-1}))$ is added to E. In this case, E remains \$-prefix-closed with respect to S. If (S, E, T) is not closed, then for some $u_1, \ldots, u_k \in S \cup \Sigma$, $\sigma(u_1, \ldots, u_k)$ is added to S. In this case, S remains subtree-closed.

Whenever *CFGQ* makes a conjecture, the observation table (S, E, T) is found to be closed and consistent. Hence by Lemma 3.2 and Theorem 2.4, $G(A(S, E, T))$ is consistent with T. □

Lemma 3.5 *The algorithm CFGQ terminates.*

Proof. Firstly we show that whenever an observation table (S, E, T) is not consistent or not closed, the number of distinct values $row(s)$ for $s \in S$ must increase. If (S, E, T) is not consistent, then since some two previously equal row values are no longer equal after E is augmented, the number of distinct values $row(s)$ increases by at least one. If (S, E, T) is not closed and some element t in $X(S)$ is added to S, then since $row(t)$ is different from $row(s)$ for all s in S before S is augmented, the number of distinct values $row(s)$ increases by at least one.

Next we will show that whenever a counter-example t and all its subtrees with depth at least 1 are added to S because $G(A(S,E,T))$ is incorrect, the skeletal tree automaton $A(S',E',T')$ for the next conjecture $G(A(S',F',T'))$ must have at least one more state than $A(S,E,T)$. Since $A(S',E',T')$ is consistent with T and inequivalent to $A(S,E,T)$ (since they disagree on t by Theorem 2.4), by Lemma 3.3, $A(S',E',T,)$ has at least one more state than $A(S,E,T)$.

Since A_U is always consistent with T, by Lemma 3.3, the number of distinct values $row(s)$ cannot be more than n. Thus *CFGQ* always eventually finds a closed, consistent observation table and makes a conjecture. Furthermore a counter-example is added to S at most n times. Hence the algorithm *CFGQ* terminates after making at most n conjectures and by Lemma 3.4, outputs a correct conjecture. □

Next we will analyze the time complexity of the algorithm *CFGQ*. That depends partly on the size of the counter-examples returned for structural equivalence queries, where the *size* of a counter-example t is the number of nodes in t, i.e. $|Dom_t|$. We will analyze the running time of the the algorithm *CFGQ* as a function of the number n of states in the minimum skeletal tree automaton for $s(D(G_U))$ and the maximum size m of counter-examples returned for structural equivalence queries during the running of *CFGQ*. We will show that its running time is bounded by a polynomial in m and n. Let k be the cardinality of the terminal alphabet Σ, l be the cardinality of the skeletal alphabet Sk (that is the number of distinct ranks of the symbol σ) and d be the maximum rank of the symbol σ in Sk.

Whenever (S,E,T) is found to be not closed, one element is added to S. Whenever (S,E,T) is found to be not consistent, one element is added to E. For each counter-example of size at most m returned for a structural equivalence query, at most m subtrees are added to S. Since the observation table is found to be not consistent at most $n-1$ times, the total number of elements in E cannot exceed n. Since the observation table is found to be not closed at most $n-1$ times, and since there can be at most n counter-examples, the total number of elements in S cannot exceed $n+mn$. Thus, the maximum cardinality of $E\#(S \cup X(S))$ is at most

$$((n+mn)+l(n+mn+k)^d)n \;=\; O(m^d n^{d+1}).$$

Now we consider the operations performed by *CFGQ*. Checking the observation table whether it is closed and consistent or not can be done in time polynomial in the size of the observation table, and must be done at most n times. Adding an element to S or E

requires at most $O(m^d n^d)$ structural membership queries to extend T for missing elements. When the observation table is closed and consistent, $A(S,E,T)$ and $G(A(S,E,T))$ may be constructed in time polynomial in the size of the observation table, and this must be done at most n times. A counter-example requires the addition of at most m subtrees to S, and this can also happen at most n times. Therefore, the total running time of *CFGQ* can be bounded by a polynomial function of m and n. Thus we have the following results.

Theorem 3.6 *Using structural equivalence and structural membership queries for an unknown context-free grammar G_U, the learning algorithm CFGQ eventually terminates and outputs a grammar structurally equivalent to G_U. Moreover, the total running time of CFGQ is bounded by a polynomial in m and n, where n is the the number of states of the minimum skeletal tree automaton for $s(D(G_U))$ and m is the maximum size of counter-examples returned for structural equivalence queries.*

A *parenthesis grammar* is a context-free grammar $G = (N, \Sigma, P, S)$ such that the productions in P are restricted to the form $A \rightarrow \langle\alpha\rangle$, where $\langle$ and $\rangle$ are special symbols not in Σ and α contains neither $\langle$ nor $\rangle$.

Since the structural information can be obtained from strings of a parenthesis grammar, we can apply the result to the learning problem of parenthesis languages.

Corollary 3.7 *There is an algorithm such that for any parenthesis grammar G it learns a parenthesis grammar equivalent to G by using equivalence and membership queries and it runs in time polynomial both in the number of states of the minimum skeletal tree automaton for $s(D(G))$ and in the maximum length of counter-examples.*

3.5 Application to Learning Logic Programs

In this section, we argue that the algorithm *CFGQ* can be applied to learning a class of logic programs, called *linear monadic logic programs*, from membership and equivalence queries in a polynomial computation time.

The study of learning logic programs from examples was initially and mostly done by E.Shapiro and his work is known as *Model Inference System* [Sha82, Sha81]. He devises a program that identifies first order sentences (Horn clauses) from examples of their logical consequences. The target of the inference is an Herbrand model. Thus Shapiro's algorithm (especially the diagnosis algorithm) deeply depends on the theory of predicate logic and logic programming. In the theory of logic programming, the least Herbrand

model $\cap M(LP)$ of a logic program LP is taken as the mathematical semantics, called *model-theoretic semantics*, for it. This semantics provides the denotation of a predicate symbol p in a logic program LP :

$$D(p) = \{(t_1, \ldots, t_k) \mid p(t_1, \ldots, t_k) \in \cap M(LP)\}.$$

$D(p)$ is the denotation of p as determined by model-theoretic semantics. Thus model-theoretic semantics gives a characterization of the set of terms computed by a logic program.

On the other hand, algebraic semantics which connects between the theory of tree languages and the semantics of programming languages is now well known and recently introduced to logic programming in [MP83]. It studies the use of tree languages in the semantics of logic programming. In algebraic semantics, the set of terms computed by a logic program LP can be viewed as a tree language. For example, the denotation of a monadic predicate p, $D(p) = \{t \mid p(t) \in \cap M(LP)\}$, is viewed as a tree language. From the result in [MP83], a set of trees is rational if and only if it is computed by a linear monadic logic program, where a *rational* set of trees is a set of trees which can be recognized by some tree automaton and a *linear monadic logic program* is a class of logic programs defined by syntactic restrictions such that predicate symbols are monadic, the height of terms involved is less than or equal to 1 and no variable in a term has more than one occurrence. Therefore, the denotation of p can be written as $D(p) = \{t \mid t$ is accepted by the tree automaton about p in $LP\}$. Based on such an algebraic semantics, we can establish a new learning schema of logic programs so that the problem of learning logic programs is reduced to the problem of learning tree automata. Then by employing the learning algorithm *CFGQ* for tree automata, we can get an efficient algorithm for learning the class of linear monadic logic programs from membership and equivalence queries. Our learning algorithm also gives a partial solution to the problem of the "Theoretical Terms" in Model Inference System. See [Sak90b] for more details.

Chapter 4

Learning Elementary Formal Systems from Queries

In this chapter, we introduce a new class of representations for formal languages in the framework of Smullyan's elementary formal systems [Smu61] for the problem of learning formal languages. The new class of representations is a natural extension of context-free grammars, and the languages defined by these representations lie between context-free languages and context-sensitive languages and contain some important classes of formal languages such as Angluin's pattern languages [Ang80a]. We demonstrate a polynomial-time algorithm for learning these representations using some reasonable queries. This implies that there exists a larger class of formal languages than the class of context-free languages that is efficiently learnable by using some reasonable queries. Our algorithm may be viewed as a natural and powerful extension of Angluin's algorithm [Ang87a].

4.1 Learning a Subclass of Context-Sensitive Languages

Recently there have been two approaches to the problem of learning context-free languages from membership and equivalence queries: one is to show that some subclass of context-free languages can be learned from membership and equivalence queries in polynomial time, and the other is to show that the full class of context-free languages can be learned from membership and equivalence queries plus some additional information (such as nonterminal membership queries [Ang87a], or information on the grammatical structure in Chapter 3) in polynomial time.

There now seem to be two directions to investigate. The first is to study the problem of an efficient learning method for the full class of context-free languages simply from

membership and equivalence queries. The second is to study the problem of an efficient learning method for a larger class of formal languages than context-free languages from membership and equivalence queries plus some additional information. In this chapter, we take the second direction and study the problem of learning a subclass of context-sensitive languages from membership and equivalence queries plus some additional information. In the course of this study, we claim the importance of representations of what is learned and introduce a new class of representations for formal languages in the framework of Smullyan's elementary formal systems. Smullyan's elementary formal system has recently got much attention [ASY89, Shi90] as a unifying framework for formal language learning.

4.2 Smullyan's Elementary Formal Systems

We explain the notion of Smullyan's elementary formal systems [Smu61] and define their languages. Let Σ be an alphabet of terminal symbols and let the elements in it be denoted by $a, b, c, \ldots$. Let V be a countable set of symbols and D be an alphabet, where Σ, V and D are mutually disjoint. Elements in V are called *variables* and denoted by $x, y, z, x_1, x_2, \ldots$, and elements in D are called *predicates* and denoted by $p, r, p_1, p_2, \ldots, r_1, r_2, \ldots$, each of which is assigned a unique positive integer called its *degree*.

Definition An *elementary formal system* (EFS, for short) over an alphabet Σ is a quadruple $E = (D, \Sigma, M, p)$, where p is a predicate in D and M is a finite set of expressions called (*well-formed*) *formulas* defined below:

1. A *term* t of E is a string in $(\Sigma \cup V)^*$, and by $V(t)$ we denote the set of variables that occur in the term t and by $t(x_1, \ldots, x_n)$ a term that exactly contains the variables $x_1, \ldots, x_n$. (The variables are not necessarily distinct.)

2. An *atomic formula* of E is an expression of the form $r(t_1, t_2, \ldots, t_m)$, where r is a predicate in D with degree m and $t_1, t_2, \ldots, t_m$ are terms of E. If $t_1, t_2, \ldots, t_m$ are terminal strings in Σ^*, then $r(t_1, t_2, \ldots, t_m)$ is said to be *ground*.

3. A (*well-formed*) *formula* of E is an expression of the form

$$R \leftarrow R_1 \& R_2 \& \cdots \& R_n \qquad (n \geq 0)$$

where $R, R_1, R_2, \ldots, R_n$ are atomic formulas of E, and $R_1, R_2, \ldots, R_n$ are called the *premises* of the formula and R is called the *conclusion* of the formula.

We call formulas in M *axioms of* E.

Note that the definition of EFS allows a formula with an empty premise, that is, a formula of the form $R \leftarrow$. In the following, we assume that all predicates are monadic, that is, that the degrees of predicates are all one, because predicates with degree one are sufficient for our purposes.

Let θ be any homomorphism (with respect to concatenation) from terms to terms. We denote the image of a term t by $t\theta$. If a homomorphism θ maps any terminal symbol a in Σ to itself, that is, if $a\theta = a$ for all $a \in \Sigma$, then θ is called a *substitution*. For a formula $F = p(t) \leftarrow p_1(t_1)\& \cdots \& p_n(t_n)$, we define $F\theta = p(t\theta) \leftarrow p_1(t_1\theta)\& \cdots \& p_n(t_n\theta)$.

Definition We say that a formula $F = p(t) \leftarrow p_1(t_1)\& \cdots \& p_{n-1}(t_{n-1})$ is *provable from* E if F satisfies one of the following conditions:

1. F is in M.
2. $F = F'\theta$ for some formula F' provable from E and some substitution θ.
3. There exists an atomic formula $p(t_n)$ such that two formulas $p(t) \leftarrow p_1(t_1)\& \cdots \& p_n(t_n)$ and $p_n(t_n) \leftarrow$ are provable from E.

We say that a formula F is *provable from* E *in* n *steps* if F is provable from E by applying any of the above three rules n times. We say that an atomic formula $p(t)$ is *provable from* E if the formula of the form $p(t) \leftarrow$ is provable from E. The *language* defined by an EFS $E = (D, \Sigma, M, p)$, denoted $L(E)$, is the set $\{w \mid w \text{ is in } \Sigma^* \text{ and } p(w) \text{ is provable from } E\}$. Two EFSs E and E' are said to be *equivalent* if and only if $L(E) = L(E')$.

Theorem 4.1 (Arikawa [Ari70]) *The languages generated by phrase-structure grammars (type 0 grammars) are defined by elementary formal systems and vice versa.*

4.3 Extended Simple Formal Systems

In this section, we introduce three restricted forms of EFSs, state their relations to other classes of formal languages and show some of their closure and nonclosure properties.

4.3.1 Restrictions of EFS

We first define a class of restricted EFSs that precisely define context-free languages.

Definition An EFS $E = (D, \Sigma, M, p)$ is called a *context-free form* if

1. Each axiom of E is of the form

$$r(t) \leftarrow r_1(x_1)\& \cdots \& r_n(x_n),$$

where $V(t) = \{x_1, \ldots, x_n\}$,

2. $x_1, \ldots, x_n$ are distinct variables, and

3. In the term t, each of the variables $x_1, \ldots, x_n$ occurs precisely once.

As we will show later, the languages defined by context-free forms of EFSs are precisely the context-free languages.

In the definition of context-free forms of EFSs, the axioms are restricted to the form $p(t(x_1, \ldots, x_n)) \leftarrow p_1(x_1)\& \cdots \& p_n(x_n)$, and there are two further syntactic restrictions 2 and 3 on the form. By relaxing condition 3, we get the following class of restricted EFSs.

Definition An EFS $E = (D, \Sigma, M, p)$ is called a *simple formal system* (SFS, for short) if

1. Each axiom of E is of the form

$$r(t) \leftarrow r_1(x_1)\& \cdots \& r_n(x_n),$$

where $V(t) = \{x_1, \ldots, x_n\}$, and

2. $x_1, \ldots, x_n$ are distinct variables.

By relaxing conditions 2 and 3, we get the following class of restricted EFSs.

Definition An EFS $E = (D, \Sigma, M, p)$ is called an *extended simple formal system* (ESFS, for short) if each axiom of E is of the form

$$r(t) \leftarrow r_1(x_1)\& \cdots \& r_n(x_n),$$

where $V(t) = \{x_1, \ldots, x_n\}$.

Note that from the definitions of context-free forms, SFSs, and ESFSs, the conclusion of an axiom that has no premise becomes ground.

The class of simple formal systems has been introduced by Arikawa [Ari70]. The class of languages defined by SFSs properly contains the class of context-free languages and is properly contained in the class of context-sensitive languages.

The *size* of the ESFS $E = (D, \Sigma, M, p)$, denoted $size(E)$, is the sum of $|D|$, $|\Sigma|$, $|M|$, plus the sum of the lengths of the terms in the conclusions of all the axioms in M.

Example 4.1 For example, the formula

$$p(axbyc) \leftarrow r_1(x)\&r_2(y)$$

is in context-free form of EFS. The formula

$$p(xax) \leftarrow r(x)$$

is not allowed in context-free form of EFS, while it is in SFSs. The formula

$$p(x) \leftarrow r_1(x)\&r_2(x)$$

is not allowed in SFSs, while it is defined in ESFSs.

Example 4.2

1. Suppose $\Sigma_1 = \{a, b, c\}$ and $D_1 = \{p, p_1, p_2, p_3, r_1, r_2, r_3\}$. Let $E_1 = (D_1, \Sigma_1, M_1, p)$ be an ESFS such that M_1 is the set of the following axioms:

$$\begin{aligned}
&p(x) \leftarrow p_1(x)\&r_1(x),\\
&p_1(xy) \leftarrow p_2(x)\&p_3(y),\\
&p_2(axb) \leftarrow p_2(x),\\
&p_2(ab) \leftarrow,\\
&p_3(cx) \leftarrow p_3(x),\\
&p_3(c) \leftarrow,\\
&r_1(xy) \leftarrow r_2(x)\&r_3(y),\\
&r_2(ax) \leftarrow r_2(x),\\
&r_2(a) \leftarrow,\\
&r_3(bxc) \leftarrow r_3(x),\\
&r_3(bc) \leftarrow .
\end{aligned}$$

   Then $L(E_1) = \{a^n b^n c^n \mid n \geq 1\}$, which cannot be generated by any context-free grammar.

2. Suppose $\Sigma_2 = \{a, b, c\}$ and $D_2 = \{p, r\}$. Let M_2 be the set of axioms:

$$\begin{aligned}
&p(xyx) \leftarrow r(x)\&r(y),\\
&r(ax) \leftarrow r(x),\\
&r(bx) \leftarrow r(x),\\
&r(cx) \leftarrow r(x),\\
&r(a) \leftarrow,\\
&r(b) \leftarrow,\\
&r(c) \leftarrow .
\end{aligned}$$

Let $E_2 = (D_2, \Sigma_2, M_2, p)$. Then $L(E_2) = \{xyx \mid x, y \in \Sigma^+\}$, which is a *pattern language* (the language generated by the pattern xyx) in the sense of Angluin [Ang80a].

3. Suppose $\Sigma_3 = \{a\}$ and $D_3 = \{p\}$. Let M_3 be the set of axioms:

$$p(xx) \leftarrow p(x),$$
$$p(a) \leftarrow .$$

Let $E_3 = (D_3, \Sigma_3, M_3, p)$. Then $L(E_3) = \{a^{2^n} \mid n \geq 0\}$.

4.3.2 Relations with Other Formal Languages

We first show that the class of languages defined by ESFSs contains some important classes of formal languages.

Theorem 4.2 *If L is a context-free language, then there exists a context-free form of EFS E such that $L(E) = L$. Conversely, if E is a context-free form of EFS, then $L(E)$ is a context-free language.*

Proof. Let $G = (N, \Sigma, P, S)$ be any context-free grammar. We define the corresponding context-free form $E(G) = (D, \Sigma, M, S)$ of EFS as follows:

$$\begin{aligned} D &= N, \\ M &= \{A(v_0 x_1 v_1 \cdots x_n v_n) \leftarrow B_1(x_1) \& \cdots \& B_n(x_n) \mid \\ &\qquad A \rightarrow v_0 B_1 v_1 \cdots B_n v_n \in P \text{ with } B_1, \ldots, B_n \in N \text{ and } v_0, \ldots, v_n \in \Sigma^*\}. \end{aligned}$$

One can easily verify by induction that for any $A \in N$ and $w \in \Sigma^*$, $A(w)$ is provable from $E(G)$ if and only if $A \stackrel{*}{\Rightarrow} w$ in G. Then clearly $L(E(G)) = L(G)$.

Conversely, let $E = (D, \Sigma, M, p)$ be any context-free form of EFS. We define the corresponding context-free grammar $G(E) = (N, \Sigma, P, p)$ as follows:

$$\begin{aligned} N &= D, \\ P &= \{r \rightarrow t(r_1, \ldots, r_n) \mid r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1) \& \cdots \& r_n(x_n) \in M\}. \end{aligned}$$

We omit the simple inductive proof of $L(G(E)) = L(E)$. □

Thus the class of languages defined by ESFSs contains the class of context-free languages.

A *pattern* introduced by Angluin [Ang80a] is a non-empty finite string of terminal and variable symbols. The *language* of a pattern is all strings obtained by substituting non-empty terminal strings for the variables of the pattern.

Theorem 4.3 *Any pattern language is defined by an ESFS.*

Proof. Let L be the language of any pattern over Σ. Any pattern is a term in the terminology of this paper. Let t be a pattern (term) that generates L. We construct the corresponding ESFS $E = (D, \Sigma, M, p)$ as follows:

$$\begin{aligned} D &= \{p, r\}, \\ M &= \{p(t) \leftarrow r(x_1)\& \cdots \& r(x_n) \mid V(t) = \{x_1, \ldots, x_n\}\} \\ &\cup \{r(ax) \leftarrow r(x) \mid a \in \Sigma\} \\ &\cup \{r(a) \leftarrow \mid a \in \Sigma\}. \end{aligned}$$

Then $L(E) = L$. □

Next, we show that languages defined by ESFSs are context-sensitive.

Assume that

$$De : S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \cdots \Rightarrow \alpha_m = w$$

is a derivation in a grammar G. Then the *workspace of w by the derivation De* is defined by

$$WS_G(w, De) = \max\{|\alpha_i| \mid 1 \leq i \leq m\}.$$

The *workspace* of $w \in L(G)$ is defined by

$$WS_G(w) = \min\{WS_G(w, De) \mid De \text{ is a derivation of } w\}.$$

Lemma 4.4 (Salomaa [Sal73]) *If G is a phrase-structure grammar and there is a natural number n such that*

$$WS_G(w) \leq n|w|$$

for all non-empty strings $w \in L(G)$, then $L(G)$ is context-sensitive.

Theorem 4.5 *If $L = L(E)$ for an ESFS $E = (D, \Sigma, M, p)$, then L is a context-sensitive language.*

Proof. Without loss of generality, we assume that each axiom of E is either of the form $r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1)\& \cdots \& r_n(x_n)$ where the variables $x_1, \ldots, x_n$ are distinct or of the form $r(x) \leftarrow r_1(x)\& r_2(x)$. Let k be the maximum number of premises of any axiom

in M. We define the corresponding phrase-structure grammar $GS(E) = (N, \Sigma, P, S)$ as follows:

$$
\begin{aligned}
N &= D \\
&\cup \{A_{a_n} \mid a \in \Sigma,\ 1 \leq n \leq k\} \\
&\cup \{A_a \mid a \in \Sigma\} \\
&\cup \{X_0, X_1, X_2, \ldots, X_k, X_\#, X_{\#\#}, Y_1, Y_2, \ldots, Y_k, Z_1, Z_2, Z_3\}, \\
S &= p, \\
P &= \{r \to X_0 r_1 X_1 \cdots r_n X_n t(Y_1, \ldots, Y_n) X_\# \mid \\
&\qquad r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1) \& \cdots \& r_n(x_n) \in M\} && (4.1) \\
&\cup \{aX_n \to X_n A_{a_n} \mid a \in \Sigma,\ 1 \leq n \leq k\} && (4.2) \\
&\cup \{A_{a_n} B \to B A_{a_n} \mid a \in \Sigma,\ 1 \leq n \leq k \text{ and} \\
&\qquad B \in \Sigma \cup \{X_{n+1}, \ldots, X_k\} \cup \{Y_1, \ldots, Y_{n-1}, Y_{n+1}, \ldots, Y_k\}\} && (4.3) \\
&\cup \{A_{a_n} Y_n \to Y_n a A_{a_n} \mid a \in \Sigma,\ 1 \leq n \leq k\} && (4.4) \\
&\cup \{A_{a_n} X_\# \to X_\# \mid a \in \Sigma,\ 1 \leq n \leq k\} && (4.5) \\
&\cup \{X_0 X_1 \cdots X_n \to X_{\#\#} \mid 0 \leq n \leq k\} && (4.6) \\
&\cup \{X_{\#\#} a \to a X_{\#\#} \mid a \in \Sigma\} && (4.7) \\
&\cup \{X_{\#\#} Y_n \to X_{\#\#} \mid 1 \leq n \leq k\} && (4.8) \\
&\cup \{X_{\#\#} X_\# \to \epsilon\} && (4.9) \\
&\cup \{r \to Z_1 r_1 Z_2 r_2 Z_3 \mid r(x) \leftarrow r_1(x) \& r_2(x) \in M\} && (4.10) \\
&\cup \{Z_2 a \to A_a Z_2 \mid a \in \Sigma\} && (4.11) \\
&\cup \{b A_a \to A_a b \mid a, b \in \Sigma\} && (4.12) \\
&\cup \{Z_1 A_a a \to a Z_1 \mid a \in \Sigma\} && (4.13) \\
&\cup \{Z_1 Z_2 Z_3 \to \epsilon\}, && (4.14)
\end{aligned}
$$

where for $a \in \Sigma$ and $1 \leq n \leq k$, the elements A_{a_n}, A_a, X_n and Y_n are new nonterminals which are mutually distinct, and X_0, $X_\#$, $X_{\#\#}$, Z_1, Z_2, and Z_3 are additional nonterminals.

We prove that $L(GS(E)) = L(E)$ by induction on the step of the derivation of $GS(E)$.

For $r \to X_0 r_1 X_1 \cdots r_n X_n t(Y_1, \ldots, Y_n) X_\#$, suppose that a string $w_i \in \Sigma^*$ is derived from r_i in the grammar $GS(E)$ when $r_i(w_i)$ is provable from E $(1 \leq i \leq n)$. Using (4.1), all strings of the form

$$X_0 w_1 X_1 \cdots w_n X_n t(Y_1, \ldots, Y_n) X_\# \tag{4.15}$$

can be derived from r. By (4.2), (4.3), (4.4), and (4.5), the string

$$X_0X_1\cdots X_nt(Y_1w_1,\ldots,Y_nw_n)X_{\#} \tag{4.16}$$

can be derived from (4.15). By (4.6), (4.7), (4.8), and (4.9), the terminal string $t(w_1,\ldots,w_n)$ can be derived from (4.16).

For $r \rightarrow Z_1r_1Z_2r_2Z_3$, suppose that two strings $w_1, w_2 \in \Sigma^*$ are derived from r_1 and r_2 in the grammar $GS(E)$, respectively, when $r_1(w_1)$ and $r_2(w_2)$ are provable from E. Using (4.10), all strings of the form

$$Z_1w_1Z_2w_2Z_3 \tag{4.17}$$

can be derived from r. By (4.11), (4.12), and (4.13), the string

$$w_1Z_1Z_2Z_3 \tag{4.18}$$

can be derived from (4.17) if and only if $w_1 = w_2$. By (4.14), the terminal string w_1 can be derived from (4.18).

Because these are the only cases where a derivation leads from r to a terminal string, we conclude that $L(GS(E)) = L(E)$.

Further $WS_G(w) \leq |w|+(2k+3)|w|+(|w|+3) \leq (2k+8)|w|$. Hence by the workspace lemma 4.4, $L(GS(E))$ is context-sensitive. □

It is an open problem whether or not there are ϵ-free context-sensitive languages that cannot be defined by any ESFS.

Between the three subclasses of EFSs, we know that $C_{CF}\subsetneq C_{SFS}\subsetneq C_{ESFS}$ holds where C_{CF}, C_{SFS}, C_{SFS} denote the class of languages defined by context-free forms of EFSs, SFSs, ESFSs, respectively. The inclusion $\subseteq$ is by the definitions and the properness of the inclusion is established by Arikawa [Ari70] and the fact that the language $\{a^nb^nc^n \mid n \geq 1\}$ cannot be defined by any SFS.

4.3.3 Closure Properties

We show some closure and nonclosure properties for the languages defined by ESFSs.

Theorem 4.6 *The class of languages defined by ESFSs is closed under the operations of union, intersection, concatenation, Kleene closure, and reversal.*

Proof. Let $E_1 = (D_1, \Sigma, M_1, p_1)$ and $E_2 = (D_2, \Sigma, M_2, p_2)$ be ESFSs. We may assume that D_1 and D_2 are disjoint. Let $L_1 = L(E_1)$ and $L_2 = L(E_2)$. Assume also that predicates p_3, p_4, p_5, and p_6 are neither in D_1 nor D_2.

For $L_1 \cup L_2$, we construct the ESFS $E_3 = (D_1 \cup D_2 \cup \{p_3\}, \Sigma, M_3, p_3)$, where M_3 is $M_1 \cup M_2$ plus the axioms

$$p_3(x) \leftarrow p_1(x),$$
$$p_3(x) \leftarrow p_2(x).$$

Then $L(E_3) = L_1 \cup L_2$.

For intersection, we construct the ESFS $E_4 = (D_1 \cup D_2 \cup \{p_4\}, \Sigma, M_4, p_4)$, where M_4 is $M_1 \cup M_2$ plus the axiom

$$p_4(x) \leftarrow p_1(x)\&p_2(x).$$

Then $L(E_4) = L_1 \cap L_2$.

For concatenation, we construct the ESFS $E_5 = (D_1 \cup D_2 \cup \{p_5\}, \Sigma, M_5, p_5)$, where M_5 is $M_1 \cup M_2$ plus the axiom

$$p_5(xy) \leftarrow p_1(x)\&p_2(y).$$

Then $L(E_5) = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$.

For Kleene closure, we construct the ESFS $E_6 = (D_1 \cup \{p_6\}, \Sigma, M_6, p_6)$, where M_6 is M_1 plus the axioms

$$p_6(xy) \leftarrow p_1(x)\&p_6(y),$$
$$p_6(\epsilon).$$

Then $L(E_6) = L_1^*$.

For a term t, let t^R denote t written backward. For reversal, we construct the ESFS $E_7 = (D_1, \Sigma, M_7, p_1)$, where

$$r(t(x_1, \ldots, x_n)^R) \leftarrow r_1(x_1)\& \cdots \&r_n(x_n) \in M_7$$
$$\text{if and only if} \quad r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1)\& \cdots \&r_n(x_n) \in M_1.$$

Then $L(E_7) = \{w^R \mid w \in L_1\}$. □

In order to obtain a nonclosure property, we quote an important result of a representation theorem in [Har78].

Theorem 4.7 (Harrison [Har78]) *For each recursively enumerable set $L \subseteq \Sigma^*$, there exist deterministic context-free languages L_1 and L_2 and a homomorphism φ such that $L = \varphi(L_1 \cap L_2)$.*

Using this result, we can show the following:

Theorem 4.8 *The class of languages defined by ESFSs is not closed under homomorphism.*

Fischer [Fis68] generalized context-free grammars by introducing *macro grammars* based on programming macros, and studied two new classes of grammars, called *IO and OI grammars*, which differ only in the order in which nested terms may be expanded: IO is expansion from the inside-out and OI from the outside-in. Both the class of IO languages and that of OI languages, which are generated by IO and OI grammars respectively, are properly contained in the class of context-sensitive languages and closed under arbitrary homomorphism. Further, the class of OI languages is identical to the class of *indexed languages* studied by Aho [Aho68]. Thus we can at least conclude from Lemma 4.6 and Theorem 4.7 that the class of languages defined by ESFSs is contained in neither the class of IO languages nor the class of OI languages (i.e., indexed languages).

We can also regard formulas in elementary formal systems as Horn formulas in logic programming [Llo84] over strings in which the only function used is the concatenation. Hence it is interesting to compare ESFSs with *Definite Clause Grammars* [PW80].

4.4 The Learning Algorithm for ESFSs

Now we present a polynomial-time algorithm for learning ESFSs using queries. The learning algorithm may be viewed as a natural extension of Angluin's algorithm [Ang87a]. The lemmas and theorems that follow are analogous to Angluin's results.

Let k be any non-negative integer. An ESFS E is *k-bounded* if and only if every axiom of E has at most k occurrences of variables in the term of the conclusion and at most k premises. In this section, we show that there is a polynomial-time algorithm for learning any k-bounded ESFS by making queries.

Suppose $E_U = (D, \Sigma, M_U, p)$ is the unknown k-bounded ESFS to be learned (up to equivalence) by the learning algorithm. We assume that k, D, Σ, and p are known to the learning algorithm, but that M_U, the set of axioms, is unknown. This assumption that the predicate alphabet D is known to the learning algorithm is the same as Shapiro's assumption that theoretical predicates can be incorporated into the model inference problem [Sha81].

4.4.1 Types of Queries

A *membership query* (*m-query*, for short) proposes a terminal string $w \in \Sigma^*$ and asks whether it is in $L(E_U)$. The reply is either *yes* or *no*.

A *query for predicate provability* (*pp-query*, for short) proposes a ground atomic formula $r(w)$ of E_U and asks whether $r(w)$ is provable from E_U. The reply is either *yes* or

no. A membership query with w can be accomplished by a query for predicate provability with $p(w)$. Queries for predicate provability are analogous to nonterminal membership queries [Ang87a].

An *equivalence query* (*e-query*, for short) proposes an ESFS E and asks whether $L(E_U) = L(E)$. The answer is either *yes* or *no*. If it is *no*, then it provides a *counter-example*, that is, a terminal string w in the symmetric difference of $L(E_U)$ and $L(E)$. If $w \in L(E_U) - L(E)$, w is called a *positive* counter-example, and if $w \in L(E) - L(E_U)$, it is called a *negative* counter-example.

4.4.2 Proof-DAGs

We introduce a notion for ESFSs, called *proof-DAG*, which is similar to a parse-DAG [Ang87a]. Let the ESFS $E = (D, \Sigma, M, p)$ be fixed.

A proof-DAG for E is a finite directed acyclic graph that has a number of special properties. At each node there is a fixed linear ordering, called *left-to-right*, of the edges directed out of that node. There is exactly one node with in-degree zero, called the *root*. Each node with out-degree zero is called a *leaf* and the other nodes are called *internal nodes*. Every node has a label consisting of a ground atomic formula of E. For each node d, a node d' such that there is an edge from d to d' is called a *child* of d. Finally, for each internal node d, if its label is R and the labels of its children in left-to-right order are $R_1, \ldots, R_n$, then there is an axiom F in M and some substitution θ such that $F\theta = R \leftarrow R_1 \& \cdots \& R_n$, and for each leaf node d, if its label is R, then there is an axiom of the form $R \leftarrow$ in M.

The *size* of the proof-DAG T, denoted $size(T)$, is defined as the sum of the number of nodes in T, the number of edges in T, and the sum of the lengths of the labels on all the nodes of T. The *depth* of a proof-DAG T is the maximum number of nodes in any directed path in T.

For any node of a proof-DAG T, the *sub-DAG rooted at* d is the induced subgraph of T on all the nodes reachable from d by a directed path in T. Note that this is also a proof-DAG for the ESFS E.

4.4.3 The Learning Algorithm *EFSQ*

A formula $r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1) \& \cdots \& r_n(x_n)$ is said to be *incorrect for* an ESFS E if and only if there are n terminal strings $v_1, \ldots, v_n$ in Σ^* such that for $1 \leq i \leq n$, $r_i(v_i)$ is provable from E, but $r(t(v_1, \ldots, v_n))$ is not provable from E. A formula is *correct for*

ALGORITHM *EFSQ*

Input: Oracles that answer pp-queries and e-queries about M_U.

Output: An ESFS E.

Procedure:

%% Initialization

1 The set M of axioms is initialized to the empty set;

2 The algorithm then iterates the following loop;

%% Main loop

3 An e-query is made, proposing $E = (D, \Sigma, M, p)$;

4 If the reply is *yes*, the algorithm outputs E and halts;

5 Otherwise, a counter-example w is returned, and there are two cases;

%% Case (a) (a negative counter-example w is returned)

6 If w is in $L(E)$, then a proof-DAG for E is found with root label $p(w)$;

7 The proof-DAG is then diagnosed to find an axiom that is incorrect for E_U;

8 This axiom is removed from M;

%% Case (b) (a positive counter-example w is returned)

9 If w is not in $L(E)$,
then the set $C(w)$ of all candidate axioms is computed from w;

10 All of them are added to M.

Figure 4.1: The learning algorithm *EFSQ* for ESFSs

E if and only if it is not incorrect for E. Clearly, every axiom of E is correct for E.

The learning algorithm *EFSQ* for ESFSs is given in Figure 4.1.

The algorithm relies on three sub-procedures: *proof*, *diagnosis*, and the computation of *candidate axioms*. These are described in the following three subsections.

4.4.4 Proof Procedure

The proof sub-procedure takes a terminal string w and the ESFS $E = (D, \Sigma, M, p)$, and determines whether $p(w)$ is provable from E. If so, a proof-DAG for E with root node labelled $p(w)$ is also returned.

Let

$$H = \{v \mid v \text{ is a substring of } w\},$$

where a string v is a *substring* of w if and only if there are strings u_1 and u_2 such that $w = u_1 v u_2$. There are at most $(|w|+1)^2$ elements of H, and H is easily computed in time polynomial in $|w|$. Let

$$I = \{r(v) \mid v \in H,\ r \in D,\ \text{and } r(v) \text{ is provable from } E\}.$$

There are at most $|D|(|w|+1)^2$ elements of I. Clearly, $p(w)$ is provable from E if and only if $p(w)$ is in I. The procedure computes I as follows.

Initially let $J = \emptyset$, and for each axiom of the form $r(v) \leftarrow$ in M such that v is a substring of w, the procedure puts $r(v)$ in J. Then the following process is iterated until the first iteration in which no new elements are added to J.

For each axiom of the form

$$r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1) \& \cdots \& r_n(x_n)$$

and for every n-tuple

$$\langle r_1(v_1), \ldots, r_n(v_n) \rangle$$

of elements of J with $v_i = v_j$ in case $x_i = x_j$ for $1 \leq i, j \leq n$, if $t(v_1, \ldots, v_n)$ is a substring of w, then put $r(t(v_1, \ldots, v_n))$ in J if it is not already there.

The following modification of the procedure records information for a proof-DAG. The nodes of the proof-DAG will be elements of I. Suppose $r(v)$ is added to J because the procedure finds an axiom of the form

$$r(t(x_1, \ldots, x_n)) \leftarrow r_1(x_1) \& \cdots \& r_n(x_n)$$

and the n-tuple of elements

$$\langle r_1(v_1), \ldots, r_n(v_n) \rangle$$

of J such that $v = t(v_1, \ldots, v_n)$ is a substring of w. Then for the element $r(v)$ the procedure adds an ordered list of n edges from $r(v)$ to the elements $r_1(v_1), \ldots, r_n(v_n)$ of J.

At the end of the procedure, if $p(w)$ is in J, a proof-DAG is constructed as follows. Discard from J all elements not reachable from $p(w)$ by a directed path of edges. The label of each node $r(y)$ is then just $r(y)$. The root node is $p(w)$. This completes the description of the proof procedure.

Since I is finite and the set M of axioms is also finite, this procedure terminates. One can easily verify by induction that this procedure computes I.

We assume that E is k-bounded.

Lemma 4.9 *There are a non-decreasing polynomial $n_1(x, y)$ such that the time used by the proof procedure on inputs E, p, and w is bounded by $n_1(size(E), |w|)$, and a non-decreasing polynomial $n_2(x, y)$ such that the proof-DAG returned by the proof procedure on inputs E, w, and p is of size bounded by $n_2(|D|, |w|)$.*

Proof. Every iteration of the loop except the last adds at least one element to J, so there are at most $|D|(|w|+1)^2$ iterations of the loop. Each iteration of the loop considers at most $|M|$ axioms, and for each axiom with m premises, considers at most all m-tuples of elements of I. Since E is k-bounded, $m \leq k$. Hence the basic operation of applying a substitution to the term in the argument of the conclusion of an axiom and testing whether it is a substring of w is done no more than

$$|M|(|D|(|w|+1)^2)^{k+1}$$

times in all. The time for proving is bounded by a polynomial in the length of w and the size of E (but exponential in k).

Clearly the proof-DAG has at most $|I|$ nodes, which is bounded by $|D|(|w|+1)^2$. Each node has a label of length bounded by $|w|+4$, and each node has at most k edges directed out of it. Thus the total size of the resulting proof-DAG is bounded by

$$|D|(|w|+1)^2(|w|+k+5).$$

This proves Lemma 4.9. □

4.4.5 Diagnosis Procedure

The diagnosis sub-procedure finds an axiom that is incorrect for E_U of the current ESFS $E = (D, \Sigma, M, p)$ rejected by an e-query with a negative counter-example. We can use Shapiro's diagnosis algorithm for Prolog programs [Sha82] for the purpose, because ESFSs can be regarded as logic programs over strings. Thus the diagnosis procedure is essentially a special case of his algorithm for diagnosing an incorrect output. The input to the diagnosis procedure is a correct proof-DAG T for E such that $p(v)$ is the label of the root and $p(v)$ is not provable from E_U. The output of the diagnosis procedure is an axiom of E that is incorrect for E_U.

If the root of T has no child (that is, if T consists of one node), then the diagnosis procedure returns the formula $p(v) \leftarrow$. Otherwise the diagnosis procedure considers in turn each child of the root of T. If the child is labelled with $r(u)$, then the diagnosis procedure makes a pp-query with $r(u)$. If the reply is *no*, then it calls itself recursively with the sub-DAG rooted at the child, and returns the resulting axiom. If the reply is *yes*, then it goes on to the next child of the root of T.

If all the queries with $p_1(v_1), \ldots, p_n(v_n)$ that are labels of the children of the root of T in left-to-right order are answered *yes*, then the diagnosis procedure finds an axiom F of E such that $F\theta = p(v) \leftarrow p_1(v_1)\& \cdots \& p_n(v_n)$ for some substitution θ, and returns F.

Lemma 4.10 *When the diagnosis procedure is given as input a proof-DAG T for E that has the root labelled $p(v)$ such that $p(v)$ is not provable from E_U, it returns an axiom of E that is incorrect for E_U. Further, there is a non-decreasing polynomial $n_3(x, y)$ such that the time required by the diagnosis procedure on input T is bounded by $n_3(size(T), |M|)$.*

Proof. If the input conditions are met on the initial call, then each recursive call preserves the input conditions. Since each recursive call is with a proper sub-DAG of its input DAG, the procedure must eventually terminate, and since the original DAG is a correct proof-DAG for E (as is every sub-DAG), the diagnosis procedure always finds an axiom of E and returns it.

If the formula $p(v) \leftarrow$ is returned, then it is clearly an axiom of E and incorrect for E_U. If the axiom of the form $p(t(x_1, \ldots, x_n)) \leftarrow p_1(x_1)\& \cdots \& p_n(x_n)$ is returned, then the queries have witnessed that there are n terminal strings $v_1, \ldots, v_n$ such that for $1 \leq i \leq n$, $p_i(v_i)$ is provable from E, but $p(t(v_1, \ldots, v_n)) = p(v)$ is not provable from E, that is, the axiom $p(t(x_1, \ldots, x_n)) \leftarrow p_1(x_1)\& \cdots \& p_n(x_n)$ is incorrect for E_U.

Next, the number of queries made by the diagnosis procedure is at most $size(T)$. The diagnosis procedure considers at most $|M|$ axioms to find the axiom F. It is clear that

a straightforward implementation of the diagnosis procedure runs in time polynomial in $size(T)$ and $|M|$. This proves Lemma 4.10. □

4.4.6 Candidate Axioms

The input to the candidate axiom procedure is a terminal string w such that w is in $L(E_U)$ but not in $L(E)$, where $E = (D, \Sigma, M, p)$ is the current ESFS rejected by an e-query with a positive counter-example w, and the output is a set $C(w)$ of axioms added to M such that at least one element of $C(w)$ is in M_U but not in M.

The procedure considers in turn every substring s of w. For every $m \leq k$ and every factorization of s into $2m+1$ substrings,

$$s = u_0 v_1 u_1 v_2 u_2 \cdots v_m u_m,$$

and for every $1 \leq n \leq k$ and every $n+1$-tuple of predicates $r, r_1, \ldots, r_n$ from D, the formula

$$r(u_0 x_1 u_1 x_2 u_2 \cdots x_m u_m) \leftarrow r_1(z_1) \& \cdots \& r_n(z_n)$$

is added to $C(w)$, where $x_1, \ldots, x_m$ are variables that are not necessarily distinct, $\{x_1, \ldots, x_m\} = \{z_1, \ldots, z_n\}$ and $z_1, \ldots, z_n$ are not necessarily distinct, and the formula

$$r(s) \leftarrow$$

is added to $C(w)$.

Example 4.3 Let $k = 2$, $D = \{p, r\}$, and the positive counter-example be aba. All possible candidates of terms for conclusions of the candidate axioms are

$$\begin{aligned}
Term_1 &= \{\epsilon, a, b, ab, ba, aba\},\\
Term_2 &= \{x, xa, ax, xb, bx, xab, axb, abx, xba, bxa, bax, axa, xaba, axba, abxa, abax\},\\
Term_3 &= \{xy, xx, xya, xxa, xay, xax, axy, axx, xyb, xxb, xby, xbx, bxy, bxx,\\
&\quad xyab, xxab, xayb, xaxb, xaby, xabx, axyb, axxb, axby, axbx, abxy, abxx,\\
&\quad xyba, xxba, xbya, xbxa, xbay, xbax, bxya, bxxa, bxay, bxax, baxy, baxx,\\
&\quad xaya, xaxa, axya, axxa, axay, axax,\\
&\quad xyaba, xxaba, xayba, xaxba, xabya, xabxa, xabay, xabax, axyba, axxba,\\
&\quad axbya, axbxa, axbay, axbax, abxya, abxxa, abxay, abxax, abaxy, abaxx\}.
\end{aligned}$$

Then

$$C(aba) = \{p(t) \leftarrow,\ r(t) \leftarrow \mid t \in Term_1\}$$

$$\cup \ \{p(t(x)) \leftarrow p(x),\ r(t(x)) \leftarrow p(x) \mid t(x) \in Term_2\}$$
$$\cup \ \{p(t(x)) \leftarrow r(x),\ r(t(x)) \leftarrow r(x) \mid t(x) \in Term_2\}$$
$$\cup \ \{p(t(x)) \leftarrow p(x)\&r(x),\ r(t(x)) \leftarrow p(x)\&r(x) \mid t(x) \in Term_2\}$$
$$\cup \ \{p(t(x,y)) \leftarrow p(x)\&r(y),\ r(t(x,y)) \leftarrow p(x)\&r(y) \mid t(x,y) \in Term_3\}$$
$$\cup \ \{p(t(x,y)) \leftarrow p(y)\&r(x),\ r(t(x,y)) \leftarrow p(y)\&r(x) \mid t(x,y) \in Term_3\}$$
$$\cup \ \{p(t(x,y)) \leftarrow p(x)\&p(y),\ r(t(x,y)) \leftarrow p(x)\&p(y) \mid t(x,y) \in Term_3\}$$
$$\cup \ \{p(t(x,y)) \leftarrow r(x)\&r(y),\ r(t(x,y)) \leftarrow r(x)\&r(y) \mid t(x,y) \in Term_3\}.$$

Lemma 4.11 *Let the candidate axiom procedure have input $w \in L(E_U) - L(E)$. Then $C(w)$ contains some axiom in M_U but not in M. Further, there are non-decreasing polynomials $n_4(x,y)$ and $n_5(x,y)$ such that on input w the candidate axiom procedure runs in time bounded by $n_4(|D|,|w|)$ and produces an output set $C(w)$ with at most $n_5(|D|,|w|)$ elements. Moreover, every axiom in $C(w)$ has a term of length at most $|w|+k$ with at most k occurrences of variables in the conclusion and at most k premises.*

Proof. Since w is in $L(E_U)$, there is a proof-DAG T for E_U with root labelled $p(w)$. We show that every axiom used in T is in $C(w)$.

Consider any sub-DAG T' of T rooted at a node with label $r(s)$. Suppose the axiom used at the root of T' is

$$r(t(x_1,\ldots,x_m)) \leftarrow r_1(x_1)\&\cdots\&r_m(x_m).$$

There are m substrings $s_1,\ldots,s_m$ of s such that

$$s = t(s_1,\ldots,s_m).$$

Since s is a substring of w and $m \leq k$, the axiom

$$r(t(x_1,\ldots,x_m)) \leftarrow r_1(x_1)\&\cdots\&r_m(x_m).$$

will be generated from s using the above factorization and choices of predicates.

Thus every axiom used in T is in $C(w)$. If every axiom in $C(w) \cap M_U$ were in M, T would be a proof-DAG for the ESFS E witnessing $w \in L(E)$, a contradiction. Thus, $C(w)$ contains some axiom in $M_U - M$.

Next, for each $m \leq k$, there are no more than $(|w|+1)^{2m}$ factorizations of the string w into $2m+1$ substrings, and no more than m^m choices of m variables for the term of the conclusion and $k(m|D|)^k$ choices of atomic formulas for the premises. Thus, the total number of axioms placed in $C(w)$ is at most

$$1 + k(|w|+1)^{2k+2}k^k|D|k(k|D|)^k.$$

Computing these axioms takes time bounded by a polynomial in $|D|$ and $|w|$. The term in the conclusion of each axiom consists of a subsequence of the terminals in w and at most k variables, for a total length bounded by $|w| + k$, which completes the proof of Lemma 4.11. □

4.4.7 Correctness and Time Complexity

In the algorithm *EFSQ*, since M is initially empty and is only augmented by axioms output by the candidate axioms procedure, $E = (D, \Sigma, M, p)$ is k-bounded at all times by Lemma 4.11. Clearly if the algorithm *EFSQ* ever terminates, its output is an ESFS E equivalent to E_U. This shows the partial correctness of the algorithm *EFSQ*. Now we estimate the number of iterations of the main loop in *EFSQ* as follows.

Lemma 4.12 *There are at most $|M_U|$ iterations of case* (b) *of the main loop in EFSQ with positive counter-examples, and if max_{pos} is the maximum length of positive counter-examples, then there are at most $|M_U| n_5(|D|, max_{pos})$ iterations of case* (a) *of the main loop in EFSQ with negative counter-examples.*

Proof. By Lemma 4.11, each iteration of case (b) with a positive counter-example must add to M at least one axiom in $M_U - M$. This axiom is correct for E_U and therefore cannot be removed from M, because the only elements removed from M are incorrect for E_U, by Lemma 4.10. Hence there are at most $|M_U|$ iterations of case (b) with positive counter-examples.

Thus there are at most $|M_U|$ positive counter-examples, say

$$w_1, w_2, \ldots, w_m,$$

where $m \leq |M_U|$. Let max_{pos} be the maximum length of w_i for $i = 1, 2, \ldots, m$.

The total number of axioms added to M is bounded by

$$|C(w_1)| + |C(w_2)| + \cdots + |C(w_m)|,$$

which by Lemma 4.11 is bounded by

$$n_5(|D|, |w_1|) + n_5(|D|, |w_2|) + \cdots + n_5(|D|, |w_m|).$$

Since $n_5(x, y)$ is non-decreasing we have a bound of

$$|M_U| n_5(|D|, max_{pos})$$

on the total number of axioms ever added to M.

Each iteration of case (a) with a negative counter-example removes one axiom from M. Hence this can happen at most as many times as there are axioms ever added to M, which is bounded by

$$|M_U| n_5(|D|, max_{pos}).$$

This proves Lemma 4.12. □

Thus the algorithm *EFSQ* must terminate after at most

$$|M_U| + |M_U| n_5(|D|, max_{pos})$$

iterations of the main loop.

Let max_{neg} be the maximum length of negative counter-examples, and let max be the maximum of max_{pos} and max_{neg}. max is the maximum length of counter-examples encountered before termination.

We bound the time used by the algorithm *EFSQ* as follows.

Lemma 4.13 *The time required by the proof procedure at each iteration of the main loop of the algorithm EFSQ is bounded by a polynomial in the size of E_U and the length of the longest counter-example.*

Proof. By Lemma 4.9 the time required to prove $p(w)$ for each counter-example w is bounded by $n_1(size(E), |w|)$. Clearly $|w| \leq max$, but we need to establish a bound on $size(E)$.

We have

$$size(E) = |D| + |\Sigma| + |M| + L,$$

where L is the sum of the lengths of the terms in the conclusions of the axioms in M. By Lemma 4.12,

$$|M| \leq |M_U| n_5(|D|, max_{pos}).$$

Each axiom in M has a term of length at most $max_{pos} + k$ in the conclusion, by Lemma 4.11.

Hence,

$$size(E) \leq |D| + |\Sigma| + |M_U| n_5(|D|, max_{pos})(1 + max_{pos} + k).$$

Therefore the size of E can be bounded by a non-decreasing polynomial in the size of E_U and the length of the longest positive counter-example,

$$size(E) \leq n_6(size(E_U), max_{pos}).$$

Thus at each iteration the time to prove $p(w)$ for a counter-example w is bounded by

$$n_1(n_6(size(E_U), max_{pos}), max),$$

which proves Lemma 4.13. □

Lemma 4.14 *The time required by the diagnosis procedure at each iteration where it is called is bounded by a polynomial in the size of E_U and the length of the longest counter-example.*

Proof. By Lemmas 4.9, 4.10 and 4.12, the time required by the diagnosis procedure at each iteration where it is called is bounded by

$$n_3(n_2(|D|, max), |M_U|n_5(|D|, max_{pos})),$$

which is bounded by

$$n_3(n_2(size(E_U), max), size(E_U)n_5(size(E_U), max_{pos})).$$

□

Lemma 4.15 *The time required by the candidate axioms procedure at each iteration where it is called is bounded by a polynomial in the size of E_U and the length of the longest counter-example.*

Proof. By Lemma 4.11, the time required by the candidate axioms procedure with input w is bounded by $n_4(|D|, |w|) \leq n_4(size(E_U), max)$. □

We now come to the main theorem of this chapter.

Theorem 4.16 *There is an algorithm that learns an ESFS equivalent to any k-bounded ESFS E_U by making e-queries and pp-queries that runs in time polynomial in the size of E_U and the length of the longest counter-example.*

Proof. Putting the bounds from the above three lemmas together with the bound of

$$|M_U| + |M_U|n_5(|D|, max_{pos})$$

on the number of iterations of the main loop of the algorithm *EFSQ*, we conclude that the total time used by the algorithm *EFSQ* is bounded by a polynomial in the size of E_U and the length of the longest counter-example. □

Part II

Learning from Large Data in Noisy Environment

Chapter 5

Probably Approximately Correct Learning from Noisy Examples

Valiant [Val84] has introduced the distribution-independent model of concept learning from random examples. Angluin and Laird [AL88] have extended its model by introducing a noise process, called *classification noise process*, to study how to compensate for randomly introduced errors, or "noise", in classifying the example data. In this chapter, we give a brief outline of the Valiant's learnability model and the notion of polynomial learnability [BEHW89] for Boolean functions, and define the classification noise process and the notion of polynomial learnability in the presence of classification noise for Boolean functions. Then we develop a technique of building efficient robust learning algorithms, called *noise-tolerant Occam algorithm*, and show that using a noise-tolerant Occam algorithm for a class of Boolean functions, one can construct a polynomial-time algorithm for learning the class in the presence of classification noise.

5.1 Probably Approximately Correct Learning for Boolean Functions

We assume that there are n Boolean attributes (or variables) to be considered, and we denote the set of such variables as $V_n = \{x_1, x_2, \ldots, x_n\}$. An *assignment* $\vec{a}$ is a mapping from V_n to the set $\{0,1\}$. Let X_n denote the set of all such assignments mapping from V_n to $\{0,1\}$. We may also think X_n denotes the set $\{0,1\}^n$ of all binary strings of length n. Then a *Boolean function* is defined to be a mapping from X_n to $\{0,1\}$.

Boolean formulae are often used as useful representations for Boolean functions. The simplest Boolean formula is just a single variable. Each variable x_i $(1 \leq i \leq n)$ is associated with two *literals*: x_i itself and its negation $\bar{x}_i$. A *term* is a conjunction of

literals and a *clause* is a disjunction of literals. The *size* of a term or clause is the number of its literals. Let **true** be the unique term of size 0, which always returns the value 1, and **false** be the unique clause of size 0, which always returns the value 0. Let C_k^n denote the set of all terms of size at most k over V_n and D_k^n denote the set of all clauses of size at most k over V_n. Thus

$$|C_k^n| = |D_k^n| = O(n^k) \quad (< (2n+1)^k).$$

For any fixed k, C_k^n and D_k^n have sizes polynomial in n.

A Boolean formula is in *conjunctive normal form* if it is the conjunction of clauses. We define k-CNF to be the class of Boolean formulae in conjunctive normal form with at most k literals per clause. Similarly, a Boolean formula is in *disjunctive normal form* if it is the disjunction of terms. We define k-DNF to be the class of Boolean formulae in disjunctive normal form with at most k literals per term.

A Boolean formula can be interpreted as a mapping from assignments X_n into $\{0,1\}$. Thus each Boolean formula defines a corresponding Boolean function from X_n to $\{0,1\}$ in a natural manner. We do not distinguish between Boolean formulae and the Boolean functions they represent.

Now we describe the Valiant's learnability model for Boolean functions. First fix a class F_n of Boolean functions over V_n and a target Boolean function f_U in F_n to be learned.

An *example* of f_U is a pair $\langle \vec{a}, l \rangle$ where $\vec{a}$ is an assignment in X_n and $l = f_U(\vec{a})$. Thus an example can be viewed as an assignment with the value of the target function f_U at the assignment. l is called the *label* of the example. An example $\langle \vec{a}, l \rangle$ is called a *positive example* of f_U if $l = 1$ and called a *negative example* of f_U if $l = 0$. A *sample* is a finite sequence of positive and negative examples of the target Boolean function f_U. The *size* of a sample S is the number of examples in it. A Boolean function g is said to *agree with* an example $\langle \vec{a}, l \rangle$ if $g(\vec{a}) = l$. A Boolean function is *consistent* with the given sample if it agrees with all examples in the sample.

We assume that there is an unknown and arbitrary probability distribution D on X_n. The probability of assignment $\vec{a} \in X_n$ with respect to D is denoted $\Pr_D(\vec{a})$. Random samples are assumed to be drawn independently from the domain X_n according to this probability distribution D on X_n. There is a *sampling oracle* $EX()$ for the target Boolean function f_U, which has no input. Whenever $EX()$ is called, it draws an assignment $\vec{a} \in X_n$ according to the distribution D, and returns $\langle \vec{a}, f_U(\vec{a}) \rangle$. We define a *learning algorithm* for the class F_n of Boolean function as an algorithm that has access to $EX()$ and produces as output a Boolean function in F_n.

A learning algorithm makes a number of calls to $EX()$ and then conjectures some Boolean function $g \in F_n$. The success of learning is measured by two parameters, the accuracy parameter ϵ and the confidence parameter δ, which are given as inputs to the learning algorithm. We define a notion of the difference between two Boolean functions f and g with respect to the probability distribution D as

$$d(f,g) = \sum_{f(\vec{a}) \neq g(\vec{a})} \Pr_D(\vec{a}).$$

The *error* of a Boolean function g with respect to the target Boolean function f_U is $d(g, f_U)$. A successful learning algorithm is one that with high probability finds a Boolean function whose error is small. A Boolean function g is called an *ϵ-approximation* of f_U if $d(g, f_U) \leq \epsilon$ and called *ϵ-bad* otherwise.

The notion of *polynomial learnability* in the Valiant's learnability model is formally defined as follows.

> A class F_n of Boolean functions over V_n is *polynomially (probably approximately correctly (PAC for short)) learnable* if there exists a learning algorithm A for F_n such that for any ϵ and δ, for any target Boolean function $f_U \in F_n$, and for any distribution D on X_n, when A is given as input parameters n, ϵ, and δ and run with the sampling oracle $EX()$ for f_U, the algorithm outputs a Boolean function $g \in F_n$ such that $d(g, f_U) \leq \epsilon$ with probability at least $1 - \delta$, and runs in time polynomial in n, $1/\epsilon$, and $1/\delta$.

The difficulty of learning a Boolean function that has been selected from F_n will depend on the size $|F_n|$ of F_n. We say a class F_n of Boolean functions is *polynomial-sized* if $\ln(|F_n|) = O(n^t)$ for some constant t, that is, $\ln(|F_n|)$ is a polynomial in n. $\ln(|F_n|)$ may be viewed as the number of bits needed to write down an arbitrary element of F_n, using an "optimal encoding". Thus any representation for a polynomial-sized class of Boolean functions has its size at most polynomial in n.

Several interesting classes of Boolean functions have been proved to be or not to be polynomially learnable in this Valiant's learnability model [KLPV87, PV88].

5.2 Classification Noise Process

Many works making progress in the Valiant's learnability model depend strongly on the assumption of perfect, noise-less examples. This assumption is generally unrealistic and

in many situations of the real world, we are not always so fortune, our observations will often be afflicted by noise and hence there is always some chance that a noisy example is given to the learning algorithm. Few works have suggested any way to make their learning algorithms noise tolerant and two formal models of noise have been studied so far in the Valiant's learnability model for concept learning. One is the *malicious error model* initiated by Valiant [Val85] and investigated by Kearns and Li [KL88]:

> Independently for each example, the example is replaced, with some small probability, by an arbitrary example classified perhaps incorrectly.

The goal of this model is to capture the worst possible case of noise process by the adversary. This model is also called *adversarial noise process* in [AL88]. The other is the *classification noise process* introduced by Angluin and Laird [AL88]:

> Independently for each example, the label of the example is reversed with some small probability.

The goal of this model is to study the question of how to compensate for randomly introduced errors, or "noise, in classifying the example data. In this paper, we consider the classification noise process to study the effect on the polynomial learnability of Boolean functions.

This classification noise process introduced for concept learning can be interpreted to be applicable to learning Boolean functions as follows:

> The sampling oracle is able to draw assignments $\vec{a}$ from X_n according to the relevant distribution D without error, but that the process of reporting the value of the target Boolean function f_U at the assignment $\vec{a}$, that is $f_U(\vec{a})$, is subject to independent random mistakes with some unknown probability η; independently for each example $\langle \vec{a}, l \rangle$, $\langle \vec{a}, 0 \rangle$ is returned when $f_U(\vec{a}) = 1$ and $\langle \vec{a}, 1 \rangle$ is returned when $f_U(\vec{a}) = 0$ with probability η.

It is assumed that the rate of noise η is less than $1/2$. To indicate that the sampling oracle is subject to errors of this type, we will denote it by $EX_\eta()$.

In [AL88], the following argument is discussed: in the presence of classification noise, we should assume that there is some information about the noise rate η available to the learning algorithm, namely an upper bound η_b such that $\eta \leq \eta_b < 1/2$, and just as the running time for polynomial-time learning is permitted in the absence of noise to be

polynomial in $1/\epsilon$ and $1/\delta$, we should permit the polynomial to have $1/(1-2\eta_b)$ as one of its arguments.

Now we give the precise definition of *polynomial learnability in the presence of classification noise*.

> A class F_n of Boolean functions over V_n is *polynomially learnable in the presence of classification noise* if there exists a learning algorithm for F_n such that for any ϵ, δ, and η ($< 1/2$), for any target Boolean function $f_U \in F_n$, and for any distribution D on X_n, when A is given as input parameters n, ϵ, δ, and η_b ($\eta \leq \eta_b < 1/2$), and run with the sampling oracle $EX_\eta()$ for f_U, the algorithm outputs a Boolean function $g \in F_n$ such that $d(g, f_U) \leq \epsilon$ with probability at least $1-\delta$, and runs in time polynomial in n, $1/\epsilon$, $1/\delta$, and $1/(1-2\eta_b)$.

5.3 Previous Research Results

In the absence of noise, when given a sample of a target Boolean function f_U, the fundamental strategy that a learning algorithm takes is producing a Boolean function consistent with the sample. When the sample contains noise, this fundamental strategy may fail because there is no guarantee that such consistent Boolean functions will be found.

Angluin and Laird [AL88] have proposed the simple strategy of finding a Boolean function that minimizes the number of disagreements with the given sample and shown that in the presence of classification noise, a learning algorithm for a class F_n of Boolean functions that outputs a Boolean function minimizing the number of disagreements PAC-learns F_n.

Let $S = \langle \vec{a}_1, l_1 \rangle, \langle \vec{a}_2, l_2 \rangle, \ldots, \langle \vec{a}_m, l_m \rangle$ be a sample drawn from an $EX_\eta()$ oracle. For a Boolean function f, let $F(f, S)$ denote the number of indices j for which f disagrees with $\langle \vec{a}_j, l_j \rangle$.

Theorem 5.1 (Angluin & Laird [AL88]) *If we draw a sample S of*

$$m \geq \frac{2}{\epsilon^2(1-2\eta_b)^2} \ln\left(\frac{2|F_n|}{\delta}\right)$$

examples from $EX_\eta()$ for the target Boolean function f_U and find any Boolean function $g \in F_n$ that minimizes $F(g, S)$, then with probability at least $1-\delta$, g is an ϵ-approximation of f_U.

Angluin and Laird [AL88] have shown that k-CNF is polynomially learnable in the presence of classification noise for any $\eta < 1/2$.

Kearns and Li [KL88] have given hardness results for learning with *malicious errors*. Let us define a class F_n of Boolean functions to be *distinct* if there are Boolean functions $f, g \in F_n$ and assignments $\vec{a}, \vec{b} \in X_n$ satisfying $f(\vec{a}) = f(\vec{b}) = 1$, $g(\vec{a}) = 0$, and $g(\vec{b}) = 1$.

Theorem 5.2 (Kearns & Li [KL88]) *Let F_n be a distinct class of Boolean functions and ϵ the accuracy parameter. Then the largest rate of malicious error that can be tolerated by any learning algorithm for F_n is less than $\epsilon/(1+\epsilon)$.*

Laird [Lai88] has discussed about estimating the noise rate, other types of noise process, etc., and shown several interesting results. Sloan [Slo88] have introduced two new models between the malicious error model and the classification noise process to show that they can be "pushed towards one another".

5.4 Noise-tolerant Occam algorithm

In this section, we develop a technique of building efficient robust learning algorithms, called *noise-tolerant Occam algorithm*, that is a generalization of Occam's Razor in [BEHW87] and show that using a noise-tolerant Occam algorithm for a class of Boolean functions, one can construct a polynomial-time algorithm for learning the class in the presence of classification noise, which we will find useful in the following chapters to show the polynomial learnability of decision lists and decision trees in the presence of classification noise.

Angluin and Laird [AL88] have proposed the strategy of finding a Boolean function that minimizes the number of disagreements with the given sample. In general, however, it is a hard problem to find a Boolean function that minimizes the number of disagreements with the sample. We weaken this criterion. The following noise-tolerant Occam algorithm takes the strategy of, rather than finding a Boolean function that minimizes the number of disagreements, finding a Boolean function consistent with a large fraction of the given sample with high probability.

> A *noise-tolerant Occam algorithm*, $OCCAM_{In}(S, \epsilon, \delta, \eta_b)$, for a class F_n of Boolean functions over V_n is an algorithm that when given as input a sample S of m examples drawn from $EX_\eta()$ for any target Boolean function f_U, and parameters ϵ, δ, η_b,

1. produces a Boolean function $g \in F_n$ such that

$$\frac{F(g,S)}{m} \leq \eta_b + \frac{\epsilon(1-2\eta_b)}{4},$$

with probability at least $1-\delta/2$, and

2. runs in time polynomial in n and m.

Now we present two theorems that give the reason why this can work. First we show the following important theorem for $OCCAM_{in}(S,\epsilon,\delta,\eta_b)$ that if $\eta \leq \eta_b \leq \eta + \epsilon(1-2\eta)/2$ and F_n is polynomial-sized, the existence of $OCCAM_{in}(S,\epsilon,\delta,\eta_b)$ for F_n implies the polynomial learnability of F_n in the presence of classification noise.

We quote the following inequality which we shall require in the proof to follow. Let p and q be numbers between 0 and 1, and let m be a positive integer. Let $GE(p,m,q)$ denote the probability of getting at least qm successes in m independent Bernoulli trials with probability p, and $LE(p,m,q)$ denote the probability of at most qm successes in m independent Bernoulli trials with probability p. The following lemma bounds these quantities.

Lemma 5.3 (Hoeffding's Inequality [AL88]) *If $0 \leq p \leq 1$, $0 \leq s \leq 1$, and m is any positive integer, then*

$$LE(p,m,p-s) \leq e^{-2s^2m}$$

and

$$GE(p,m,p+s) \leq e^{-2s^2m}.$$

Theorem 5.4 *Suppose that $\eta \leq \eta_b \leq \eta + \epsilon(1-2\eta)/2$. Suppose also that F_n is polynomial-sized. If there exists a noise-tolerant Occam algorithm for F_n, then F_n is polynomially learnable in the presence of classification noise. The sample size required is*

$$m \geq \frac{8}{\epsilon^2(1-2\eta_b)^2} \ln\left(\frac{2|F_n|}{\delta}\right).$$

Proof. First we consider the effect of classification noise on searching any Boolean function in F_n. We analyze the expected rate of disagreement between any Boolean function g and example sequences produced by the sampling oracle $EX_\eta()$ for the target Boolean function f_U. Let $d_g = d(g,f_U)$. The probability that an example produced by $EX_\eta()$ disagrees with g is

1. the probability that an example is drawn from $\{\vec{a} \in X_n \mid f_U(\vec{a}) \neq g(\vec{a})\}$ and reported correctly (which is just $d_g(1-\eta)$)

2. plus the probability that an example is drawn from $\{\vec{a} \in X_n \mid f_U(\vec{a}) = g(\vec{a})\}$ and reported incorrectly (which is just $(1 - d_g)\eta$).

Let p_g denote the probability that an example from $EX_\eta()$ disagrees with g. Then we have

$$\begin{aligned} p_g &= d_g(1-\eta) + (1-d_g)\eta \\ &= \eta + d_g(1-2\eta). \end{aligned}$$

For the target Boolean function f_U we have $p_{f_U} = \eta$, and for any ϵ-bad Boolean function g we have

$$p_g \geq \eta + \epsilon(1-2\eta).$$

Thus any ϵ-bad Boolean function has an expected rate of disagreement that is greater than that of the target Boolean function by at least $\epsilon(1-2\eta)$.

We now show that with probability at least $1-\delta$, $OCCAM_{ln}(S, \epsilon, \delta, \eta_b)$ outputs an ϵ-approximation of f_U. Let $s = \epsilon(1-2\eta_b)$. The probability that the target Boolean function f_U has more than $(\eta_b + s/4)m$ disagreements with a sample S of m examples drawn from $EX_\eta()$ is

$$\begin{aligned} GE(\eta, m, \eta_b + s/4) &\leq GE(\eta_b, m, \eta_b + s/4) \\ &\leq e^{-2(s/4)^2 m} \end{aligned}$$

by the Hoeffding's inequality lemma and the lower bound on m implies that this is less than $\delta/2$. Hence with probability at least $1-\delta/2$, $OCCAM_{ln}(S, \epsilon, \delta, \eta_b)$ can find a Boolean function $g \in F_n$ such that $F(g,S)/m \leq \eta_b + s/4$ to output. The probability that a Boolean function with error greater than ϵ has at most $(\eta_b + s/4)m$ disagreements is

$$\begin{aligned} LE(\eta + \epsilon(1-2\eta), m, \eta_b + s/4) &\leq LE(\eta_b + s/2, m, \eta_b + s/4), \\ &\quad \text{by the assumption } \eta_b \leq \eta + \epsilon(1-2\eta)/2 \\ &\leq e^{-2(s/4)^2 m}, \\ &\quad \text{by the Hoeffding's Inequality lemma.} \end{aligned}$$

Since there are at most $|F_n|$ Boolean functions in F_n, the probability of producing a Boolean function with error greater than ϵ is less than

$$|F_n| \cdot e^{-2(s/4)^2 m}$$

and by the lower bound on m, it is less than $\delta/2$. Hence with probability at least $1-\delta$, $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ outputs an ϵ-approximation of f_U.

Further since F_n is polynomial-sized, m is a polynomial in n, $1/\epsilon$, $1/\delta$, and $1/(1-2\eta_b)$ and hence $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ runs in time polynomial in n, $1/\epsilon$, $1/\delta$, and $1/(1-2\eta_b)$. □

The above theorem indicates that $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ requires an accurate estimate η_b of the actual noise rate η for polynomial learnability. In the following, however, we show that for any upper bound $\eta_b < 1/2$, by iterating $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ for successively smaller values of η_b (down to almost 0) and picking the best Boolean function among the outputs of $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$, the existence of $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ for F_n implies the polynomial learnability of F_n in the presence of classification noise.

Theorem 5.5 *Suppose that $\eta \le \eta_b < 1/2$. Suppose also that F_n is polynomial-sized. If there exists a noise-tolerant Occam algorithm for F_n, then F_n is polynomially learnable in the presence of classification noise.*

Proof. We will construct by using $OCCAM_{ln}(S,\epsilon,\delta,\eta_b)$ a learning algorithm for F_n that with probability at least $1-\delta$, outputs an ϵ-approximation of f_U from a sampling oracle $EX_\eta()$ with $\eta < 1/2$. The learning algorithm, *POLY-LEARN*, is illustrated in Figure 5.1.

We will show that with probability at least $1-\delta$, *POLY-LEARN* outputs an ϵ-approximation of f_U. Among the successively smaller values η_e from η_b down to almost 0, there will exist an η_e such that $\eta \le \eta_e \le \eta + \epsilon(1-2\eta)/2$ because η_e is decreasing by $\frac{\epsilon(1-2\eta_b)}{2}$ and $\epsilon(1-2\eta_b) \le \epsilon(1-2\eta)$. Then the lower bound on m implies that with probability at least $1-\delta/2$, there will be at least one Boolean function g_j $(1 \le j \le k)$ in the queue Q of *POLY-LEARN* such that

$$\begin{aligned} F(g_j,S)/m &\le \eta_e + \epsilon(1-2\eta_e)/4, \\ &\quad \text{where } \eta \le \eta_e \le \eta + \epsilon(1-2\eta)/2 \\ &\le \eta + 3\epsilon(1-2\eta)/4. \end{aligned}$$

Hence with probability at least $1-\delta/2$, the Boolean function g_i $(1 \le i \le k)$ in Q that minimizes $F(g_i,S)$ must have $F(g_i,S)/m \le \eta + 3\epsilon(1-2\eta)/4$.

The probability that a Boolean function $g \in F_n$ with error greater than ϵ has $F(g,S)/m \le \eta + 3\epsilon(1-2\eta)/4$ is

$$LE(\eta+\epsilon(1-2\eta), m, \eta+3\epsilon(1-2\eta)/4) \le e^{-2(\epsilon(1-2\eta)/4)^2 m},$$

ALGORITHM *POLY-LEARN*
Input:

- $OCCAM_{ln}(S, \epsilon, \delta, \eta_b)$ for a class F_n of Boolean functions,
- A sampling oracle $EX_\eta()$ subject to classification noise,
- A number n and positive fractions ϵ, δ, and η_b, with $0 \leq \eta \leq \eta_b < 1/2$.

Output:

A Boolean function $g \in F_n$.

Procedure:

1. Request a sample S of m examples, where

$$m \geq \frac{8}{\epsilon^2(1-2\eta_b)^2} \ln\left(\frac{2|F_n|}{\delta}\right);$$

2. $\eta_e \leftarrow \eta_b$;
3. $Q \leftarrow emptyqueue$;
4. Repeat:
   4.1. Call $OCCAM_{ln}(S, \epsilon, \delta, \eta_e)$;
   4.2. Add the output to Q;
   4.3. $\eta_e \leftarrow \eta_e - \frac{\epsilon(1-2\eta_b)}{2}$;

   until $\eta_e \leq 0$;
5. Let $Q = \langle g_1, \ldots, g_k \rangle$;
6. Output a Boolean function g_i $(1 \leq i \leq k)$ in Q that minimizes $F(g_i, S)$.

Figure 5.1: Polynomial learning with classification noise

by the Hoeffding's Inequality lemma.

Since there are at most $|F_n|$ Boolean functions in F_n, the probability of producing a Boolean function with error greater than ϵ is less than

$$|F_n| \cdot e^{-2(\epsilon(1-2\eta)/4)^2 m}$$

and by the lower bound on m, it is less than $\delta/2$. Hence with probability at least $1-\delta$, *POLY-LEARN* outputs an ϵ-approximation of f_U.

Further there are at most $\lceil \frac{1}{\epsilon(1-2\eta_b)} \rceil$ repetitions of calling $OCCAM_{ln}(S, \epsilon, \delta, \eta_e)$ and hence k is at most $\lceil \frac{1}{\epsilon(1-2\eta_b)} \rceil$ in *POLY-LEARN*. These are executed in time polynomial in $1/\epsilon$, $1/(1-2\eta_b)$, and the running time of $OCCAM_{ln}(S, \epsilon, \delta, \eta_e)$ which is bounded by a polynomial in n, $1/\epsilon$, $1/\delta$, and $1/(1-2\eta_b)$. Searching a Boolean function g_i in Q that minimizes $F(g_i, S)$ is executed in time polynomial $1/\epsilon$, $\ln(1/\delta)$, and $1/(1-2\eta_b)$, since there are at most $\lceil \frac{1}{\epsilon(1-2\eta_b)} \rceil$ Boolean functions in Q. Therefore *POLY-LEARN* runs in time polynomial in n, $1/\epsilon$, $1/\delta$, and $1/(1-2\eta_b)$. □

In contrast to Theorem 5.2 for the malicious error model, a noise rate is independent of the desired accuracy ϵ and a noise rate close to $1/2$ is achievable in Theorem 5.5.

Chapter 6

An Efficient Robust Algorithm for Learning Decision Lists

Since it is common in the machine learning literature to consider concepts defined on a set of objects in which the objects are described in terms of a set of Boolean attribute-value pairs, the problem of learning Boolean functions from examples have widely been studied both theoretically and empirically. Learning decision trees is a typical example that can be formulated as learning Boolean functions and also has the most successful counterpart of practical applications. One of famous such practical systems is ID3 by Quinlan [Qui86b]. Decision trees are often used for classification tasks and as the representation of acquired knowledge in a learning system. ID3 induces such decision trees from examples. Numerous applications based on ID3 have also been investigated.

Recently Rivest [Riv87] has introduced another useful way, called *decision lists*, to represent Boolean functions and to perform classification tasks. In fact, decision lists are an important class because k-DL (the class of decision lists with conjunctive clauses of size at most k at each decision) properly includes other well-known techniques for representing Boolean functions such as k-CNF, k-DNF, and decision trees of depth at most k. Rivest [Riv87] has shown that k-DL is polynomially learnable in the Valiant's learnability model. However Rivest's learning algorithm for k-DL is not robust for noisy data and he has left an open problem to study whether the Boolean functions in k-DL can be learned efficiently when the classifications of the given examples may be erroneous with some small probability.

In this chapter, we give the affirmative answer for this open problem. We present a noise-tolerant Occam algorithm for k-DL and hence conclude that k-DL is polynomially learnable in the presence of classification noise. This strictly increases the class of Boolean functions that are known to be polynomially learnable in the presence of classification

noise: the only example of a class of Boolean functions is k-CNF that has been shown to be polynomially learnable in the presence of classification noise [AL88] and k-DL properly includes k-CNF.

6.1 Decision Lists

Rivest [Riv87] has introduced a new representation for Boolean functions, called *decision lists*. A *decision list* is a list L of pairs

$$\langle (t_1, v_1), (t_2, v_2), \ldots, (t_r, v_r) \rangle$$

where each t_i is a term in C_k^n, each v_i is a value in $\{0,1\}$, and the last term t_r is the unique term **true**. A decision list L defines a Boolean function as follows: for any assignment $\vec{a} \in X_n$, $L(\vec{a})$ is defined to be equal to the value v_i where i is the *least* index such that $t_i(\vec{a}) = 1$. (Such an item always exists, since the last term always **true**.) Let k-DL denote the class of all Boolean functions defined by decision lists, where each term in the list is of size at most k. As k increases, the class k-DL becomes increasingly expressive. Note that k-DL is closed under complementation (negation).

We may think of a decision list as an extended "**if** $-$ **then** $-$ **elseif** $-$ ... **else** $-$" rule. Or we may think of decision lists as a "linearly ordered" set of *production rules*. For example, the decision list

$$L = \langle (x_1 x_2, 1),\ (\bar{x}_2 x_3 x_5, 0),\ (x_3 x_4, 1),\ (\mathbf{true}, 0) \rangle \tag{6.1}$$

may be diagrammed as in Figure 6.1. For example, $L(1,0,1,1,1) = 0$; this value is specified by the second pair in the decision list.

For the matter of notations, when we wish to emphasize the number of variables upon which a class of Boolean functions depends, we will indicate this in parentheses after the class name, as in k-CNF(n), k-DNF(n), or k-DL(n).

Decision lists are a strict generalization of the representation techniques for Boolean functions described before.

Theorem 6.1 (Rivest [Riv87]) *For $0 < k < n$, k-CNF(n) and k-DNF(n) are proper subclasses of k-DL(n). For $0 < k < n$ and $n > 2$, (k-CNF(n) $\cup$ k-DNF(n)) is a proper subclass of k-DL(n).*

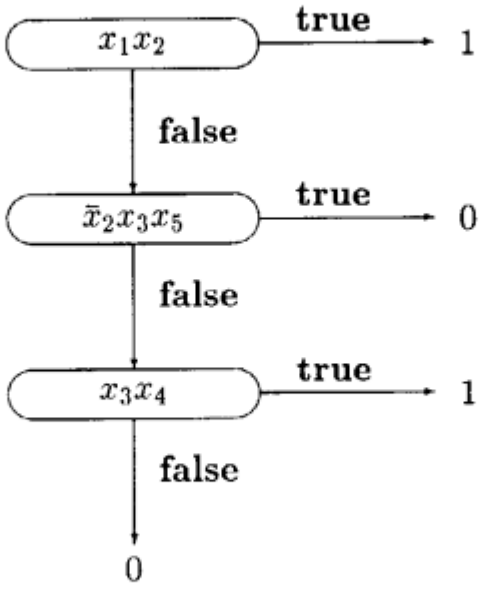

Figure 6.1: Diagram of the decision list $\langle (x_1x_2, 1), (\bar{x}_2x_3x_5, 0), (x_3x_4, 1), (\mathbf{true}, 0)\rangle$.

6.2 Efficient Robust Learning of k-DL

Now we consider an efficient robust algorithm for learning decision lists in the presence of classification noise. We show that there exists a noise-tolerant Occam algorithm for k-DL and hence k-DL is polynomially learnable in the presence of classification noise.

Let L_U denote the target decision list in k-DL(n) over n variables to be learned. We say a pair (t, v) *disagrees with* an example $\langle \vec{a}, l\rangle$ if $t(\vec{a}) = 1$ and $v \neq l$. We say a pair (t, v) is *correct w.r.t.* a sample S drawn from $EX_\eta()$ for L_U if for every example $\langle \vec{a}, l\rangle$ in S such that $t(\vec{a}) = 1$, $v = L_U(\vec{a})$.

We begin with the trivial fact observed by Rivest [Riv87] in the absence of noise that *if a decision list is consistent with a sample S, then it is consistent with any subset of S.* This can be restated in the presence of classification noise as follows: *if a sample S is drawn from $EX_\eta()$ for a decision list L_U, then for any subset S' of S there exists a correct pair w.r.t. S'.*

Now we present an efficient robust learning algorithm for k-DL. Our noise-tolerant Occam algorithm *NODL* for k-DL is shown in Figure 6.2. Here we try to give the intuitive explanation of the algorithm *NODL*. We say a pair (t, v) *explain* an example $\langle \vec{a}, l\rangle$ if

$t(\vec{a}) = 1$. Given the sample S, *NODL* proceeds by identifying the pairs of the decision list in order. *NODL* selects as the first pair of the decision list any pair (t, v) of $C_k^n \times \{0,1\}$ if it disagrees with a small fraction of the examples that are explained by it in S or the number of examples explained by it is below a given threshold. *NODL* then proceeds to delete from S any example explained by the chosen pair (t, v), and to construct the remainder of the decision list in the same way using the remaining part of S.

Let $S = \langle \vec{a}_1, l_1 \rangle, \langle \vec{a}_2, l_2 \rangle, \ldots, \langle \vec{a}_m, l_m \rangle$ be a sample drawn from an $EX_\eta()$ oracle. For a decision list L, let $F(L, S)$ denote the number of indices j for which L disagrees with $\langle \vec{a}_j, l_j \rangle$. For a pair $(t, v) \in C_k^n \times \{0,1\}$, let $F((t, v), S)$ denote the number of indices j for which (t, v) disagrees with $\langle \vec{a}_j, l_j \rangle$ and let $T((t, v), S)$ denote the number of indices j for which $t(\vec{a}_j) = 1$.

Lemma 6.2 *Suppose that NODL is given a sample S of m examples drawn from $EX_\eta()$ and parameters n, k, ϵ, and η_b. If NODL halts and outputs a decision list L, then*

$$\frac{F(L,S)}{m} \leq \eta_b + \frac{\epsilon(1-2\eta_b)}{4}.$$

Proof. Let $L = \langle (t_1, v_1), \ldots, (t_r, v_r) \rangle$. For i-th item (t_i, v_i) of L, let $F_0((t_i, v_i), S)$ denote the number of examples $\langle \vec{a}, l \rangle$ in S such that (t_i, v_i) disagrees with $\langle \vec{a}, l \rangle$ and i is the least index such that $t_i(\vec{a}) = 1$, and let $T_0((t_i, v_i), S)$ denote the number of examples $\langle \vec{a}, l \rangle$ in S for which i is the least index such that $t_i(\vec{a}) = 1$. Any decision list L output by *NODL* has the property that for any i $(1 \leq i \leq r)$,

$$\frac{F_0((t_i, v_i), S)}{T_0((t_i, v_i), S)} \leq \eta_b + \frac{\epsilon(1-2\eta_b)}{8}$$

or

$$\frac{T_0((t_i, v_i), S)}{m} \leq Q_1.$$

Therefore the total number of disagreements of L with S, that is $F(L, S)$, is at most the sum of

$$\begin{aligned} \sum_{i=1}^{r} F_0((t_i, v_i), S) &\leq \sum_{i=1}^{r} (\eta_b + \epsilon(1-2\eta_b)/8) \cdot T_0((t_i, v_i), S) \\ &= (\eta_b + \epsilon(1-2\eta_b)/8) \cdot \sum_{i=1}^{r} T_0((t_i, v_i), S) \\ &= (\eta_b + \epsilon(1-2\eta_b)/8)m \end{aligned}$$

and

$$\begin{aligned} Q_1 m \cdot \frac{1}{2} \cdot r &\leq \frac{1}{2} \frac{\epsilon(1-2\eta_b)}{4M} mM \\ &\leq (\epsilon(1-2\eta_b)/8)m \end{aligned}$$

ALGORITHM *NODL*
Input:

- A sample S of m examples drawn from $EX_\eta()$ subject to classification noise,
- Numbers n, k and positive fractions ϵ, and η_b, with $0 \leq \eta \leq \eta_b < 1/2$.

Output:

A decision list L in k-DL(n).

Procedure:

1. Calculate the following:

$$\begin{aligned} C_k^n &= \{t \mid t \text{ is a term over } n \text{ variables with at most } k \text{ literals}\}, \\ M &= |C_k^n|, \\ Q_I &= \frac{\epsilon(1-2\eta_b)}{4M}; \end{aligned}$$

2. $SS \leftarrow S$; $CC \leftarrow C_k^n \times \{0,1\}$; $L \leftarrow$ *emptylist*; $i \leftarrow 1$;
3. Repeat:
   3.1. If SS is empty, then output L and halt;
   3.2. Find a pair (t,v) in CC such that

$$\frac{F((t,v),SS)}{T((t,v),SS)} \leq \eta_b + \frac{\epsilon(1-2\eta_b)}{8};$$

   3.3. If no such pair can be found, then find a pair (t,v) in CC such that

$$\frac{T((t,v),SS)}{m} \leq Q_I$$

   and $v = 1$ if $F((t,1),SS) \leq F((t,0),SS)$ and $v = 0$ otherwise;
   3.4. Let T denote those examples in SS which make t true;
   3.5. Add the pair (t,v) to L as the i-th item of the decision list L;
   3.6. $SS \leftarrow SS - T$; $CC \leftarrow CC - (t,v)$; $i \leftarrow i+1$;

   until it halts.

Figure 6.2: Efficient robust learning of k-DL

since $r \leq M$. Hence $F(L,S)/m \leq \eta_b + \epsilon(1-2\eta_b)/4$. □

Lemma 6.3 *Suppose that $\eta \leq \eta_b < 1/2$. Suppose also that*

$$m \geq \frac{128M}{\epsilon^3(1-2\eta_b)^3} \ln\left(\frac{2^{M+1}M}{\delta}\right).$$

Then with probability at least $1-\delta/2$, NODL halts and outputs a decision list L.

Proof. First we prove that with probability at least $1-\delta/2$, a sample S is drawn such that for all choices $t_1, t_2, \ldots, t_j$ of terms in C_k^n and for all $t \in C_k^n$, whenever $T((t,v),R) \geq Q_I m$, the number of occurrences of classification noise in $\{\langle \vec{a}, l\rangle \in R \mid t(\vec{a}) = 1\}$ is at most $(\eta_b + \epsilon(1-2\eta_b)/8) \cdot T((t,v),R)$, where $R = S - \{\langle \vec{a}, l\rangle \in S \mid t_1(\vec{a}) = 1\} - \cdots - \{\langle \vec{a}, l\rangle \in S \mid t_j(\vec{a}) = 1\}$. Let $t_1, t_2, \ldots, t_j$ in C_k^n and $t \in C_k^n$ be fixed. By the Hoeffding's inequality lemma, whenever $T((t,v),R) \geq Q_I m$, the probability that classification noise occurs more than $(\eta_b + \epsilon(1-2\eta_b)/8) \cdot T((t,v),R)$ times is at most

$$\begin{aligned} GE(\eta, T((t,v),R), \eta_b + \epsilon(1-2\eta_b)/8) &\leq GE(\eta_b, Q_I m, \eta_b + \epsilon(1-2\eta_b)/8) \\ &\leq e^{-2(\epsilon(1-2\eta_b)/8)^2 Q_I m} \end{aligned}$$

and the lower bound on m implies that this is less than $\delta/(2^{M+1}M)$. Since there are at most 2^M choices of terms in C_k^n and M terms in C_k^n, the probability that for all choices $t_1, t_2, \ldots, t_j$ of terms in C_k^n and for all $t \in C_k^n$, the number of occurrences of classification noise in $\{\langle \vec{a}, l\rangle \in R \mid t(\vec{a}) = 1\}$ is at most $(\eta_b + \epsilon(1-2\eta_b)/8) \cdot T((t,v),R)$ is at least $1-\delta/2$. This completes the proof.

Next we assume that for all choices $t_1, t_2, \ldots, t_j$ of terms in C_k^n and for all $t \in C_k^n$, the number of occurrences of classification noise in $\{\langle \vec{a}, l\rangle \in R \mid t(\vec{a}) = 1\}$ is at most $(\eta_b + \epsilon(1-2\eta_b)/8) \cdot T((t,v),R)$ whenever $T((t,v),R) \geq Q_I m$. We show that this assumption implies that *NODL* halts and outputs a decision list L. For any stage, say i, in the repetition in *NODL* where SS is not empty, there is at least one correct pair (t,v) w.r.t. SS in CC. If $T((t,v),SS)/m \leq Q_I$, then *NODL* can select the pair (t,v) as the i-th item of L. If $T((t,v),SS)/m \geq Q_I$, by the above assumption, the correct pair (t,v) has at most $(\eta_b + \epsilon(1-2\eta_b)/8) \cdot T((t,v),SS)$ disagreements with SS. Then *NODL* can select the pair (t,v) as the i-th item of L. Hence eventually *NODL* halts and outputs a decision list L.

Therefore with probability at least $1-\delta/2$, *NODL* halts and outputs a decision list L. □

Theorem 6.4 *NODL is a noise-tolerant Occam algorithm for k-DL(n).*

Proof. Since $|C_k^n| \leq (2n+1)^k$, it is clear that *NODL* runs in time polynomial in n and m. Then it is straightforward from Lemmas 6.2 and 6.3. □

We quote the following important lemma by Rivest [Riv87].

Lemma 6.5 (Rivest [Riv87]) *k-DL(n) is polynomial-sized.*

Proof. $|k\text{-DL}(n)| = O(3^{|C_k^n|}(|C_k^n|)!)$. This implies that $\ln(|k\text{-DL}(n)|) = O(n^t)$ for some constant t. □

Now we have the main theorem.

Theorem 6.6 *k-DL(n) is polynomially learnable in the presence of classification noise.*

Proof. It is straightforward from Theorems 6.4, 5.5, and Lemma 6.5. □

Schapire [Sch91] had independently achieved this result by using probabilistic decision lists which are a probabilistic analog of (deterministic) decision lists. Schapire has shown that a special class of probabilistic decision lists with conjunctive clauses of size at most k at each decision can be learned efficiently and the result can be applied to learn ordinary decision lists when the supplied examples are noisy.

Chapter 7

An Efficient Robust Algorithm for Learning Decision Trees

Recently Ehrenfeucht and Haussler [EH89] have introduced the notion of the *rank* of a decision tree and shown that the class of all decision trees of rank at most r on n Boolean variables is polynomially learnable in the Valiant's learnability model and Rivest's result for decision lists can be interpreted as a special case of their result for rank 1. However their learning algorithm is not robust for noisy data and hence an efficient robust learning algorithm for decision trees of rank at most r has been expected to be developed because when we study the problem of learning decision trees concerned with a large amount of learning data, it is practical to assume that the data contain some noise. Few empirical works for decision trees have suggested any way to make their learning algorithms noise tolerant and there has been no theoretical treatment for learning decision trees from noisy examples so far.

In Chapter 5, we have developed a technique of building efficient robust learning algorithms, *noise-tolerant Occam algorithm*, in the classification noise process and shown that using a noise-tolerant Occam algorithm for a class of Boolean functions, one can construct a polynomial-time algorithm for learning the class in the presence of classification noise. In Chapter 6, we have presented a noise-tolerant Occam algorithm for decision lists and hence concluded that the decision lists is polynomially learnable in the presence of classification noise.

In this chapter, we extend the noise-tolerant Occam algorithm for decision lists to the one for decision trees and conclude that the class of all decision trees of rank at most r on n Boolean variables is polynomially learnable in the presence of classification noise. This result can hold at a noise rate even close to $1/2$.

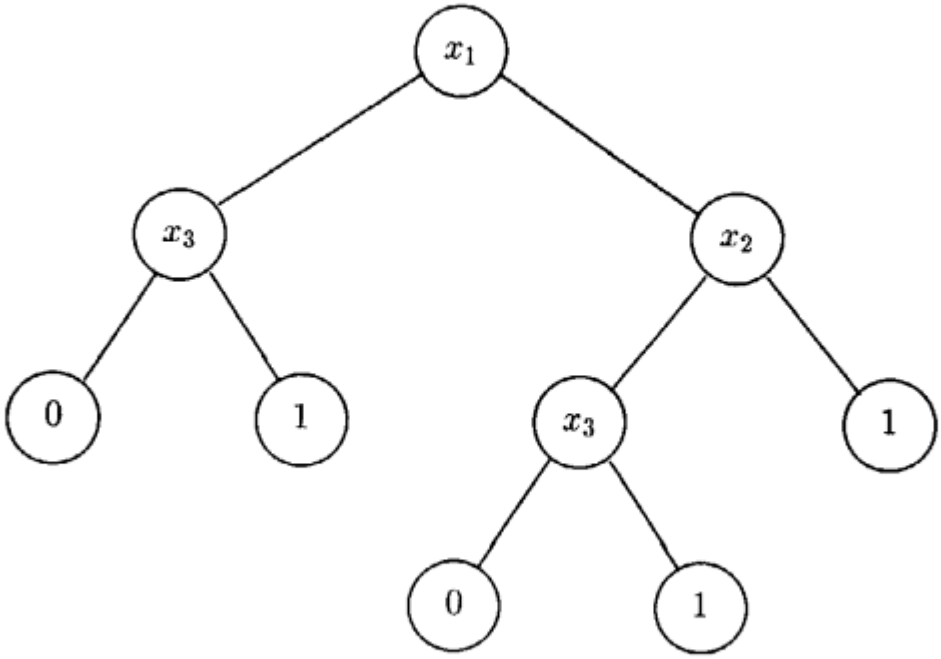

Figure 7.1: A decision tree representation for $x_1x_2 \vee x_3$.

7.1 Decision Trees

First we give formal definitions of decision trees and their rank introduced by Ehrenfeucht and Haussler [EH89].

A *decision tree* is a binary tree where each internal node is labelled with a variable and each leaf is labelled with 0 or 1. A decision tree is a useful way to represent a Boolean function. The class $\mathcal{T}_n$ of *decision trees* over V_n is defined recursively as follows:

1. If T is the binary tree consisting of only a root node labelled either 0 or 1, then $T \in \mathcal{T}_n$. (We will abbreviate this case by simply saying "$T = 0$" or "$T = 1$".)

2. If $T_0, T_1 \in \mathcal{T}_n$ and $x \in V_n$, then the binary tree with root labelled x, left subtree T_0, and right subtree T_1 is in $\mathcal{T}_n$. (We will refer to the left subtree as the 0-*subtree* and the right subtree as the 1-*subtree*.)

A decision tree $T \in \mathcal{T}_n$ defines a Boolean function f_T as follows:

1. If $T = 0$ then f_T is the constant function 0 and if $T = 1$ then f_T is the constant function 1.

2. Else if x_i is the variable labelled at the root node, T_0 the 0-subtree and T_1 the 1-subtree, then for any assignment $\vec{a} = (a_1, \ldots, a_n)$, if $a_i = 0$ then $f_T(\vec{a}) = f_{T_0}(\vec{a})$, else $f_T(\vec{a}) = f_{T_1}(\vec{a})$.

A decision tree is *reduced* if each variable appears at most once in any path from the root to a leaf.

The *rank* of a decision tree T, denoted $r(T)$, is defined as follows:

1. If $T = 0$ or $T = 1$ then $r(T) = 0$.

2. Else if r_0 is the rank of the 0-subtree of T and r_1 is the rank of the 1-subtree, then

$$r(T) = \begin{cases} \max(r_0, r_1) & \text{if } r_0 \neq r_1 \\ r_0 + 1 (= r_1 + 1) & \text{otherwise.} \end{cases}$$

Let r-DT(n) denote the set of all Boolean functions on X_n represented by decision trees of rank at most r.

It is easily verified that every function in r-DT(n) can be represented by a reduced decision tree of rank at most r.

We quote the following important lemma by Ehrenfeucht and Haussler [EH89].

Lemma 7.1 (Ehrenfeucht & Haussler [EH89]) *1. Let $L(n,r)$ denote the maximum number of leaves of any reduced decision trees over V_n of rank r. Then*

$$\begin{aligned} L(0,r) &= 1 \quad \textit{for all } r \geq 0, \\ L(n,0) &= 1 \quad \textit{for all } n \geq 0, \\ L(n,r) &= L(n-1,r) + L(n-1,r-1) \quad \textit{for all } n,r \geq 1. \end{aligned}$$

Further $L(n,r)$ is bounded by $(en/r)^r$ for all $n \geq r \geq 1$.

2. If $r < n$, then $|r\text{-}DT(n)| \leq (8n)^{(en/r)^r}$.

7.2 Efficient Robust Learning of Decision Trees of rank r

Now we present a noise-tolerant Occam algorithm for r-DT(n) and hence conclude that r-DT(n) is polynomially learnable in the presence of classification noise.

Let $S = \{\langle \vec{a}_1, l_1\rangle, \langle \vec{a}_2, l_2\rangle, \ldots, \langle \vec{a}_m, l_m\rangle\}$ be a sample drawn from an $EX_\eta()$ oracle. For a decision tree T, let $F(T, S)$ denote the number of indices j for which T disagrees with $\langle \vec{a}_j, l_j\rangle$.

Let x be a variable in V_n. Assume $x = x_i$, where $1 \leq i \leq n$. Then S_0^x denotes the set of all examples $\langle \vec{a}, l\rangle$ in a sample S such that $\vec{a} = (a_1, \ldots, a_n)$ and $a_i = 0$, and S_1^x denotes the set of all examples $\langle \vec{a}, l\rangle$ in S such that $\vec{a} = (a_1, \ldots, a_n)$ and $a_i = 1$. We say a variable x is *informative* (on S) if both S_0^x and S_1^x are nonempty.

We give a noise-tolerant Occam algorithm for r-DT(n) in Figures 7.2 and 7.3. The subroutine *RFINDT* that finds a decision tree of rank at most r consistent with a large fraction of the given sample is an extension of the algorithm *NODL* for k-DL and based on the FIND procedure by Ehrenfeucht and Haussler [EH89] to find a decision tree of rank at most r consistent with the given sample in the absence of noise.

Now we show that *NODT* is a noise-tolerant Occam algorithm for decision trees of rank r and hence the class of decision trees of rank at most r is polynomially learnable in the presence of classification noise for any $n \geq r \geq 0$. Throughout the following sequence of lemmas and theorems, we assume that $n \geq r \geq 0$.

Lemma 7.2 *Suppose* $Q_F = \eta_b + \frac{\epsilon(1-2\eta_b)}{8}$ *and* $Q_I = \frac{\epsilon(1-2\eta_b)}{4(en/r)^r}|S|$. *If* $RFINDT(S, r, Q_F, Q_I)$ *outputs a decision tree* T *(not "none"), then* T *is a decision tree of rank at most* r *and*

$$\frac{F(T,S)}{|S|} \leq \eta_b + \frac{\epsilon(1-2\eta_b)}{4}.$$

Proof. First it is clear that if $RFINDT(S, r, Q_F, Q_I)$ returns a decision tree (which occurs either in step 1, 4b, or 4c), then by the definitions of reduced and rank it will be a reduced decision tree over V_n of rank at most r.

Next any decision tree T output by *RFINDT* has the following property. For any leaf, say j, of T, let $S(j)$ denote the set of all examples in S which reaches the leaf j. Then $F(T, S(j))/|S(j)| \leq Q_F$ or $|S(j)| \leq Q_I$. Therefore the total number of disagreements of T with S, that is $F(T, S)$, is at most the sum of

$$\begin{aligned}\sum_{\text{all leaves } j} F(T, S(j)) &\leq \sum_{\text{all leaves } j} Q_F \cdot |S(j)| \\ &= Q_F \cdot \sum_{\text{all leaves } j} |S(j)| \\ &= (\eta_b + \epsilon(1-2\eta_b)/8)|S|\end{aligned}$$

and

$$Q_I \cdot \frac{1}{2} \cdot \left(\frac{en}{r}\right)^r \leq \frac{1}{2}\frac{\epsilon(1-2\eta_b)}{4(en/r)^r}|S|(en/r)^r$$

ALGORITHM *NODT*
Input:

- A sample S of m examples drawn from $EX_\eta()$ subject to classification noise,
- Integers $n, r \geq 0$ and positive fractions ϵ, and η_b, with $0 \leq \eta \leq \eta_b < 1/2$.

Output:

A decision tree T of rank at most r such that $F(T,S)/m \leq \eta_b + \epsilon(1-2\eta_b)/4$ if one exists, else "none".

Procedure:

1. Calculate the following:

$$\begin{aligned} Q_F &= \eta_b + \frac{\epsilon(1-2\eta_b)}{8}; \\ Q_I &= \frac{\epsilon(1-2\eta_b)}{4(en/r)^r}|S|; \end{aligned}$$

2. Call $RFINDT(S, r, Q_F, Q_I)$;
3. Let $T = RFINDT(S, r, Q_F, Q_I)$;
4. Output T and halt.

Figure 7.2: Efficient robust learning of decision trees of rank r

ALGORITHM $RFINDT(S, r, Q_F, Q_I)$
Input:

A sample S, integers $r, Q_I \geq 0$ and a positive fraction Q_F.

Output:

A decision tree T of rank at most r or "none".

Procedure:

1. If $F(1,S)/|S| \leq Q_F$, stop and return the decision tree $T = 1$;
   If $F(0,S)/|S| \leq Q_F$, stop and return the decision tree $T = 0$;
2. If $|S| \leq Q_I$ and $F(1,S) \leq F(0,S)$, stop and return the decision tree $T = 1$;
   If $|S| \leq Q_I$ and $F(0,S) \leq F(1,S)$, stop and return the decision tree $T = 0$;
3. If $r = 0$, stop and return "none";
4. For each informative variable $x \in V_n$
   (a) Let $T_0^x = RFINDT(S_0^x, r-1, Q_F, Q_I)$ and $T_1^x = RFINDT(S_1^x, r-1, Q_F, Q_I)$;
   (b) If both recursive calls are successful (i.e., neither T_0^x ="none", nor T_1^x ="none"), then stop and return the decision tree with root labelled x, 0-subtree T_0^x and 1-subtree T_1^x;
   (c) If one recursive call is successful but the other is not, then
      i. Reexecute the unsuccessful recursive call with rank bound r instead of $r-1$;
      ii. If the reexecuted call is now successful, then let T be the decision tree with root labelled x, 0-subtree T_0^x and 1-subtree T_1^x, else let T ="none";
      iii. Stop and return T;
5. Stop and return "none".

Figure 7.3: Finding a decision tree of rank r

$$\leq (\epsilon(1-2\eta_b)/8)|S|$$

since the maximum number of leaves of any reduced decision trees over V_n of rank r is bounded by $(en/r)^r$ by Lemma 7.1. Hence $F(T,S)/|S| \leq \eta_b + \epsilon(1-2\eta_b)/4$. □

Lemma 7.3 *Let T be a reduced decision tree over V_n of rank r, S be a sample consistent with T, and x be a variable which appears in T. Let T_0^x (T_1^x) denote the decision tree obtained by replacing every subtree with root labelled x of T by the 0-subtree (1-subtree) of that subtree. Then T_0^x (T_1^x) is a reduced decision tree of rank at most r consistent with S_0^x (S_1^x).*

Proof. It is straightforward from the definitions of "rank" and "reduced", and the observation that if a decision tree T is consistent with a sample S, then T is consistent with any subset of S, and for every example $\langle \vec{a}, l \rangle$ in S_0^x (S_1^x) and for $x_i = x$ and $\vec{a} = (a_1, \ldots, a_n)$, $a_i = 0$ ($a_i = 1$). □

For a term t over V_n, let $var(t) = \{x \in V_n \mid x \text{ or its negation } \bar{x} \text{ appears in } t\}$. For a sample S and a term t, let $S_{[t]} = \{\langle \vec{a}, l \rangle \in S \mid t(\vec{a}) = 1\}$.

We say a decision tree T is *correct w.r.t.* a sample S drawn from $EX_\eta()$ for a decision tree T_U if for every example $\langle \vec{a}, l \rangle$ in S, $f_T(\vec{a}) = f_{T_U}(\vec{a})$.

Lemma 7.4 *Suppose $Q_F = \eta_b + \frac{\epsilon(1-2\eta_b)}{8}$ and $Q_I = \frac{\epsilon(1-2\eta_b)}{4(en/r)^r}|S|$. Suppose also that S consists of m examples drawn from $EX_\eta()$ for the target decision tree over V_n of rank r and*

$$m \geq \frac{128(en/r)^r}{\epsilon^3(1-2\eta_b)^3} \ln\left(\frac{2 \cdot 3^n}{\delta}\right).$$

Then with probability at least $1-\delta/2$, $RFINDT(S, r, Q_F, Q_I)$ outputs a decision tree T.

Proof. Let T_U denote the target reduced decision tree over V_n of rank r to be learned and a sample S is drawn from $EX_\eta()$ for T_U.

First we prove that with probability at least $1-\delta/2$, a sample S is drawn such that for all terms t over V_n, whenever $|S_{[t]}| \geq Q_I$, the number of occurrences of classification noise in $S_{[t]}$ is at most $Q_F \cdot |S_{[t]}|$. Let a term t over V_n be fixed. By the Hoeffding's inequality lemma, whenever $|S_{[t]}| \geq Q_I$, the probability that classification noise occurs more than $Q_F \cdot |S_{[t]}|$ times is at most

$$\begin{aligned} GE(\eta, Q_I, Q_F) &\leq GE(\eta_b, Q_I, \eta_b + \epsilon(1-2\eta_b)/8) \\ &\leq e^{-2(\epsilon(1-2\eta_b)/8)^2 Q_I} \end{aligned}$$

and the lower bound on m implies that this is less than $\delta/(2 \cdot 3^n)$. Since there are at most 3^n terms over V_n, the probability that for all terms t over V_n, the number of occurrences of classification noise in $S_{[t]}$ is at most $Q_F \cdot |S_{[t]}|$ is at least $1 - \delta/2$. This completes the proof.

Second we assume that for all terms t over V_n, the number of occurrences of classification noise in $S_{[t]}$ is at most $Q_F \cdot |S_{[t]}|$ whenever $|S_{[t]}| \geq Q_I$. We show that this assumption implies (*) that for a term t over V_n, if there is a correct reduced decision tree T over $V_n - var(t)$ of rank at most r' w.r.t. $S_{[t]}$, then $RFINDT(S_{[t]}, r', Q_F, Q_I)$ outputs a decision tree. We prove it by induction on r' and the number i of variables in $V_n - var(t)$.

If $r' = 0$ or $i = 0$, a correct reduced decision tree T of rank r' w.r.t. $S_{[t]}$ consists of only a root node labelled either 0 or 1 (i.e., $T = 0$ or $T = 1$). By the above assumption, $|S_{[t]}| \geq Q_I$, $F(0, S_{[t]}) \leq Q_F \cdot |S_{[t]}|$ or $F(1, S_{[t]}) \leq Q_F \cdot |S_{[t]}|$. Hence $RFINDT$ outputs a decision tree.

Next suppose that (*) holds for $r' - 1$ and i, and for r' and $i - 1$. In the case that a correct decision tree T w.r.t. $S_{[t]}$ consists of only a root node (i.e., $T = 0$ or $T = 1$), by the above assumption, $RFINDT$ outputs a decision tree $T = 0$ or $T = 1$. In the case that T has more than two internal nodes, we prove that $RFINDT$ cannot return "none" in step 5 and 4c. First we prove that $RFINDT$ cannot reach step 5 and return "none". Let y be the variable labelled at the root node in a correct reduced decision tree T of rank r'. Since $RFINDT$ will eventually find the variable y when both recursive calls for any other informative variable x in step 4a are not successful, we assume that $RFINDT$ chooses the variable y as an informative variable in step 4. (In the case that the variable y is not informative on $S_{[t]}$, either $S_{[t]_0^y}$ or $S_{[t]_0^y}$ is empty. We assume that $S_{[t]_0^y}$ is empty. Then $S_{[t]} = S_{[t]_1^y} = S_{[t \wedge y]}$ and by Lemma 7.3, the 1-subtree of T is a correct reduced decision tree over $V_n - var(t \wedge y)$ of rank at most r' w.r.t. $S_{[t \wedge y]}$. By the inductive hypothesis, $RFINDT(S_{[t \wedge y]}, r', Q_F, Q_I)$ outputs a decision tree, and so does $RFINDT(S_{[t]}, r', Q_F, Q_I)$.) By the definition of rank,

(1) both 0-subtree and 1-subtree of T have the rank $r' - 1$ or

(2) either 0-subtree or 1-subtree of T has the rank at most $r' - 1$ and the other has the rank r'.

In both cases, by Lemma 7.3, the 0-subtree of T is a correct reduced decision tree over $V_n - var(t) - \{y\}$ w.r.t. $S_{[t]_0^y}$ and the 1-subtree is a correct reduced decision tree over $V_n - var(t) - \{y\}$ w.r.t. $S_{[t]_1^y}$. By the inductive hypothesis, both recursive calls $RFINDT$

for $S_{[t]_0^y}$ and $S_{[t]_1^y}$ are successful, and hence $RFINDT(S_{[t]}, r', Q_F, Q_I)$ outputs a decision tree.

Next we prove that $RFINDT$ cannot return "none" in step 4c. Assume that in step 4a, $RFINDT(S_{[t]_0^x}, r'-1, Q_F, Q_I)$ is successful for an informative variable x. Note that any informative variable does not appear in the term t. If x does not appear in T, then T is a correct reduced decision tree over $V_n - var(t) - \{x\}$ of rank r' w.r.t. $S_{[t]_1^x} = S_{[t \wedge x]}$. If x appears in T, then by Lemma 7.3, T_1^x is a correct reduced decision tree over $V_n - var(t) - \{x\}$ of rank at most r' w.r.t. $S_{[t]_1^x}$. By the inductive hypothesis, $RFINDT(S_{[t]_1^x}, r', Q_F, Q_I)$ returns a decision tree. This completes the proof of (*).

Since T_U is a correct reduced decision tree over V_n of rank r w.r.t. S, $RFINDT(S, r, Q_F, Q_I)$ outputs a decision tree with probability at least $1 - \delta/2$. □

Lemma 7.5 *For any nonempty sample S drawn from $EX_\eta()$ for a decision tree over V_n and $r \geq 0$, the running time of $RFINDT(S, r, Q_F, Q_I)$ is $O(|S|(n+1)^{2r})$.*

Proof. The proof is almost same as the one for the time analysis of the procedure FIND by Ehrenfeucht and Haussler in [EH89].

Let $t(i, r)$ be the maximum running time needed for $RFINDT(S, r, Q_F, Q_I)$ when S is drawn from $EX_\eta()$ for a decision tree over V_n of rank at most r and at most i variables are informative on S. Let $m = |S|$.

If $i = 0$, then $t(i, r)$ is $O(1)$, since $|S| = 1$ in this case. If $r = 0$, then $t(i, r)$ is clearly $O(m)$. If $r \geq 1$, then the time required to test whether $F(1, S)/|S| \leq Q_F$ or $F(0, S)/|S| \leq Q_F$ (step 1), to test whether $|S| \leq Q_I$ (step 2), to determine which variables are informative (step 4), and to perform the other miscellaneous tests is $O(mn)$. Each of the two recursive calls for an informative variable x in step 4a takes time at most $t(i-1, r-1)$ since the variable x is no longer informative in either S_0^x or S_1^x. Since there are at most i informative variables on S, these calls are made at most i times in the course of the loop of step 4, which gives a total time for all executions of step 4a of at most $2it(i-1, r-1)$. The only remaining step is 4(c)i, where a recursive call is made either to $RFINDT(S_0^x, r, Q_F, Q_I)$ or $RFINDT(S_1^x, r, Q_F, Q_I)$ for some informative variable x. This takes time at most $t(i-1, r)$. Since this step terminates the loop, this call is made at most once. It follows that for $r \geq 1$,

$$t(i, r) \leq O(mn) + 2i \cdot t(i-1, r-1) + t(i-1, r),$$

which is bounded by $O(mn(i+1)^{2r-1} + m(i+1)^{2r})$ (see Ehrenfeucht and Haussler [EH89]).

Since $i \leq n$ and $m = |S|$, this implies that the running time for $RFINDT(S, r, Q_F, Q_I)$ is $O(|S|(n+1)^{2r})$. □

Theorem 7.6 *NODT is a noise-tolerant Occam algorithm for decision trees of rank r.*

Proof. It is straightforward from Lemmas 7.2, 7.4, and 7.5. □

Now we have the main theorem.

Theorem 7.7 *r-DT(n) is polynomially learnable in the presence of classification noise.*

Proof. It is straightforward from Theorems 5.5, 7.6, and Lemma 7.1. □

Elomaa and Kivinen [EK91] had independently achieved this result based on our result (Theorem 5.5) for noise-tolerant Occam algorithms and by using the same technique as ours shown in Chapter 6 for proving the polynomial learnability of k-DL in the presence of noise.

Recently Blum [Blu91] has shown that every decision tree of rank r can be represented as a decision list in r-DL, that is, r-DT(n) is a subclass of r-DL(n). The idea to construct a decision list in r-DL corresponding to an arbitrary decision tree of rank r is as follows: Find a leaf that is closest to the root in the decision tree. Form a term that corresponds to the path from the root to the leaf. Add a pair of the term and the label of the leaf to the decision list. Delete the leaf and the node closest to the leaf from the tree. Repeat the above procedure until no leaf exists.

Combined with this result, it is straightforward to show that r-DT(n) is polynomially learnable in the presence of classification noise in terms of r-DL(n) by using *NODL*. Now it is interesting for us to compare the sample size needed by *NODT* with the one by *NODL*. By Lemma 7.4, *NODT* gives the sample size $m \geq \frac{128(en/r)^r}{\epsilon^3(1-2\eta_b)^3} \ln\left(\frac{2 \cdot 3^n}{\delta}\right)$. By Lemma 6.3, *NODL* gives the sample size $m \geq \frac{128M}{\epsilon^3(1-2\eta_b)^3} \ln\left(\frac{2^{M+1}M}{\delta}\right)$, where $M = O(n^r)$. Thus *NODT* might give a better bound than *NODL*.

Chapter 8

Conclusions

We briefly summarize the results presented in this thesis with some future problems. The material of this thesis focuses on two topics. The first topic is the study of learning formal languages from queries. Two learning algorithms, *RCFG* and *CFGQ*, for context-free grammars presented in Chapters 2 and 3 use information on the grammatical structure of the unknown language, and it has been shown that such an additional information is quite useful to break inherent difficulties in learning formal languages (e.g., the weakness of learning from positive presentations and the computational hardness of learning context-free grammars from queries).

In Chapter 2, we have introduced a new normal form for context-free grammars, reversible context-free grammars, and shown that the class of reversible context-free grammars can be identified in the limit from positive structural presentations. We have demonstrated the algorithm *RCFG* for learning reversible context-free grammars that runs in time polynomial in the sum of the sizes of the input skeletons. However this result does not imply "polynomial-time identification in the limit" in some sense. Pitt [Pit89] has recently discussed and analyzed various definitions for *polynomial-time identification in the limit* to find natural formal definitions that capture the notion of "efficient" identification in the limit. In the course of his study, he has proposed two important notions that seem to be needed for natural definitions, *polynomial update time* and *implicit error of prediction*. His work is notable here, so we describe these notions by using our terminology in the case of identification of tree automata. (See [Pit89] for the general definition.) The algorithm implemented so as to have *polynomial update time* is allowed at most $p(n, m_1 + m_2 + \cdots + m_i)$ running time to output it's i-th output, where p is any polynomial function of two variables, n is the number of states of the unknown tree automaton A, and m_j $(1 \leq j \leq i)$ is the size of j-th input skeleton in a presentation of

A. The algorithm is said to make *implicit error of prediction* at stage i if it's i-th output of tree automata does not accept $(i+1)$-st input skeleton. Ultimately, he has arrived at what he believes to be one of few possible natural definitions for polynomial-time identification in the limit: Tree automata can be *identified in the limit in polynomial time* if and only if there exists an algorithm M such that for any tree automaton A, M has polynomial update time and the number of implicit errors of prediction made by M is at most $q(n)$, where q is a polynomial and n is the number of states of A. By Theorem 2.20, the algorithm RTA_∞ has polynomial update time. However according to the result of Angluin [Ang90], no algorithm for learning reversible skeletal tree automata can achieve the polynomial number of implicit errors of prediction. Thus the number of implicit errors of prediction made by RTA_∞ is not bounded by any polynomial in n. Theorem 2.22 only shows that the number of implicit errors of prediction made by RTA_∞ is finite. An interesting future work concerned with this problem is to find a class of context-free grammars which is a normal form for context-free grammars and can be identified in the limit from positive structural presentations in polynomial time in the sense of Pitt.

In Chapter 3, we have considered the problem of learning the whole class of context-free grammars using queries and shown that the whole class of context-free grammars can be exactly learned from structural membership queries and structural equivalence queries in a polynomial computation time. By an easy argument based on the result in [Ang81], it can be shown that there is no polynomial-time algorithm using only structural membership queries for this problem even if we are given a bound on the size of an skeletal tree automaton for the structural descriptions of derivation trees of the unknown grammar. The result of Angluin [Ang90] implies that there is no polynomial-time algorithm using only structural equivalence queries for learning the whole class of context-free grammars. Recently Angluin and Kharitonov [AK91] have shown that assuming the intractability of quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, there is no polynomial-time algorithm for learning context-free grammars from membership and equivalence queries. Thus the problem of learning the whole class of context-free grammars from membership and equivalence queries is computationally as hard as those cryptographic problems, for which there is currently no known polynomial-time algorithm. These observations of negative results indicate that membership queries and information on the grammatical structure are essential to our positive result.

An important future work based on our theoretical results is to find their practical applications. As Crespi-Reghizzi et al. [CRML73] have suggested, the grammatical inference may be useful in specifying programming languages. A practical application of

our algorithm is designing programming languages or synthesis of compilers, because the syntax of a programming language constitutes a context-free language in principle and the structure or syntax of programming languages is defined by means of a context-free grammar. As stated in [CRML73], the definition of grammatical structure and the definition of meaning should be interconnected since structural information is an aid for interpreting a sentence. Hence in an application of formal language learning to designing a programming language, the learned grammar should be constructed so that it not only generates sentences correctly but also assigns to each sentence the structure required by the designer. Then our approach will provide an effective method for the process of programming language design. We [TSO91] are also investigating an application of our learning algorithms to the improvement of usability of a syntax-directed editor. We are constructing a "syntax-directed editor customizable from examples and queries".

The second topic of this thesis is the design and analysis of algorithms for learning decision trees from large data in noisy environment. We have focused on the problem of learning Boolean functions represented by decision trees from large data and assumed that the data contain some noise. In Chapter 5, we have developed a technique of building efficient robust learning algorithms, called *noise-tolerant Occam algorithm*, in the presence of classification noise, and shown that using a noise-tolerant Occam algorithm for a class of Boolean functions, one can construct a polynomial-time algorithm for PAC learning the class in the presence of classification noise. In Chapter 6, we have presented a noise-tolerant Occam algorithm for k-DL and hence conclude that k-DL is polynomially learnable in the presence of classification noise. This strictly increases the class of Boolean functions that are known to be polynomially learnable in the presence of classification noise: the only example of a class of Boolean functions is k-CNF that has been shown to be polynomially learnable in the presence of classification noise [AL88] and k-DL properly includes k-CNF. In Chapter 7, we have extended the noise-tolerant Occam algorithm for decision lists to the one for decision trees and conclude that the class of decision trees of rank at most r is polynomially learnable in the presence of classification noise.

Sloan [Slo89] have introduced two new models between the classification noise process and the malicious error model to show that they can be "pushed towards one another". He has exhibited a more severe model of noise than classification noise process where we can still tolerate the noise rate η less than $1/2$. The noise model, called *malicious misclassification model*, is the following:

> Independently for each example $\langle \vec{a}, l \rangle$, with probability $1 - \eta$, the correct example $\langle \vec{a}, l \rangle$ is returned and with probability η, the example $\langle \vec{a}, l' \rangle$ is returned where l' is a label about which no assumption whatsoever may be made.

He has argued that this model is a very strong model of classification noise and could be used to well model the case where "borderline" examples from the domain are misclassified much more than "obvious" examples, which is a case appealing to intuition. By a simple argument similar to the one presented by Sloan [Slo89], we can strengthen our algorithms, *POLY-LEARN*, *NODL*, and *NODT*, to work in the malicious misclassification model and tolerate the noise rate η less than $1/2$.

It is also important for us to have empirical studies and to see how well our algorithms work in a practical situation. Quinlan's research [Qui86a] is a good empirical study of the effects of different sorts of noise on learning decision trees. It is an important future problem to evaluate our algorithms empirically and compare its result with Quinlan's one.

Bibliography

[Aho68] Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *Journal of the ACM*, 15:647–671, 1968.

[AHU83] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? To appear in *23rd Annual ACM Symposium on Theory of Computing*, 1991.

[AL88] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2:343–370, 1988.

[Ang80a] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[Ang80b] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[Ang81] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[Ang82] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29:741–765, 1982.

[Ang87a] Dana Angluin. Learning k-bounded context-free grammars. Technical Report YALEU/DCS/RR-557, Department of Computer Science, Yale University, 1987.

[Ang87b] Dana Angluin. Learning regular sets from queries and counter-examples. *Information and Computation*, 75:87–106, 1987.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[Ari70] Setsuo Arikawa. Elementary formal systems and formal languages - simple formal systems. *Memoirs of the faculty of science, Kyushu university, Series A*, 24:47–75, 1970.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference : Theory and methods. *ACM Computing Surveys*, 15(3):237–269, 1983.

[ASY89] Setsuo Arikawa, Takeshi Shinohara, and Akihiro Yamamoto. Elementary formal systems as a unifying framework for language learning. In *Proceedings of 2nd Workshop on Computational Learning Theory*, pages 312–327. Morgan Kaufmann, 1989.

[BEHW87] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36:929–965, 1989.

[BFOS84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth Statistics/Probability Series, 1984.

[Blu91] Avrim Blum. Basic argument for rank-k DTs being contained in k-DLs. Unpublished manuscript, 1991.

[BR87] Piotr Berman and Robert Roos. Learning one-counter languages in polynomial time. In *Proceedings of 28th IEEE Symposium on Foundations of Computer Science*, pages 61–67. IEEE Computer Society Press, 1987.

[Bra68] Walter S. Brainerd. The minimalization of tree automata. *Information and Control*, 13:484–491, 1968.

[CR72] Stefano Crespi-Reghizzi. An effective model for grammar inference. In B. Gilchrist, editor, *Information Processing 71*, pages 524–529. Elsevier North-Holland, 1972.

[CRGM78] Stefano Crespi-Reghizzi, Giovanni Guida, and Dino Mandrioli. Noncounting context-free languages. *Journal of the ACM*, 25:571–580, 1978.

[CRML73] Stefano Crespi-Reghizzi, M. A. Melkanoff, and L. Lichten. The use of grammatical inference for designing programming languages. *Communications of the ACM*, 16:83–90, 1973.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989.

[EK91] Tapio Elomaa and Jyrki Kivinen. Learning decision trees from noisy examples. Report A-1991-3, Department of Computer Science, University of Helsinki, 1991.

[Fas83] Leona F. Fass. Learning context-free languages from their structured sentences. *SIGACT News*, 15:24–35, 1983.

[Fis68] Michael J. Fischer. Grammars with macro-like productions. In *Proceedings of 9th Annual IEEE Symposium on Switching and Automata Theory*, pages 131–142, 1968.

[GH72] James N. Gray and Michael A. Harrison. On the covering and reduction problems for context-free grammars. *Journal of the ACM*, 19:675–698, 1972.

[Gol67] E Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Gol78] E Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Ish90] Hiroki Ishizaka. Polynomial time learnability of simple deterministic languages. *Machine Learning*, 5:151–164, 1990.

[KL88] Michael Kearns and Ming Li. Learning in the presence of malicious errors. In *Proceedings of 20th Annual ACM Symposium on Theory of Computing*, pages 267–279. ACM, 1988.

[KLPV87] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of Boolean formulae. In *Proceedings of 19th Annual ACM Symposium on Theory of Computing*, pages 285–295. ACM, 1987.

[Lai88] Philip D. Laird. *Learning from Good and Bad Data.* Kluwer Academic, 1988.

[LJ78] Leon S. Levy and Aravind K. Joshi. Skeletal structural descriptions. *Information and Control*, 39:192–211, 1978.

[Llo84] John Wylie Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[McN67] Robert McNaughton. Parenthesis grammars. *Journal of the ACM*, 14:490–500, 1967.

[MP83] G. Marque-Pucheu. Rational set of trees and the algebraic semantics of logic programming. *Acta Informatica*, 20:249–260, 1983.

[Nis90] Tetsuro Nishino. Mathematical analysis of lexical-functional grammars - complexity, parsability, and learnability-. In *Proceedings of Seoul International Conference on Natural Language Processing*, 1990.

[Pag90] Giulia M. Pagallo. *Adaptative decision tree algorithms for learning from examples.* PhD thesis, Department of Computer and Information Sciences, University of California, Santa Cruz, 1990. Technical Report UCSC-CRL-90-27.

[Pit89] Leonard Pitt. Inductive inference, DFAs, and computational complexity. In *Proceedings of AII-89 Workshop on Analogical and Inductive Inference (Lecture Notes in Computer Science, 397)*, pages 18–44. Springer-Verlag, 1989.

[PV88] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35:965–984, 1988.

[PW80] Fernando C. N. Pereira and David H. D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[PW89] Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *Proceedings of 21st Annual ACM Symposium on Theory of Computing*. ACM, 1989.

[Qui86a] J. Ross Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning (Vol. 2)*, pages 149–166. Morgan Kaufmann, 1986.

[Qui86b] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[Sak88a] Yasubumi Sakakibara. An efficient learning of context-free grammars for bottom-up parsers. In *Proceedings of International Conference on Fifth Generation Computer Systems 1988 (FGCS '88)*, pages 447–454. Ohmsha, Ltd, 1988.

[Sak88b] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. In *Proceedings of 1st Workshop on Computational Learning Theory*, pages 330–344. Morgan Kaufmann, 1988.

[Sak89] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. Research Report 93, IIAS-SIS, FUJITSU LIMITED, 1989. To appear in *Information and Computation*.

[Sak90a] Yasubumi Sakakibara. An efficient robust algorithm for learning decision lists. Research Report 105, IIAS-SIS, FUJITSU LIMITED, 1990.

[Sak90b] Yasubumi Sakakibara. Inductive inference of logic programs based on algebraic semantics. *New Generation Computing*, 7:365–380, 1990.

[Sak90c] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76:223–242, 1990.

[Sak90d] Yasubumi Sakakibara. Occam algorithms for learning from noisy examples. In *Proceedings of 1st Workshop on Algorithmic Learning Theory (ALT'90)*, pages 193–208. Ohmsha, Ltd, 1990.

[Sak90e] Yasubumi Sakakibara. On learning smullyan's elementary formal systems: Towards an efficient learning method for context-sensitive languages. *Advances in Software Science and Technology*, 2:79–101, 1990.

[Sal73] Arto Salomaa. *Formal Languages.* Academic Press, Inc., 1973.

[Sch91] Robert Elias Schapire. *The design and analysis of efficient learning algorithms.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1991. Technical Report MIT/LCS/TR-493.

[Sha81] Ehud Y. Shapiro. Inductive inference of theories from facts. Research Report 192, Department of Computer Science, Yale University, 1981.

[Sha82] Ehud Y. Shapiro. *Algorithmic program debugging.* PhD thesis, Department of Computer Science, Yale University, 1982. Published by MIT Press, 1983.

[Shi90] Takeshi Shinohara. Inductive inference from positive data is powerful. In *Proceedings of 3rd Workshop on Computational Learning Theory*, pages 97–110. Morgan Kaufmann, 1990.

[Slo88] Robert Sloan. Types of noise in data for concept learning. In *Proceedings of 1st Workshop on Computational Learning Theory*, pages 91–96. Morgan Kaufmann, 1988.

[Slo89] Robert H. Sloan. *Computational learning theory: new models and algorithms.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1989. Technical Report MIT/LCS/TR-448.

[Smu61] Raymond M. Smullyan. *Theory of Formal Systems.* Princeton University Press, 1961.

[Tak88] Yuji Takada. Grammatical inference for even linear languages based on control sets. *Information Processing Letters*, 28:193–199, 1988.

[TSO91] Yuji Takada, Yasubumi Sakakibara, and Takeshi Ohtani. ACE: A syntax-directed editor customizable from examples and queries. Research Report IIAS-RR-91-3E, IIAS-SIS, FUJITSU LABORATORIES LTD., 1991.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–81, 1968.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Val85] Leslie G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of 9th International Joint Conference on Artificial Intelligence*, pages 560–566. Morgan Kaufmann, 1985.