Studies on the Learnability of Formal Languages via Queries

坂本, 比呂志
Graduate School of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.11501/3147889

出版情報：九州大学, 1998, 博士（理学）, 課程博士
バージョン：
権利関係：

Studies on the Learnability of Formal Languages via Queries

坂本 比呂志

Studies on the Learnability of Formal Languages via Queries

坂本　比呂志

ABSTRACT

Studies on the Learnability of Formal Languages via Queries

Hiroshi Sakamoto
Kyushu University
1998

The present thesis deals with the learnability of formal languages via *queries* based on Angluin's [6] learning protocol. Assuming a class of concepts and a finite or countable hypothesis space corresponding to it, for every unknown target concept, a learning algorithm is required to output a hypothesis that correctly describes the target using additional information provided by *membership queries* and *equivalence queries*.

For such learning problems, two *oracles* for membership and equivalence queries are assumed, i.e., for the membership query, one oracle exactly answers any question "Is this example a member of the target concept?" and for the equivalence query, the other exactly answers any question "Does this hypothesis correctly describe the target concept?". The equivalence oracle returns a *counterexample* for the hypothesis if it does not.

First, *language learning from good examples* is investigated. The concept class studied is a subclass of *context-free languages* because of their rich potential of application. On learning via queries, the capacity of an equivalence query may be too powerful and not reasonable since the equivalence problem for context-free languages is undecidable. On the other hand, it is also true that additional information sometimes makes language learning efficient. Thus, the aim of this research is to present a natural learning model by combining learning via queries with learning from good examples. Instead of equivalence queries, a method for providing good examples is applied to the class of *parenthesis languages* (cf. [44]). These good examples are called *characteristic strings*. Assuming the setting of membership queries and characteristic strings for the class of parenthesis grammars, it is shown that there exists a polynomial-time algorithm to learn every parenthesis language.

Second, the *learnability of finite-memory automata via queries* is studied. Learning the regular languages is one of the ultimate objectives learning theory. Although the learnability of the regular languages via queries w.r.t. the hypothesis space of deterministic finite automata [4], there are very few results for other hypothesis space,

e.g., regular expressions and nondeterministic finite automata. The aim of this research is to present a learning algorithm for regular languages using a new hypothesis space, i.e., the *finite-memory automata* (cf. [32]). A finite-memory automaton is assumed to use *registers* to store symbols, and the number of these registers is fixed *a priori*. Such an automaton is defined over an infinite alphabet, and for any finite alphabet $\mathcal{A}$, the language accepted by the automaton is equivalent to a regular language over the alphabet $\mathcal{A}$. Thus, we start with the natural question whether finite-memory automata represent regular languages than finite automata more compactly. This compactness is indirectly shown by the negative results for computational hardness of decision problems for finite-memory automata, and the learnability of the automata is expected to be difficult. Then, we investigate the learnability of a subclass of finite-memory automata referred to as *simple deterministic* finite-memory automata. Almost all computational problems are intractable even if restricted to the class of simple deterministic automata. Thus, the learnability of this simple deterministic class is also difficult. However, we show that there exists a learning algorithm for this class using membership and equivalence queries.

Finally, *language learning from incomplete data* is studied. There is another hierarchy of formal languages, the so-called *pattern languages* (cf. [1]). The aim of this study is to analyze the contribution of *negative examples* to the learnability of pattern languages. Several negative results concerning the feasibility of the *consistency* problem for pattern languages have been obtained previously (cf., e.g., [1, 46, 73]). Here, the consistency problem is defined as follows. Given any set of labeled examples, decide whether there exists a pattern generating all positive examples given and none of the negative examples. These results provide substantial evidence for the difficulty of learning the pattern languages consistently w.r.t. the hypothesis space of all patterns. However, there is an interesting case remaining open, i.e., the case of *one-variable pattern languages*. In this case, the consistency problem is neither known to be NP-complete nor to be in P. While these results support the difficulty of learning patterns from both positive and negative examples, there is an expectation for effective learnability of the class of *one-variable pattern languages*, and it is an interesting open question whether the consistency problem is decidable in time polynomial.

In order to approach a solution of this problem, *incomplete strings* are introduced. This notion is an application of the framework of *monotone extensions* introduced by Boros *et al.* [14] for the setting of learning Boolean functions. An incomplete string is assumed to contain unsettled symbols denoted by a *wild card* $\star$ which potentially

matches with every symbol. Thus, by fixing any finite alphabet Σ plus $\star$, the consistency problem is generalized informally as follows. Given two disjoint sets T and F of strings in $(\Sigma \cup \{\star\})^+$, an algorithm must decide whether there exists a one-variable pattern consistent with each strings in T and F with respect to several criteria of suitable settlements for these wild cards $\star$. The computational complexity of these problems is investigated, and it is concluded that incomplete strings make the consistency problem difficult, i.e., almost problems studied are shown to be NP-complete.

ACKNOWLEDGMENTS

This research was supported by a JSPS (Japan Society for the Promotion of Science) Research Fellowship for Young Scientists. The results in this thesis were or will be partially published in the proceedings of ALT'95, '97, and '98, the proceeding of MCU'98, the Bulletin of Informatics and Cybernetics, and in Theoretical Computer Science. I am thankful to all editors, program committees, the anonymous referees, and the publishers.

At ALT'95, the chairman of my session was Prof. Klaus P. Jantke. His smile relaxed me and my presentation was successfully completed. At ALT'97, I had the opportunity to meet Prof. Ming Li. He seriously listened to my talk in front of me. At MCU'98, the program committee chair, Prof. Maurice Margenstern, gave us a warm welcome. Prof. Markus Holzer was interested in our study and he bombarded us with questions one after another. We are deeply grateful to Prof. Volker Diekert for his careful proof reading of our submission to this colloquium and to the TCS special issue. At ALT'98, I got response to my study from Prof. Frank Stephan and his colleagues as soon as I came back to Japan. They solved an important open problem, and I am very glad to have their permission to include the new result into my publication.

I investigated a theme together with Daisuke Ikeda. He got married this month, and I shall wish them good luck. I appreciate the benefits from Miss Noriko Sugimoto, Mr Eiju Hirowatari, and other colleagues. My sincere gratitude is owing to the teaching staff of Department of Informatics, Kyushu University. I further give my gratitude to members of our monthly "severe seminar". This seminar is supported by Prof. Setsuo Arikawa, Hiroki Arimura, Hiroki Ishizaka, Ayumi Shinohara, Takeshi Shinohara, Masayuki Takeda, and Thomas Zeugmann.

This thesis was published with helpful and accurate comments from Prof. Fumihiro Matsuo. I want to express special thanks to my supervisor, Prof. Thomas Zeugmann, who had guided me for a long time. Finally, my deepest thanks go to my family.

October 31, 1998
Fukuoka

坂本比呂志
Hiroshi Sakamoto

Table of Notations

In this thesis, we will use several notations which may be not standard. The following table consists of these notations. Moreover, we provide another table listing computational problems as well as their abbreviations, and locations in this thesis.

Expression	Usage
$\log n$	$\log_2 n$
$\lfloor x \rfloor$; $\lceil x \rceil$	$\max\{n \mid n \leq x\}$; $\min\{n \mid n \geq x\}$
$f : A \to B$	a partial function from domain A to range B
small Greek letter	potentially any string, e.g., $\alpha, \beta, \ldots, \gamma$
A^*	the free monoid over the set A
$\|\|A\|\|$; $\|\alpha\|$	cardinality of set A; length of string α
$A \setminus B$ and $A \oplus B$	difference and symmetric difference of sets
$\mathbb{N}$	the set of all natural numbers
$\mathbb{B}^n$	the set of all n-ary Boolean vectors
$\Sigma = \{a, b, \ldots, c\}$	a finite alphabet
$\Omega = \{a_i \mid i \in \mathbb{N}\}$	an infinite alphabet
calligraphic face	a class of representations e.g., $\mathcal{P}$ and $\mathcal{P}_k$
small capital face	a computational problem, e.g., SAT and CVP
capital bold italic face	name of an algorithm, e.g., $\boldsymbol{A}$

Table of Computational Problems

We shall summarize representative decision problems in this thesis, where consistency problem is equal the following E, and problems for finite automata are omitted by analogy with finite-memory automata.

List of Figures

Contents

CHAPTER 1

Introduction

The present thesis studies learning based on Angluin's [6] framework of *learning via queries*. Let $\Sigma = \{a, b, \ldots, c\}$ be any finite *alphabet*. We write Σ^* to denote the free monoid over Σ. The underlying learning domain is Σ^*. The *concepts* to be learned are recursively enumerable subsets of Σ^*, i.e., any r.e. language in $\mathcal{R}(\Sigma^*)$ may be a target concept. Subclasses of $\mathcal{R}(\Sigma^*)$ are referred to as concept classes. Throughout this thesis, we shall mainly study indexable concept classes. A language class $\mathcal{L} \subseteq \mathcal{R}(\Sigma^*)$ is said to be indexable provided all $L \in \mathcal{L}$ are non-empty and there exists a recursive enumeration $(L_j)_{j \in \mathbb{N}}$ of all the languages in $\mathcal{L}$ as well as a recursive procedure f such that for all $j \in \mathbb{N}$ and all $s \in \Sigma^*$,

$$f(j, s) = \begin{cases} 1, & s \in L_j \\ 0, & \text{otherwise.} \end{cases}$$

Prominent examples of indexable classes are the *regular languages*, the *context-free languages*, and the *pattern languages*. The indices j can be thought of as suitable finite encodings or, synonymously, finite representations. For example, the regular languages can be represented by finite automata or regular expressions, while pushdown automata or context-free grammars may serve as representations for the context-free languages. Pattern languages are most usually represented by patterns. It should be considered that all these representation classes constitute themselves indexable classes in a normal way. Therefore, in the following, we shall always assume all language classes and all representations for them to be indexable.

Now, *learning* is a process of identifying a target language automatically by means of a finite representation for it. More formally, assuming a class $\mathcal{L}$ of languages and

a class $\mathcal{H}$ of representations corresponding to $\mathcal{L}$, a target $L \in \mathcal{L}$ is arbitrarily chosen, and information about L is provided to an algorithm as input.

The algorithm is said to *learn* $\mathcal{L}$ if for every target $L \in \mathcal{L}$, it stops and outputs a hypothesis $h \in \mathcal{H}$ which is a representation for L. Moreover, we are interested in *efficient learning*: the running time of the algorithm for every concept $L \in \mathcal{L}$ can be estimated by the size of a minimum representation h for L and by the size of input in binary. For a concept class, when we construct a learning algorithm satisfying the criterion of efficient learning, a problem arises naturally—*From what information does the algorithm learn?* Throughout this thesis, we mainly consider active learning. That is, the learner obtains information about the unknown target by asking *queries*. The motivation for a theoretical study of query learning goes back to system implementations that allow a computer to ask its user.

For instance, Sammut-Banerji's [60] *expert system* uses questions about specific examples as part of its strategy for efficient learning a target concept, and Shapiro's [62, 63, 64] *algorithmic debugging system* makes a variety of questions possible to a user to pinpoint errors in **Prolog** programs.

These intelligent systems stimulated the theoretical analysis of exact learning, and the framework of Angluin's [6] query learning model has been introduced to model the situation in which a *learner* can put queries to a *teacher* by oracle Turing machines. She studied the power of several types of queries, referred to as *membership*, *equivalence*, *subset*, *superset*, *disjointness*, and *exhaustiveness*. In particular, the membership and equivalence query attracted considerable attention during the last decade.

The learning protocol using membership and equivalence queries is referred to as *minimally adequate teacher*. Consider the problem of identifying a target language $L \in \mathcal{L}$ from a finite or countable hypothesis space $\mathcal{H} = h_0, h_1, \ldots$. Then, a learning algorithm has access to a fixed set of *oracles* that correctly answer the following questions.

Membership: Input is an element $x \in \Sigma^*$, and the output is *yes* if $x \in L$ and *no* otherwise.

Equivalence: Input is an index n in the hypothesis space, and the output is *yes* if $L_n = L$ and *no* otherwise. Moreover, if the answer is *no*, then an element

$x \in L_n \oplus L$, called a *counterexample*, is returned, where $L_n \oplus L$ is the symmetric difference of L_n and L.

For example, let $\mathcal{L}$ be the set of all regular languages $L \subseteq \Sigma^*$, and let the class of deterministic finite automata, DFAs, be assumed to be hypothesis space for the learning algorithm. The goal is to identify a correct DFA which accepts the target language L chosen by the teacher. For each $w \in \Sigma^*$, the membership oracle answers whether $w \in L$, and for each DFA M, the equivalence oracle answers whether $L = L(M)$. If not, then it returns a counterexample w for the hypothesis M such that $w \in L \oplus L(M)$.

The choice of a hypothesis space $\mathcal{H}$ plays an important role in the running time of a query learning algorithm $\boldsymbol{A}$ for a class $\mathcal{L}$ of languages. A class $\mathcal{L}$ is said to be *learnable in time polynomial* using the specified queries w.r.t. the hypothesis space $\mathcal{H}$, if for every target $L \in \mathcal{L}$, the total running time of $\boldsymbol{A}$ is bounded by a polynomial in n and m, where n is the size of a minimum $h \in \mathcal{H}$ for L and m is the length of a longest counterexample returned. The query model is powerful to learn various classes of formal languages. Angluin [4] showed that the class of regular languages is learnable in time polynomial in the parameters using equivalence and membership queries w.r.t. the hypothesis space of DFAs.

After her work, a number of researchers succeeded to expand her result, thereby still achieving polynomial-time learning algorithm, e.g., the learnability of the class of languages accepted by *one-counter automata* by Bermann and Roos [12], *even-liner grammars* by Takada [70], and *simple deterministic grammars* by Ishizaka [29]. These classes of formal languages are subclasses of the *context-free* languages which have attracted a great deal of attention because of the rich potential of application.

For example, it is well-known that all programming languages in **BNF** (Backus Naur Form) are mainly defined by context-free grammars, e.g., **C** and **PASCAL**. Furthermore, Shinohara [67] developed a data entry system for text data base. He investigated the learnability of *regular pattern languages* in the framework of Gold's [21] *identification in the limit*. Although different data bases usually have different formats for records, e.g., "Author", "Title", and "Year", his system effectively learns different types of these formats using the pattern inference techniques.

We can find other examples of wide application of context-free languages nearby. An HTML document on web-page is defined by a *parenthesis* grammar that is also context-free since a source file consists of structural text parenthesized by a pair of beginning tags and the corresponding ending tags, e.g., <html>*text*</html>. When considering the setting of equivalence queries, the answer to an equivalence query is carried by users in real application.

However, this setting would make the burden too heavy for users because we need enough information about the target beforehand to provide a correct answer to an equivalence query. Moreover, it seems that equivalence query is not reasonable according to hypothesis spaces, since the equivalence problem of two context-free grammars is computationally undecidable.

Since membership queries alone are too weak to achieve powerful learning algorithms (cf., e.g., [6, 8]), various authors have considered suitably chosen finite sets of strings as information given to the learner. Intuitively, all these sets can be thought of as sets of *good examples*. For instance, Angluin's *live-complete sets* [2], Ibarra-Jiang's *lexicographically first string* [28], and Oncina-García's *characteristic set* [49] for DFAs and Ibarra-Jiang's *shortest strings* [28], Marron-Ko's *positive initial sample* [43], and Marron's *single positive example* [42] for pattern languages.

Note that humans also learn mainly from good examples, or at least much more efficiently. Therefore, we shall continue along this line. The motivation of this study to provide good examples comes from the class of *parenthesis languages* (cf. [44]). A parenthesis language is a context-free language possessing a grammar in which each application of a production rule introduces a unique pair of parentheses, delimiting the scope of that production. Parenthesis languages are nontrivial since only one kind of parenthesis is used, and they are one of rich classes for which equivalence problems are decidable (cf. [44, 36]).

However, it is not known whether the equivalence problem for given two parenthesis grammars is in P. Thus, instead of equivalence query, we propose a setting to provide polynomially many good examples to learning algorithm, and present a learning model by combining membership query with *good examples*. Our idea comes from the fact that each parenthesized string preserves the structure of its derivation, in other words, a pair of parentheses in the string corresponds to an application of a production rule.

A good example of a parenthesis grammar G is, intuitively, a string derived by as many production rules of G as possible. Let L be a target parenthesis language defined by a grammar G. The learning algorithm takes good examples of G, and outputs a grammar G' such that $L(G) = L(G')$ using only membership queries. The total running time is bounded by a polynomial in the number of productions of the target as well as in the length of a longest string provided to the algorithm. From the resulting correctness of the algorithm, we conclude that good examples contribute to learning parenthesis grammars without equivalence queries.

As we shall saw in the above paragraph, when a target parenthesis language L is selected, good examples are decided by a grammar G such that $L = L(G)$. Thus, the result depends on the hypothesis space of parenthesis grammars. This phenomenon is also found in other results in language learning, for example, when dealing with the class of regular languages. It is well-known that regular languages are learnable in time polynomial w.r.t. the hypotheses of DFAs, however, few results were obtained for other hypothesis space (cf., e.g., [4, 15, 75]). That is, both successful learning and efficiency of learning may depend on the hypothesis space chosen. This study is motivated by also the above problem.

There is no polynomial-time algorithm to decide whether two regular expressions are inequivalent* even if only $\cup$, $\cdot$, and ^{2} are allowed, where e^2 is a regular expression equal to $e \cdot e$ for every regular expression e. Similarly, inequivalence problem for finite automata is PSPACE-complete (cf., e.g., [69, 23]). These equivalence oracles are too powerful, and it is not reasonable to assume such oracles for these hypothesis spaces. Then, we consider another hypothesis space consisting of *finite-memory automata*, FMAs, for regular languages (cf. [32]).

Compared with finite automata, the difference of definition is that an FMA can use k registers to memorize k symbols. It is possible to replace the content of a register by any input symbol. The transition is defined by only the state and the address of a register which contains the same symbol to input. Thus, FMAs are released from specification of alphabets of its definition. That is, an FMA potentially represents a language over an infinite alphabet.

Assuming the hypothesis space of all FMAs, the following question arises naturally:

*This problem is NEXP-complete [69], where NP $\neq$ NEXP $= \cup_{k>0}$NTIME$[2^{n^k}]$.

Can finite-memory automata represent regular languages more compactly than finite automata? This question is partially solved by making a comparison of the complexity of several decision problems between finite-memory automata and finite automata. The complexity of studied decision problems is also interesting in its own right, and has been remained open in [32]. However, unfortunately, the resulting intractability of almost all decision problems supports the hardness of the learnability of the full class of finite-memory automata.

Therefore, we introduce the class of *simple* FMAs and investigate the learnability of simple DFMAs via membership and equivalence queries, where DFMA denotes a deterministic FMA (cf. [32]), and an FMA is simple if its all registers are initially empty. Although the studied decision problems remain intractable for the restricted class, we construct an algorithm to learn each target language. This algorithm is based on Angluin's [4] observation table technique. The hypothesis space assumed is all simple DFMAs, and it is allowed to use membership and equivalence queries for each target language over an infinite alphabet. For every target language L accepted by a simple DFMA, our learning algorithm terminates and outputs a correct hypothesis A such that $L = L(A)$ over the infinite alphabet.

We have mentioned learning subclasses of languages in the Chomsky hierarchy at the points: the confidence of a hypothesis proposed by an algorithm and the convergence of a hypotheses into a correct one. The former is solved by equivalence oracles for these hypothesis space, but the latter is not because these are independent problems each other, for instance, there is other interesting hierarchy of the classes of *pattern languages* (cf. [1]).

Let Σ be any finite alphabet, and let $X = \{x_1, x_2, \ldots, \}$ such that $\Sigma \cap X = \emptyset$. A pattern is a non-null string over $\Sigma \cup X$. Let f be a nonerasing homomorphism from patterns to patterns. If $f(a) = a$ for all Σ, then f is called a *substitution*. The language of a pattern π is the set $L(\pi) = \{w \in \Sigma^+ \mid w = f(\pi),\ f \text{ is a substitution}\}$, where $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ for the null string ε. The class of all patterns is denoted by $\mathcal{P}$.

For the class $\mathcal{P}$, there is no polynomial-time learning algorithm even though membership and equivalence queries are assumed (cf. [6]). This is because, in a worst case, a learner must receive exponentially many counterexamples to achieve a correct hypothesis when a target pattern language is a singleton. Moreover, learning pattern

languages is still difficult even if we assume a target is not singleton. This difficulty is related to the computational problem: assuming the hypothesis space $\mathcal{P}$, and given finite sets T and F of strings, decide whether there is a pattern language $L(\pi)$ such that $T \subseteq L(\pi)$ and $F \cap L(\pi) = \emptyset$. The samples T and F are called *positive* and *negative*, respectively. This problem is referred to as the *consistency problem for* $\mathcal{P}$.

The computability and learnability of patterns have been widely investigated and several negative results were shown (cf., e.g., [1, 31, 46, 61, 73]). From these results it follows that learning pattern languages consistently is very hard, nevertheless, not all possibilities disappeared. An interesting question that remains open is the consistency problem for the *one-variable patterns*. The class of one-variable patterns, denoted by $\mathcal{P}_1$, is the set of all patterns over $\Sigma \cup X$ such that $||X|| \leq 1$. It is neither known whether this problem is in P nor NP-complete. In order to shrink the gap of our knowledge concerning the complexity of the consistency problem, we shall relax this problem step by step, and we analyze the complexity of each of the resulting problems.

The idea of relaxation of this problem comes from the motivation of *incomplete data* given in Boros *et al.* [14], which is helpful when looking at *monotone extensions* from the view point of how noisy data may influence the complexity of learning. Since real world data may be noisy, allowing strings to contain indefinite values can be modeled by introducing a wild card $\star$ as a placeholder. Given samples T and F of strings containing $\star$, we provide the following interpretations for these indefinite values w.r.t. the consistency problem for $\mathcal{P}_1$. The ordinary consistency problem for $\mathcal{P}_1$ is equivalent to the case that any given string contains no wild card.

If a string w contains at least one $\star$, then we consider an assignment for w such that it replaces each $\star$ by a constant and does not replace any constant. Then, the first problem is whether there exist a pattern $\pi \in \mathcal{P}_1$ and a suitable assignment for each string in $T \cup F$ such that π is consistent with all the assigned strings. The second problem is whether there exists a pattern $\pi \in \mathcal{P}_1$ such that π is consistent with all the assigned strings whatever assignment for $T \cup F$ we choose.

In order to study the above problems in detail, we also consider the restricted version for them such that the positive sample consists of only constant strings. Consequently, we conclude that the first problem is equivalent to the ordinary consistency problem w.r.t. log-space reductions if the positive sample is restricted. Moreover, all

other problems are intractable. In particular, the proof of the NP-completeness of the first problem was provided in [68]. This problem was an open question of this study, and the author had expected that it is also equivalent to the consistency problem.

We have discussed the outline of this thesis, now, explain the technical part for each theme. The following chapter consists of the bases on formal languages, computational complexity, and learning theory, which are necessary for our discussions. Recent results of language learning via queries are also presented in this chapter.

In Chapter 3, we deal with the announced learning parenthesis languages using membership and good examples. Let G be a context-free grammar over a set Σ of terminals and a set $N = N' \cup \{(,)\}$ of nonterminals. Then, the grammar G is said to be *parenthesized* if each production rule $A \to \alpha$ of G satisfies that $A \in N'$, $\alpha = (\beta)$ and $\beta \in (\Sigma \cup N')^*$. On a parenthesis grammar, for each usage of a production rule, exactly one pair of '(' and ')' is derived in the leftmost and rightmost positions of a string. Hence, for any parenthesis grammar G, the language $L(G)$ becomes to be *unambiguous*[†], i.e., for every $w \in L(G)$, there exists exactly one leftmost derivation of G for w. This point is one of the critical parts of our study.

The unambiguity of a target grammar $G = (N, \Sigma, P, S)$ can be considered that a string w derived from G preserves the structure of its derivation tree T. The task of the learning algorithm is to decide all labels of internal nodes of T. A string w is said to be a *characteristic string* of G if all production rules of G are used to derive the w, but not all parenthesis grammars have such a string.

Thus, we refine the notion of good examples for G by partitioning the grammar G. A grammar $G' = (N', \Sigma', P', S)$ is said to be a *sub-grammar* of G if $N' \subseteq N$, $\Sigma' \subseteq \Sigma$, and $P' \subseteq P$. We prove that there are polynomially many sub-grammars $G_1, \ldots, G_k$ for any context-free (of course, parenthesis) grammar G such that G_i derives at least one characteristic string w_i for all $i = 1, \ldots, k$. Thus, we assume that these strings $w_1, \ldots, w_k$ are given to the learning algorithm, and show that every parenthesis grammar G is learnable using the characteristic examples and membership queries in time polynomial in n and m, where $n = ||P||$ and $m = \max\{|w_i| \mid i = 1, \ldots, k\}$.

In Chapter 4, we study the learnability of simple DFMAs via membership and

[†]Although a context-free grammar is inherently ambiguous, Sakakibara [53] avoided this difficulty using special strings represented by trees for learning context-free grammars.

equivalence queries. Let $\mathbb{N}$ be the set of all natural numbers. Then, an FMA defines a language over the infinite alphabet $\Omega = \{a_i \mid i \in \mathbb{N}\}$. Let Σ be any finite subset of Ω, and let A be an FMA. The language $L(A) \cap \Sigma^*$ accepted by A is a regular language, i.e., the class of languages of FMAs over the finite alphabet Σ is equivalent to that of finite automata over Σ.

The deterministic class DFMAs is a subclass of FMAs. One easily shows that this class is closed under complement, but not closed under union and intersection (cf. [32]). Moreover, there exists a language L accepted by an FMA but the language $\Omega^* \setminus L$ is not. Thus, it is straightforward that the deterministic class is properly included in the general class (cf. [32]). Several interesting closure properties of both classes of finite-memory automata were investigated in [32], however, there is a lack of investigation on decision problems for them.

As it is well-known, the decision problems for finite automata referred to as *membership*, *non-emptiness*, and *inequivalence* are complete for the classes NLOG, NLOG, and PSPACE, respectively, and the corresponding problems for the deterministic finite automata are complete for DLOG, NLOG, and NLOG, respectively. On the other hand, we prove that the membership and non-emptiness problems for the class of FMAs are both NP-complete. Furthermore, for the class of DFMAs, the membership problem is P-complete and the non-emptiness problem is NP-complete.

From these results, we observe that, in the polynomial hierarchy, the complexity of the studied problems for FMAs shape a counterpart of the corresponding problems for finite automata. Thus, the inequivalence problem for the deterministic and nondeterministic classes are expected to be NP-complete and NEXP-complete, respectively. While the inequivalence problem for the deterministic class is in PSPACE and its NP-hardness is proved, the problem whether it is in NP remains open.

We next turn our attention to the learnability of a subclass of FMAs via membership and equivalence queries. We introduce the class of simple DFMAs, and for every target, we assume membership and equivalence queries to the learner. Even though the alphabet Ω is infinite, we conclude that the setting is reasonable since, as we have mentioned above, both related decision problems for the class are decidable.

When a counterexample is returned, our learning algorithm constructs a finite automaton M based on the notion of *observation table* [4], and in the next stage, this

algorithm successively translates M into a simple DFMA A such that $L(A) \cap \Sigma^* = L(M)$, where Σ is a set of all symbols in counterexamples returned so far. Consequently, we show that the class of languages accepted by simple DFMAs is learnable using membership and equivalence queries w.r.t. the hypothesis space.

In Chapter 5, the difficulty of the consistency problem for one-variable patterns is studied. It is known that consistency problem is very hard for almost all subclasses of the class $\mathcal{P}$ (cf., e.g., [1, 46, 73]), however, no one has been proved its intractability within $\mathcal{P}_1$ yet. Thus, we analyze the consistency problem as well as its variants for the class $\mathcal{P}_1$ w.r.t. *incomplete examples* defined as follows.

An incomplete example is any string over $\Sigma \cup \{\star\}$, where the $\star$ potentially matches with every symbols. We assume the set of all functions $f : (\Sigma \cup \{\star\})^+ \to \Sigma^+$ such that it maps every $\star$ in a string to a constant in Σ and maps any constant in the string to itself. Then, given finite positive and negative samples $T, F \subseteq (\Sigma \cup \{\star\})^+$, an algorithm must decide whether there exists a $\pi \in \mathcal{P}_1$ and a function f defined above such that π is consistent with $f(T)$ and $f(F)$, where $f(T) = \{f(w) \mid w \in T\}$ and $f(F)$ is analogous. The studied problems are defined by the following criteria:

1. There exists a one-variable pattern π consistent with the given T and F provided $T, F \subset \Sigma^+$. This is the ordinary consistency problem referred to as *extension*.

2. There exists a one-variable pattern π and a suitable function f such that π is consistent with $f(T)$ and $f(F)$. This problem is referred to as *consistent extension*.

3. There exists a one-variable pattern π consistent with $f(T)$ and $f(F)$ for all f. This problem is referred to as *robust extension*.

Moreover, the *restricted* consistent extension and robust extension are also studied, where a restricted problem is that any string in T contains no $\star$. We show that the extension and restricted consistent extension are computationally equivalent with respect to log-space reductions. The robust extension is NP-complete even if an alphabet consists of only two symbols. Additionally, the consistent extension is also NP-complete[‡], thus, we arrive at the conclusion that almost all problems are intractable.

[‡]The NP-completeness was proved by Stephan [68], personal communication.

Finally, in Chapter 6, we mainly discuss several open questions not solved in this thesis, i.e., the NP-completeness of the equivalence problem for the class of DFMAs, and the computability of the ordinary consistency problem for the class $\mathcal{P}_1$.

CHAPTER 2

Preliminaries

This chapter contains basic definitions necessary to make this thesis self-contained. We assume familiarity with formal language theory (cf., e.g., [26] and [27]), computational complexity theory (cf., e.g., [23] and [50]), and algorithmic learning theory (cf., e.g., [35] and [48]). For our framework, in the first section of this chapter, we begin with formal languages including context-free grammars, patterns, and other convenient notions. In the next section, we deal with decision problems. Typical complete problems are specified as well as the notions of reduction and completeness. Finally, we formalize our learning model using queries and summarize previously known results on learning formal languages.

2.1 Basic Notations and Definitions

A *graph* is denoted by $G = (N, E)$, where N is a finite set of *nodes* and $E \subseteq N \times N$ is a set of *edges*. A *path* in G is a sequence of nodes $n_1, n_2, \cdots, n_k$ such that there is an edge $\{n_i, n_{i+1}\}$ for each $1 \leq i \leq k-1$. The number $k-1$ is called the *length* of the path. A *directed graph*, also denoted by $G = (N, E)$, consists of a finite set of nodes N and a set of ordered pairs of nodes E called *arcs*. If $(v, w) \in E$, we refer to (v, w) as to an arc from v to w and denote it sometimes by $v \to w$. A *path* in a directed graph is a sequence of nodes $n_1, n_2, \cdots, n_k$ such that $n_i \to n_{i+1}$ is an arc for each $1 \leq i \leq k-1$. We say the path is *from* n_1 to n_k. If $v \to w$ is an arc, then we call that v is a *predecessor* of w and w is a *successor* of v.

A *tree* is a directed graph that satisfies the following conditions. There is one node, called the *root*, that has no predecessor and from which there is a path to every node.

Each node other than the root has exactly one predecessor. The successors of each node are ordered from the left. We continue with some special terminology for trees. A successor of a node is called its *child*, and the predecessor is called its *parent*. If there is a path from node n_i to node n_j, then n_i is said to be an *ancestor* of n_j, and n_j is said to be a *descendant* of n_i, where each node is an ancestor and a descendant of itself. A node with no child is called a *leaf*, and all other nodes are called *internal* nodes except the root.

A *binary relation* is a set of pairs. The first component of each pair is chosen from a set called the *domain*, and the second component of each pair is chosen from a (possibly different) set called the *range*. In particular we are interested in relations in which the domain and range are the same set S. In this case we say the relation is *on* S. If R is a relation and $(a,b) \in R$, then we also write aRb. We say a relation R on S is *reflexive* if aRa for all $a \in S$, *transitive* if aRb and bRc imply aRc, and *symmetric* if aRb implies bRa for all $a,b,c \in S$.

A reflexive, symmetric and transitive relation is said to be an *equivalence relation*. An equivalence relation R on S partitions S into disjoint nonempty sets called *equivalence classes*. That is, $S = S_1 \cup S_2 \cup \cdots$ such that for each i and j with $i \neq j$, $S_i \cap S_j = \emptyset$, aRb is true for each $a,b \in S_i$, and aRb is false for each $a \in S_i$ and $b \notin S_j$.

The *transitive closure* of a relation R, denoted by R^+, is defined recursively by the following conditions. (1) If $(a,b) \in R$, then $(a,b) \in R^+$. (2) If $(a,b) \in R^+$ and $(b,c) \in R$, then $(a,c) \in R^+$. (3) Nothing is in R^+ unless it follows from the condition (1) and (2). Furthermore the *reflexive, transitive closure* of R, denoted by R^*, is the set $R^+ \cup \{(a,a) | a \in S\}$.

2.2 Formal Languages

An *alphabet* $\Sigma = \{a_0, a_1, \ldots\}$ is a set of partial ordered symbols (i.e., for all $a_i, a_j \in \Sigma$, $a_i < a_j$ iff $i < j$). The expression Σ^* denotes the free monoid over Σ and we set $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ (i.e., the set of all non-null strings over Σ), where ε denotes the empty string. The *length* of a string w and the *cardinality* of a set S are denoted by $|w|$ and $||S||$, respectively. The *lexicographical ordering relation* $<$ on Σ^* is defined as follows. Let $x = x_1 \cdots x_s$ and $y = y_1 \cdots x_t$ be strings in Σ^*. Then, $x < y$ if (a) $s < t$ or (b)

$s = t$ and there exists an $1 \leq r \leq s$ such that $x_i = y_i$ for all $i = 1, \ldots, r-1$, and $x_r < y_r$. A *language* over Σ is any subset of Σ^* and a *class of languages* over Σ is a collection of languages containing at least one nonempty language over Σ.

DEFINITION 2.2.1. A *context-free grammar* CFG is 4-tuple $G = (N, \Sigma, P, S)$, where N and Σ are finite sets of symbols such that $N \cap \Sigma = \emptyset$, $S \in N$, and P is a finite subset of $N \times (N \cup \Sigma)^*$. Elements of N and Σ are called *nonterminals* and *terminals*, respectively. The S is called the *start symbol*. Any element of P is called a *production rule* denoted by $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in (N \cup \Sigma)^*$.

Let $\alpha', \beta' \in (N \cup \Sigma)^*$. We say that α' *directly derives* β', denoted by $\alpha' \Rightarrow \beta'$, if there exist $\alpha_1, \alpha_2, \alpha, \beta \in (N \cup \Sigma)^*$ such that $\alpha' = \alpha_1 \alpha \alpha_2$, $\beta' = \alpha_1 \beta \alpha_2$ and $\alpha \rightarrow \beta \in P$. If there exist $\alpha_1, \alpha_2, \cdots, \alpha_n \in (N \cup \Sigma)^*$ such that α_i directly derives α_{i+1} for all $i = 1, \ldots, m$, then we say that α_1 *derives* α_m and this is denoted by $\alpha_1 \Rightarrow^* \alpha_m$. That is, $\Rightarrow^*$ is the transitive closure of $\Rightarrow$ on $(N \cup \Sigma)^*$.

A sequence like the above is called a *derivation*. The set of *sentential forms* of a CFG $G = (N, \Sigma, P, S)$, denoted by $S(G)$, is the set $\{\alpha \in (N \cup \Sigma)^* \mid S \Rightarrow^* \alpha\}$. Thus, the *language generated by* G, denoted by $L(G)$, is the set $S(G) \cap \Sigma^*$. A language generated by a CFG is referred to as a *context-free language* denoted by CFL. Two CFGs G_1 and G_2 are said to be *equivalent* if $L(G_1) = L(G_2)$.

A *derivation tree* T of a grammar $G = (N, \Sigma, P, S)$ is a tree such that each internal node of T is labeled with an element of N, each leaf of T is labeled with an element of Σ and, for each internal node labeled with $A \in N$, there exists a production rule $A \rightarrow \alpha \in P$, where α is the concatenation of the labels of its children in left-to-right order.

We can characterize sufficiently long strings in a CFL L by the following, referred to as the *pumping lemma*. This lemma is useful to obtain several results in Chapter 3.

LEMMA 2.2.1. (Harrison [26]). Let L be a CFL such that L is not finite. There exists a constant n such that if $z \in L$ and $|z| \geq n$, then $z = uxvyw$ satisfies (1) $|xy| \geq 1$, (2) $|xvy| \leq n$, and (3) $ux^ivy^iw \in L$ for all $i \geq 1$.

Every CFG $G = (N, \Sigma, P, S)$ has several *normal forms* equivalent to G as follows. A nonterminal $A \in N$ is said to be *useless* if either S derives no sentential form

containing A, or A derives no terminal string. Nonterminals $A, B \in N$ are said to be *equivalent* if for every sentential form $w \in S(G)$, A derives w iff B derives w. A production rule $A \to B \in P$ is said to be a *chain rule* if $A, B \in N$.

A CFG G is said to be *reduced* if no useless nonterminal, no two equivalent nonterminals, and no chain rule are defined in G. It is well known that every CFL L is generated by a reduced CFG G and we can effectively compute such a G from arbitrary CFG G' such that $L = L(G')$. Thus, in this study, we assume that a CFG always denotes a "reduced CFG".

A production rule of the form $A \to \varepsilon$ is said to be an *ε-production*. A CFG G is said to be *ε-free* if G has no ε-production. A CFL L is also called ε-free if $\varepsilon \notin L$. Furthermore, a CFG $G = (N, \Sigma, P, S)$ is said to be *invertible* if $A \to \alpha, B \to \beta \in P$ implies $A = B$. Invertible grammar is one of normal forms of context-free grammars.

A CFG $G = (N, \Sigma, P, S)$ is said to be in *Greibach normal form* if each production rule in P is of the form $A \to a\alpha$, where $a \in \Sigma$ and $\alpha \in N^*$. Moreover, G is said to be in *m-standard form* if G is Greibach normal form and, for each $A \to a\alpha \in P$, $|\alpha| \leq m$.

THEOREM 2.2.1. (Harrison [26]). Every CFL L is generated by an invertible CFG G such that if L is ε-free, then so is G. Moreover, every CFL $L \setminus \{\varepsilon\}$ is generated by a CFG in Greibach normal form.

A subclass of CFG is obtained by restricting forms of production rules. A CFG $G = (N, \Sigma, P, S)$ is said to be *right linear grammar* if each production rule in P is of the form $A \to \alpha B$ or $A \to \alpha$, where $A, B \in N$, and symmetrically, a *left linear grammar* is defined. A right (or left) linear grammar G is said to be a *regular grammar* and the language of $L(G)$ is said to be a *regular language*. The class of regular languages is an interesting subclass of CFLs for our study. A regular language can be alternatively defined by the following deterministic sequential machine.

DEFINITION 2.2.2. A *deterministic finite automaton*, denoted by DFA, is 5-tuple $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a finite set of *states*, Σ is a finite *alphabet*, δ is a *transition function*: $Q \times \Sigma \to Q$ such that for each $p \in Q$ and $a \in \Sigma$, exactly one $q \in Q$ satisfies $\delta(p, a) = q$, $q_0 \in Q$ is the *start state*, and $F \subseteq Q$ is a set of *final states*.

The extension of δ to handle input string $w \in \Sigma^*$ is the reflexive, transitive closure of δ, denoted by δ^*, such that for each $p \in Q$, $\delta(p, \varepsilon) = p$ and for all $p \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$, $\delta^*(p, wa) = \delta(\delta^*(p, w), a)$. For simplicity, we denote δ^* by just δ. Thus, the *language accepted by* A, denoted by $L(A)$, is the set $\{w \in \Sigma^* \mid \delta(w, q_0) \in F\}$.

Two DFAs A and B are said to be *equivalent* if $L(A) = L(B)$. A DFA A is said to be *minimum* if for any DFA B such that $L(A) = L(B)$, the number of states of A is less than or equal to that of B. Moreover, a minimum DFA is referred to as *canonical* if it is lexicographically first.

DEFINITION 2.2.3. (Angluin [1]). Let Σ be a finite alphabet of *constant* and let $X = \{x_1, x_2, \ldots\}$ be a recursively enumerable set of *variables*, where $\Sigma \cap X = \emptyset$. Any finite string in $(\Sigma \cup X)^+$ is said to be a *pattern*.

Let f be a nonerasing homomorphism from patterns to patterns over $\Sigma \cup X$. If $f(a) = a$ for all $a \in \Sigma$, then f is called a *substitution*. We may use the notation $[s_1/x_1, \ldots, s_k/x_k]$ for the substitution which maps each variable x_i to the string s_i $(i = 1, \ldots, k)$ and maps any other symbol to itself. Thus, for a pattern π containing variables $x_1, \ldots, x_k$, the expression $\pi[s_1/x_1, \ldots, s_k/x_k]$ denotes the string obtained by replacing x_i by s_i for all $i = 1, \ldots, k$. The *language of a pattern* π, denoted by $L(\pi)$, is the set of all $w \in \Sigma^+$ such that there exists a substitution $f : (\Sigma \cup X) \to \Sigma^+$ such that $w = f(\pi)$.

A pattern π is said to be a *k-variable pattern* provided the π contains exactly k different variables for $k \geq 0$. In particular, if $k = 0$, then the π is said to be a *proper pattern*. Moreover, a pattern π is said to be a *regular pattern* provided the π contains any variable in X at most one time.

The class of k-variable patterns is denoted by $\mathcal{P}_k$, the class of regular patterns is denoted by $\mathcal{P}_R$, and analogously, we denote the class of all patterns by $\mathcal{P} = \cup_{k \geq 0} \mathcal{P}_k$.

THEOREM 2.2.2. (Angluin [1]). The class of all pattern languages is incomparable with the class of regular languages and with the class of context-free languages. The class of all pattern languages is closed under concatenation and reversal, but not closed under union, complement, intersection, Kleene plus, homomorphism, and inverse homomorphism.

A finite set S of Σ^+ is referred to as a *sample*. A pattern π is said to be *descriptive* of S iff $S \subseteq L(\pi)$ and for any pattern π' such that $S \subseteq L(\pi')$, $L(\pi')$ is not a proper subset of $L(\pi)$. Therefore, we shall consider the problem of finding a descriptive pattern.

PROBLEM 2.2.1. Given a sample S, find a pattern which is descriptive of S.

Angluin [1] also studied the problem of finding descriptive patterns and she showed that there is an algorithm which, given a sample $S \subset \Sigma^+$ as input, outputs a pattern $\pi \in (\Sigma \cup X)^+$ which is descriptive of S. In particular, she proposed an effective algorithm for this problem in the special case of the class of one-variable patterns as follows.

THEOREM 2.2.3. (Angluin [1]). There exists an algorithm which, given a sample $S \subset \Sigma^+$, outputs a one-variable pattern that is descriptive of S within $\mathcal{P}_1$ in $O(n^4 \log n)$ time, where $n = \sum_{s \in S} |s|$.

This result has been improved to $O(n^2 \log n)$ time by Erlebach *et al.* (cf. [18]). For the class $\mathcal{P}_R$ of regular patterns, Shinohara [65] provided a polynomial-time algorithm for the problem of finding a descriptive pattern within $\mathcal{P}_R$.

THEOREM 2.2.4. (Shinohara [65]). There exists an algorithm which, given a sample $S \subset \Sigma^+$, outputs a regular pattern that is descriptive of S within $\mathcal{P}_R$ in $O(m^2 n)$ time, where $m = \max\{|w| \mid w \in S\}$ and $n = ||S||$.

Shinohara [66] showed that a descriptive pattern is polynomially computable within $\mathcal{P}_R$ with respect to any (possibly erasing) substitution. Since for every sample S and $x_i \in X$, $S \subseteq L(x_i)$, the problem of finding descriptive pattern must have at least one solution. On the other hand, let us take a pair of samples, denoted by (T, F), such that $T, F \subset$ and $T \cap F = \emptyset$. Elements of the sample T are called *positive examples* and elements of F are called *negative examples*. A pattern π is said to be *consistent with* (T, F) if $T \subseteq L(\pi)$ and $F \cap L(\pi) = \emptyset$.

PROBLEM 2.2.2. Given a pair (T, F) of samples, decide whether there exists a pattern that is consistent with (T, F). This problem is referred to as the *consistency problem*.

The consistency problem is a *decision problem*. Unfortunately, there is no known polynomial-time algorithm for the consistency problem even within $\mathcal{P}_1$. Moreover, we suspect that this problem is *intractable*, that is, the consistency problem within $\mathcal{P}_1$ can be regarded as a most difficult problem of NP and there is no polynomial-time algorithm for this problem unless P = NP. For mathematical discussion of intractable problems, in the next section, we shall introduce *reductions* and *completeness* for complexity classes, and discuss the difficulty of several decision problems with respect to the criterion.

2.3 Decision Problems

Let us take a graph $G = (V, E)$. Many computational problems are connected with graphs. The most basic problem on graphs is called the *reachability* problem: Given a graph G and two nodes $m, n \in V$, is there a path from m to n? Like this problem, an interesting problem has an infinite set of possible instances. Each instance is a mathematical object (in this case, a graph and two nodes), of which we ask a question and expect an answer. Note that the reachability problem asks a question that requires either "yes" or "no". In complexity theory, we usually find it conveniently unifying and simplifying to consider only these problems, instead of problems requiring all sorts of different answers. Such problems are called *decision problems*.

The elements of a recursively enumerable set $X = \{x_1, x_2, \ldots\}$ are called *Boolean variables*. Boolean variables take the two values '1' or '0'. We combine these variables using *Boolean connectives* such as $\vee$ (*logical or*), $\wedge$ (*logical and*) and $\neg$ (*logical not*).

Definition 2.3.1. A *Boolean expression* is one of (1) a Boolean variable and (2) an expression of the form $\neg\phi$, $\phi_1 \vee \phi_2$, or $\phi_1 \wedge \phi_2$, where ϕ, ϕ_1, and ϕ_2 are Boolean expressions. In particular, a Boolean expression in case $\neg\phi$, $\phi_1 \vee \phi_2$, and $\phi_1 \wedge \phi_2$ are respectively called the *negation* of ϕ, the *disjunction* of ϕ_1 and ϕ_2, and the *conjunction* of ϕ_1 and ϕ_2. An expression of the form x_i or $\neg x_i$ is called a *literal* of the variable x_i.

A *truth assignment* T is a mapping from a finite set X' of Boolean variables to the set of *truth values* $\{0, 1\}$. Let ϕ be a Boolean expression and $X(\phi)$ denote the set of Boolean variables appearing in ϕ. Then, we call T is *appropriate* to ϕ if $X(\phi) \subseteq X'$.

We next define what it means for T to satisfy ϕ, denoted by $T \models \phi$. If ϕ is a variable $x_i \in X(\phi)$, then $T \models \phi$ if $T(x_i) = 1$. If $\phi = \neg\phi_1$, then $T \models \phi$ if $T \not\models \phi_1$. If $\phi = \phi_1 \vee \phi_2$, then $T \models \phi$ if $T \models \phi_1$ or $T \models \phi_2$. Finally, if $\phi = \phi_1 \wedge \phi_2$, then $T \models \phi$ if both $T \models \phi_1$ and $T \models \phi_2$.

We say two expressions ϕ_1 and ϕ_2 are *equivalent* if for each truth assignment T appropriate to both of them, $T \models \phi_1$ iff $T \models \phi_2$. Every Boolean expression can be rewritten into an equivalent one in a convenient specialized style as follows.

DEFINITION 2.3.2. A Boolean expression ϕ is said to be in *conjunctive normal form* if $\phi = \bigwedge_{i=1}^{n} C_i$, where $n \geq 1$, and each of the C_js is in the disjunction of one or more literals. The C_js are called the *clauses* of the ϕ. Symmetrically, an expression ϕ is said to be in *disjunctive normal form* if $\phi = \bigvee_{i=1}^{n} D_i$, where $n \geq 1$, and each of the D_js is the conjunction of one or more literals. The D_js are called the *terms* of the ϕ.

We say that a Boolean expression ϕ is *satisfiable* if there exists a truth assignment T appropriate to it such that $T \models \phi$. Especially, a ϕ is said to be *tautology* if $T \models \phi$ for all T appropriate to it. Satisfiability is an important property of Boolean expressions, so we shall consider the following decision problem.

PROBLEM 2.3.1. Given a Boolean expression ϕ in conjunctive normal form, decide whether ϕ is satisfiable. This problem is referred to as the *satisfiability problem* and is denoted by SAT.

This problem is one of the most fundamental decision problems. It is of interest to note that SAT can be solved by a deterministic algorithm that tries all possible combinations of truth values for the variables appearing in the expression. Besides, this problem can be solved by a nondeterministic polynomial-time algorithm that guesses a truth assignment and checks that it indeed satisfies all clauses. Hence, this problem is in NP, but presently, we do not know whether it is in P.

An *n-ary Boolean function* is a function $f : \{0,1\}^n \to \{0,1\}$. A Boolean expression ϕ with variables $x_1, \ldots, x_n$ expresses the n-ary Boolean function f if, for each n-tuple of truth values $t = (t_1, \ldots, t_n)$, $f(t) = 1$ if $T \models \phi$, and $f(t) = 0$ if $T \not\models \phi$, where $T(x_i) = t_i$ for $i = 1, \ldots, n$. Thus, every Boolean expression expresses some Boolean

function and the converse is also true. On the other hand, there is a potentially more economical way than expressions for representing Boolean functions as follows.

DEFINITION 2.3.3. A *Boolean circuit* is a directed acycle graph $C = (V, E)$, where the nodes in $V = \{1, 2, \ldots, n\}$ are called *gates* of C and each edge in E is of the form (i, j) such that $i < j$. Each gate in V has indegree equal to zero, one, or two. Also, each gate $i \in V$ has a *sort* $s(i) \in \{0, 1, \vee, \wedge, \neg\} \cup \{x_1, \ldots, x_n\}$. If $s(i) \in \{0, 1\} \cup \{x_1, \ldots, x_n\}$, then the indegree of i is zero. The gates with indegree zero are called the *inputs* of C. If $s(i) = \neg$, then i has indegree one. If $s(i) \in \{\vee, \wedge\}$, then the indegree of i must be two. Finally, node n is called the *output* of C.

This definition explains the syntax of circuits and the following semantics specifies a truth value for each appropriate truth assignment. Let $X(C)$ be the set of variables appearing in a circuit C and a truth assignment T *appropriate* for C is defined analogously.

Given such a T, the *truth value* of gate $i \in V$, denoted by $T(i)$, is defined recursively as follows. Initially, each input i of C is assigned by T such that $T(i) = 1$ if $s(i) = 1$, $T(i) = 0$ if $s(i) = 0$, and $T(i) = T(s(i))$ if $s(i) \in X$. If $s(i) = \neg$, then there is a unique $(j, i) \in E$ such that $j < i$. By induction, we know $T(j)$, and then $T(i) = 1$ if $T(j) = 0$, and vice versa. If $s(i) = \vee$, then there are $(j, i), (j', i) \in E$ such that $j, j' < i$. Similarly, we know $T(j)$ and $T(j')$, and then $T(i) = 1$ iff either $T(j) = 1$ or $T(j') = 1$. If $s(i) = \wedge$, then $T(i) = 1$ iff both $T(j) = 1$ or $T(j') = 1$. Finally, the *value of the circuit*, $T(C)$, is $T(n)$.

In analogy with Boolean functions expressed by Boolean expressions, a Boolean circuit with n-variable computes an n-ary Boolean function, and the converse is true.

PROBLEM 2.3.2. Given a circuit C, decide whether there exists a truth assignment T appropriate to C such that $T(C) = 1$. This problem is referred to as the *circuit satisfiability problem* and denoted by circuit SAT.

As we shall see below, circuit SAT is computational equivalent to the satisfiability problem, and thus presumably very hard. However, the same problem for circuits with no variable gates, referred to as the *circuit value problem* and denoted by CVP, obviously has a polynomial-time algorithm and it is another fundamental computational problem.

As we have mentioned so far, NP contains the satisfiability problem and the circuit satisfiability problem. In addition, NP certainly contains the reachability problem and the circuit value problem. It is intuitively clear, however, that the former two problems are somehow more worthy representatives of NP than the latter two. We shall now make this intuition precise and mathematically provable. We recall the notion of *reduction* for making it precise what does it mean for a decision problem to be at least as hard as another one. A natural definition of "effective reduction" is defined as follows.

DEFINITION 2.3.4. (Papadimitriou [50]). Let L_1 and L_2 be languages over finite alphabets Σ_1 and Σ_2, respectively. Then, L_1 is said to be *reducible* to L_2 if there exists a computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $x \in L_1$ iff $f(x) \in L_2$. The function f is called a *reduction from* L_1 *to* L_2. In particular, if $f \in \mathbf{DSPACE}(\log n)$, then L_1 is said to be log-*space reducible* to L_2 and f is called a log-*space reduction from* L_1 *to* L_2.

For example, the reachability problem is reducible to the circuit value problem and the circuit satisfiability problem is reducible to the satisfiability problem. However, no one has succeeded to prove the reducibility from the former two problems to the latter two. Thus, we shall be particularly interested in the maximal elements of this partial order.

DEFINITION 2.3.5. Let $\mathcal{C}$ be a complexity class, and let L be a language. If every $L' \in \mathcal{C}$ is log-space reducible to L, then L is said to be $\mathcal{C}$-*hard*. In particular, if $L \in \mathcal{C}$, then the L is said to be $\mathcal{C}$-*complete*.

It is not *a priori* clear that complete problems exist, indeed, no complete problem has been found in RP (i.e., the class of all languages accepted by polynomial Monte Carlo Turing machines). The first complete problem was provided for NP by Cook [17].

THEOREM 2.3.1. (Cook [17]). The satisfiability problem is NP-complete.

Stockmeyer and Meyer [69] proved that the satisfiability problem for *quantified Boolean formulae* is PSPACE-complete and Ladner [37] proved that the circuit value problem is P-complete.

Moreover, we have a useful property of reductions, that is, the fact that reducibility is transitive.

PROPOSITION 2.3.1. (Papadimitriou [50]). If f is a reduction from language L_1 to L_2 and f' is a reduction from language L_2 to L_3, then the composition $f \cdot f'$ is a reduction from L_1 to L_3.

Thus, it is straightforward that if L is $\mathcal{C}$-complete, $L' \in \mathcal{C}$, and L is log-space reducible to L', then L' is also $\mathcal{C}$-complete. Using this chain of reductions, a large number of decision problems has been proved to be complete for several complexity classes (cf., e.g., [23] and [50]). For the class of pattern languages or its subclasses, the following results concerning decision problems are known.

THEOREM 2.3.2. (Angluin [1]). The problem of deciding whether $w \in L(\pi)$ for given a string $w \in \Sigma^*$ and a pattern $\pi \in \mathcal{P}$ is NP-complete.

COROLLARY 2.3.1. (Angluin [1]). The problem of deciding whether $\pi_1 <' \pi_2$ for given patterns $\pi_1, \pi_2 \in \mathcal{P}$ is NP-complete.

THEOREM 2.3.3. (Miyano, Shinohara, and Shinohara [46]). The consistency problem within $\mathcal{P}_R$ is NP-complete, even if $||\Sigma|| = 2$.

THEOREM 2.3.4. (Wiehagen & Zeugmann [73]). Given a pair (T, F) of samples, the problem of constructing a pattern $\pi \in \mathcal{P}$ consistent with (T, F) is NP-hard*.

2.4 Language Learning via Queries

In this section we summarize Angluin's [7] terminology. First, a *hypothesis space* $\mathcal{H}$ for a class $\mathcal{L}$ of target languages is assumed. A *representation* of languages is specified by a 4-tuple $\mathcal{R} = (\Sigma, \Delta, R, \mu)$, where Σ and Δ are finite alphabets, R is a subset of Δ^* and μ is a mapping from R to subsets of Σ^*. A *language* is a subset of Σ^*. R is the set of *representations*, and μ is the map that specifies which language is represented by a given representation. The class of languages represented by $\mathcal{R}$ is just $\{\mu(r) | r \in R\}$.

*The ordinary consistency problem within $\mathcal{P}$ is known to be Σ_2^p-complete.

For each language L, we denote χ_L the *characteristic function* of L, that is, $\chi_L(w) = 1$ if $w \in L$ and $\chi_L(w) = 0$ otherwise.

For example, let us consider the class of regular languages. We may assume that the strings in R represent deterministic finite automata (DFAs). The *size* of a DFA M is the number of transition functions defined in M. We assume that there is a polynomial $p(s)$ such that every DFA of size at most s can be represented by a string in R of length at most $p(s)$. Moreover, we assume that for every $r \in R$, the DFA represented by r is of size at most $|r|$.

An unknown language L is selected from a target class $\mathcal{L}$ and information can be gathered about L by asking:

1. *Equivalence query*: An input is a representation $r \in R$. The response to this question is *yes* if $\mu(r) = L$ and *no* otherwise. In addition to the answer *no*, a *counterexample* w is arbitrarily selected from Σ^* and returned such that $\chi_L(w) \neq \chi_{\mu(r)}(w)$.

2. *Membership query*: An input is a string $w \in \Sigma^*$. The response to this question is $\chi_L(w)$, that is, *yes* if $w \in L$ and *no* otherwise.

We say that a deterministic algorithm A *exactly identifies* the class $\mathcal{L}$ with respect to the hypothesis space $\mathcal{R}$ using equivalence and membership queries iff for each $L \in \mathcal{L}$, when A runs with an oracle for equivalence and membership queries for L, the A eventually halts and outputs an $r \in R$ such that $\mu(r) = L$. Such an algorithm runs in time polynomial iff there is a polynomial $p(m, n)$ such that for every $L \in \mathcal{L}$, if m is the minimum length of any $r \in R$ such that $\mu(r) = L$, then at any stage in the run, the time used by A is bounded by $p(m, n)$, where n is the length of the longest counterexample provided so far in the run.

The first result on language learning via membership and equivalence queries for subclasses of CFLs is due to Angluin (cf. [4]). She introduced a data structure, called *observation table*, for regular languages as follows. Let L be a regular language over an alphabet Σ. An observation table for L is a two-dimensional matrix, consisting of three things: a nonempty finite *prefix-closed* set S of strings, a nonempty finite *suffix-closed* set E of strings, and a finite function $T : ((S \cup S \cdot \Sigma) \cdot E) \to \{0, 1\}$,

where a set is prefix-closed iff every prefix of each member of the set is also in the set. *Suffix-closedness* is defined analogously.

The interpretation of T is that $T(u) = 1$ iff $u \in L$. The observation table initially has $S = E = \{\varepsilon\}$ and it has a two-dimensional array with *rows* labeled by elements of $(S \cup S \cdot \Sigma)$ and *columns* labeled by elements of E with the entry for row s and column e equal to $T(s \cdot e)$. For $s \in (S \cup S \cdot \Sigma)$, $row(s)$ denotes the finite function $f : E \to \{0, 1\}$ defined by $f(e) = T(s \cdot e)$.

An observation table is called *closed* provided that for each $t \in S \cdot \Sigma$, there exists an $s \in S$ such that $row(t) = row(s)$. An observation table is called *consistent* provided that whenever s_1 and s_2 are elements of S such that $row(s_1) = row(s_2)$, for all $a \in \Sigma$, $row(s_1 \cdot a) = row(s_2 \cdot a)$.

Angluin [4] presented a polynomial-time algorithm that computes a closed, consistent observation table for a target language L using membership queries and she showed that DFAs computed from these tables converge to a correct one for L whenever tables are renewed by counterexamples returned.

THEOREM 2.4.1. (Angluin [4]). Every regular language L is learnable using membership and equivalence queries with respect to the hypothesis space DFAs in time polynomial in n and m, where n is the number of states of a minimum DFA M such that $L = L(M)$ and m is the length of a longest counterexample returned.

This result was extended to other subclasses of the context-free languages. The following results are interesting since classes studied properly includes the class of regular languages. For example, the language $\{\{a^n b^n \mid n \geq 1\}c\}^+$ is *deterministic one-counter*, but not *simple deterministic*. Contrarily, the $\{a^m b^n c a^n b^m \mid m, n \geq 1\}$ is simple deterministic, but not deterministic one-counter. These languages are not regular languages.

THEOREM 2.4.2. (Berman & Roos [12]). Every deterministic one-counter language L is learnable using membership and equivalence queries in time polynomial in n and m, where n is the number of states of a minimum deterministic one-counter automaton A such that $L = L(A)$ and m is the length of a longest counterexample returned.

THEOREM 2.4.3. (Takada [70]). Every even linear language L is learnable using membership and equivalence queries in time polynomial in n and m, where n is the number of nonterminals of a minimum even linear grammar G such that $L = L(G)$ and m is the length of a longest counterexample returned.

In particular, Ishizaka [29] presented a new method to identify a correct hypothesis grammar for the class of *simple deterministic languages*, which is different from Angluin's observation table. A context-free grammar G in Greibach normal form defined over terminals Σ and nonterminals N is called *simple deterministic* if for any $A \in N$, $a \in \Sigma$, and $\alpha, \beta \in N^*$, if $A \to a\alpha$ and $A \to a\beta$ are production rules of G, then $\alpha = \beta$. In Ishizaka's setting, membership queries and *extended equivalence queries* are assumed for a target language L. The extended equivalence query is a relaxation of equivalence query, that is, it asks whether $L = L(G)$ for a hypothesis grammar G in *2-standard form*, but it is not necessary that G is simple deterministic.

Ishizaka's algorithm [29] begins with a trivial hypothesis G such that $L(G) = \emptyset$. If a positive counterexample w is returned, then the algorithm introduces new nonterminals. It computes production rules to derive w using the nonterminals and puts all of them into the set of productions. If a negative counterexample w is returned, then the algorithm finds an incorrect production rule that is causing the derivation of the counterexample w by using membership queries. The algorithm continues the above routine until an extended equivalence query returns "yes". This algorithm outputs a grammar G in 2-standard form such that $L = L(G)$ for every target simple deterministic language L. However, the learnability of the simple deterministic languages in terms of minimally adequate teacher is still open.

THEOREM 2.4.4. (Ishizaka [29]). Every simple deterministic language L is learnable using membership and extended equivalence queries in time polynomial in n and m, where n is the number of nonterminals of a minimum grammar G in 2-standard form such that $L = L(G)$ and m is the length of a longest counterexample returned.

We shall refer to several results on the learnability of the whole class of context-free languages using queries. Angluin [3] showed that the class of context-free languages is learnable in time polynomial with respect to the hypothesis space of k-bounded

context-free grammars using membership queries, nonterminal membership queries, and equivalence queries.

However, we suspect that the problem of identifying a context-free grammar using only membership and equivalence queries is very hard. This problem was characterized by Angluin and Kharitonov [8]. They showed that there is no polynomial-time algorithm to learn the class of context-free languages in terms of minimally adequate teacher assuming the intractability of several cryptographic problems (e.g., *quadratic residues modulo a composite* and *inverting RSA encryption*).

Another result concerning the learnability of the whole class of context-free languages is due to Sakakibara [53]. He introduced *tree automata* that accept *structural strings*. A structural string of a target context-free grammar is computed from its derivation tree by replacing all labels of internal nodes by one special symbol not belonging to any alphabet. By this string, the learner knows the *skeleton* of the derivation tree. Sakakibara [53] extended the tree automata to Angluin's observation table and he assumed to the learner to use *structural membership queries* and *structural equivalence queries*.

THEOREM 2.4.5. (Sakakibara [53]). For every target context-free grammar G, using structural membership queries and structural equivalence queries, there exists an algorithm that outputs a grammar G' canonically equivalent to G such that $L(G) = L(G')$ in time polynomial in n and m, where n is the number of states of a minimum tree automaton for G and m is the length of a longest counterexample returned.

We next focus on results of language learning using special strings. On DFA learning, this idea goes back to Trakhtenbrot and Barzdin's [71] framework. They presented an algorithm for constructing the smallest DFA consistent with *a complete labeled sample*, that is, a sample that includes all strings upto a particular length and the corresponding label that states whether or not the string is accepted by the target DFA. However, the size of a complete labeled sample for a target DFA may be exponentially large in dependence on the size of the target.

Following this idea, Angluin [2] introduced a *live-complete set* of strings each of which contains a representative string for a target DFA. Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a DFA canonical for a regular language L. A state $p \in Q$ is called *live* if there exist

strings $x, y \in \Sigma^*$ such that $\delta(q_0, x) = p$ and $xy \in L(A)$. A *witness* of a live state p is each string x such that $\delta(q_0, x) = p$. The lexicographically first witness of a live state p is called the *canonical witness* of p. Then, a *live-complete set* for $L(A)$ is any finite subset of Σ^* that contains at least one witness for each live state of A. Note that the set of canonical witness of all live states of A is a live-complete set for $L(A)$.

THEOREM 2.4.6. (Angluin [2]). Every target DFA is exactly learnable using a live-complete set of examples and membership queries.

Ibarra and Jiang [28] studied the power of our learning model using equivalence queries only. A language L over an alphabet Σ is said to be *k-bounded regular language* if there exist strings $w_1, \ldots, w_k \in \Sigma^+$ such that $L \subseteq \{w_1^{i_1} \cdots w_k^{i_k} \mid i_1, \ldots, i_k \geq 0\}$.

THEOREM 2.4.7. (Ibarra & Jiang [28]). Every k-bounded regular languages L is learnable in time polynomial in n and m using equivalence queries, where n is the number of states of the canonical DFA for L and m is the maximum length of the counterexamples returned.

Moreover, Ibarra and Jiang [28] investigated how a partial ordering on counterexamples affects the learnability of formal languages. Two partial orderings on counterexamples returned by equivalence queries are considered, that is, *ordering by length* and *lexicographical ordering*. The following result tells us that lexicographical ordering on counterexamples contributes to DFA learning.

THEOREM 2.4.8. (Ibarra & Jiang [28]). Assuming that any equivalence query always returns the lexicographically first counterexample for the target and the hypothesis, every regular language L is learnable in time polynomial in n using equivalence queries, where n is the number of states of the canonical DFA for L.

When a target DFA with n states is defined over an alphabet Σ, Ibarra-Jiang's algorithm [28] learns the DFA from $O(||\Sigma||n^3)$ many counterexamples. Recently, this result has been improved to be $O(||\Sigma||n^2)$ by Birkendorf *et al.* (cf. [13]).

Oncina and García [49] defined another type of examples, the co-called *characteristic set* for a target DFA, and they proposed a polynomial-time algorithm to identify

the target DFA using this type of sample as follows. Let L be a regular language, and let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a canonical DFA such that $L = L(A)$. We denote $pr(L) = \{\alpha \mid \alpha\beta \in L\}$, $L_\alpha = \{\beta \mid \alpha\beta \in L\}$, and $pr_s(L) = \{\alpha \in pr(L) \mid \neg\exists\beta \in \Sigma^*,\ [L_\alpha = L_\beta, \beta < \alpha]\}$, where $<$ is a standard ordering of strings on Σ. Then, the *kernel* of L, denoted by $N(L)$, is the set $\{\varepsilon\} \cup \{\alpha a \in pr(L) \mid \alpha \in pr_s(L), a \in \Sigma\}$.

A sample $S = S_+ \cup S_-$ such that $S_+ \subseteq L(A)$ and $S_- \cap L(A) = \emptyset$ is said to be a *characteristic set of* L if the following conditions are satisfied.

1. $\forall\alpha \in N(L)$, if $\alpha \in L$, then $\alpha \in S_+$ else $\exists\beta \in \Sigma^*$ such that $\alpha\beta \in S_+$.
2. $\forall\alpha \in pr_s(L)$, $\forall\beta \in N(L)$, if $L_\alpha \neq L_\beta$, then $\exists\gamma \in \Sigma^*$ such that ($\alpha\gamma \in S_+$ and $\beta\gamma \in S_-$) or ($\alpha\gamma \in S_-$ and $\beta\gamma \in S_+$).

THEOREM 2.4.9. (Oncina & García [49]). Given a sample S of a regular language L, if S is a superset of a characteristic set of L, then there exists a polynomial-time algorithm to identify a DFA such that $L = L(A)$.

As compared with DFA learning, it seems that learning pattern languages is rather difficult. Angluin [6] showed that if each equivalence query may return an *arbitrary* counterexample, any algorithm for exact identification of pattern languages from equivalence and membership queries must ask exponentially many queries[†]. Thus, previous results on learning pattern languages requires special strings as additional information.

As we have mentioned above, Ibarra and Jiang [28] assumed an ordering by length on counterexamples, that is, a *shortest* counterexample is always returned. They proposed a learning model under this assumption and showed the following result.

THEOREM 2.4.10. (Ibarra & Jiang [28]). The class of pattern languages is learnable in time polynomial in n using equivalence queries that always return a shortest counterexample where n is the length of a target pattern.

Marron and Ko [43] considered necessary and sufficient conditions on a finite positive initial sample that would allow exact identification of a target k-variable pattern

[†]This upper bound does not collapse even if *subset queries* are allowed, but *superset* queries are sufficient for polynomial-time identification.

from the initial sample and from polynomially many membership queries. Subsequently, Marron [42] considered the learnability of k-variable patterns in the same model, but where the initial sample consists of only a single positive example of a target pattern. For the case of one-variable and two-variable patterns, Marron gave a careful analysis of the structural properties of initial examples that can cause his algorithm to fail. He also showed that only a small fraction of strings possess these properties.

CHAPTER 3

Learning Parenthesis Grammars

> A complete English text is beginning at a capital letter and ended by a period. Similarly, an HTML document consists of structural texts each of which is parenthesized by a beginning tag and the corresponding ending tag. In formal languages, these parentheses are useful for analysis of sentence structure of a grammar owing to its beneficial byproduct of the unambiguity. We study the contribution of these parentheses to formal language learning.

A membership query tells us one bit of information: whether or not a string is a member of an unknown language. Nevertheless, membership queries often play an important role in efficient learning (cf., e.g., [2, 4, 7, 8]). Furthermore, it seems that the claim of membership queries is reasonable because the membership problem is effectively decidable for CFGs. In other words, the capability of a teacher is incomplete, that is, not all the questions from a learner can be answered by a teacher and a membership query is a typical question a teacher can answer. In this chapter, we focus on membership queries, and our teacher can answer membership queries only.

On the other hand, the learning ability of a learner depends on the information a teacher provides. As an example, consider the case that a target language can be divided into some disjoint sub-languages. If a teacher gives a learner no information about one of these sub-languages (i.e., no example is given from a sub-language), then the learner can never identify the whole language. Thus, we assume a *careful* teacher, that is, the teacher carefully selects good examples from a target language. Informally, the task of a teacher is described as follows. A teacher divides a target language into a finite number of sub-languages each of which has a kind of representative elements. A

representative element of a language is a terminal string derived using all production rules of a grammar that generates the language. Such sub-languages and representative elements are called *complete* languages and *characteristic examples*, respectively.

We consider the problem of learning parenthesis languages [44] using characteristic examples and membership queries. A parenthesis language is a CFL in which ambiguity is avoided by the systematic and tedious use of parentheses.

The first section contains the definition of characteristic examples for CFGs. The properties of characteristic examples are investigated. In particular, we prove the decidability of characteristic examples for a given CFG. In the next section, our learning algorithm is presented. The input is a set of characteristic examples and finitely many membership queries are allowed. We prove the correctness of our algorithm and polynomial-time learnability of the parenthesis grammars. Finally, an open problem is discussed.

3.1 Characteristic Examples

In this section, we introduce the notion of *characteristic examples* for the class of CFGs and investigate their properties. A characteristic example is a string w such that there is a derivation of w requiring the application of each production. Thus, we have to ask whether or not every CFG does possess a characteristic example. As we shall see, this is not the case. Therefore, we additionally deal with the following question;

Does there exist an algorithm deciding whether or not a CFG possesses a characteristic example?

We answer this question affirmatively. Therefore, it remains to ask how to proceed, if a CFG does not possess a characteristic example. We solve this problem by providing a decomposition theorem for CFGs. That is, each CFG can be decomposed in finitely many sub-grammars each of which has a characteristic example. Finally, we provide a method for constructing characteristic examples for parenthesis grammars.

The class of parenthesis grammar has been introduced by McNaughton (cf. [44]). Note that the equivalence problem for CFGs is undecidable. He showed that the equivalence problem for parenthesis grammars is decidable. Furthermore Knuth [36] proved the decidability of the following problem. Given any CFG G, decide whether

or not there exists a parenthesis grammar G' such that $L(G) = L(G')$. We continue with the formal definition of parenthesis grammars.

DEFINITION 3.1.1. (McNaughton [44]). A *parenthesis grammar* is a CFG $G = \{N, \Sigma, P, S\}$ such that each production in P is of the form $A \to (\alpha)$, where $A \in N$, $(,) \in \Sigma$ and $\alpha \in ((N \cup \Sigma) \backslash \{(,)\})^*$.

A parenthesis language is a language L such that $L = L(G)$ for a parenthesis grammar G. Intuitively, a parenthesis language is intended to avoid ambiguity by the systematic use of parentheses, so that a sentential form (or terminal string) wears its syntactical structure on its sleeve. In each derivation, exactly one pair of parentheses is introduced each time a production is applied.

Now, we provide the theoretical background for our learning algorithm to be presented in the next section. In particular, we introduce the notion of a characteristic example. Furthermore, we establish the decomposability of CFLs into sublanguages possessing characteristic examples.

DEFINITION 3.1.2. Let G be a CFG and let $w \in L(G)$. We call w a *characteristic example* for G if there exists a derivation of w in which each production of G is applied at least once. A grammar G is said to be *complete* if G has a characteristic example.

Our learning algorithm requires characteristic examples of a target language as input. Therefore, we have to take care whether or not characteristic examples do always exist. As the following example shows there are even regular grammars not having a characteristic example. On the other hand, context-free but not regular grammars may possess characteristic examples as displayed in Example 3.1.1.

EXAMPLE 3.1.1. Let us consider the CFGs G_1 and G_2.

$G_1 = \{\{S, A, B\}, \{a, b\}, P_1, S\}$, where $P_1 = \{S \to AB, A \to aA|a, B \to bB|b\}$.

$G_2 = \{\{S, A, B\}, \{a, b\}, P_2, S\}$, where $P_2 = \{S \to A|B, A \to aA|a, B \to bB|b\}$.

$L(G_1)$ is regular, since it is $\{a^n b^m \mid n, m \geq 1\}$. Every string $a^n b^m$ is a characteristic example of G_1 for $n, m \geq 2$. On the other hand, clearly, $L(G_2) = \{a^n \mid n \geq 1\} \cup \{b^m \mid m \geq 1\}$, that is, $L(G_2)$ is also regular, and thus there is no characteristic example of G_2.

For overcoming this difficulty, we consider the decomposition of CFGs into finitely many 'sub-grammars'. Let us define the notion of sub-grammars by a relation $\sqsubseteq$ on CFGs as follows.

DEFINITION 3.1.3. Let $G_1 = (N_1, \Sigma_1, P_1, S)$ and $G_2 = (N_2, \Sigma_2, P_2, S)$ be CFGs. The grammar G_1 is said to be a *sub-grammar* of the grammar G_2 if $G_1 \sqsubseteq G_2$, where $G_1 \sqsubseteq G_2$ denotes $N_1 \subseteq N_2$, $\Sigma_1 \subseteq \Sigma_2$, and $P_1 \subseteq P_2$.

The next proposition shows that for each CFG, there are polynomially many sub-grammars each of which has a characteristic example.

PROPOSITION 3.1.1. Let $G = (N, \Sigma, P, S)$ be a CFG. There exist complete grammars $G_1, G_2, \cdots, G_k \sqsubseteq G$ such that $k \leq ||P||$ and $\cup_{i=1}^{k} P_i = P$, where P_i is the set of productions of G_i for all $i = 1, \ldots, k$.

PROOF. Let $L = L(G)$. Without loss of generality, we can assume that $A \to \varepsilon \notin P$ for each $A \in N \backslash \{S\}$. That is, if $\varepsilon \notin L$, then G has no ε-production, otherwise G has exactly one ε-production of the form $S \to \varepsilon$. Thus, we can assume that $\varepsilon \notin L$.

For every $w \in L$, there exists $G' \sqsubseteq G$ such that w is a characteristic example of G'. Since the number of nonempty subsets of P is $2^{||P||} - 1$, there exist complete grammars $G_1, G_2 \cdots, G_k \sqsubseteq G$ such that $k \leq 2^{||P||} - 1$ and $\cup_{i=1}^{k} P_i = P$, where P_i is the set of productions of G_i for all $i = 1, \ldots, k$.

If $k > ||P||$, then there exists G_i such that

1. $P_i \subset P_j$ for some $1 \leq i \neq j \leq k$, or

2. for every $r \in P_i$, there exists $j \in \{1, 2, \cdots, k\}$ such that if $i \neq j$, and $r \in P_j$.

First, assume that Condition 1 holds. We can remove G_i from $G_1, G_2 \cdots, G_k$ without losing the statement

$$\bigcup_{\substack{1 \leq \ell \leq k \\ \ell \neq i}} P_\ell = P.$$

Next, suppose that Condition 2 holds. It follows that

$$\bigcup_{1 \leq \ell \leq k} P_\ell = \bigcup_{\substack{1 \leq \ell \leq k \\ \ell \neq i}} P_\ell.$$

Thus, we can remove G_i from $G_1, G_2 \cdots, G_k$. Continuing this process until neither Condition 1 nor 2 are fulfilled, we obtain grammars $G_1, G_2 \cdots, G_k$ such that for each $r \in P$, there is exactly one production set P_i, $1 \leq i \leq k$ such that $r \in P$. Consequently, $k \leq ||P||$, and there exist $G_1, G_2 \cdots, G_k \sqsubseteq G$ such that $\cup_{i=1}^{k} P_i = P$. **Q.E.D.**

Let G and G' be CFGs, then G' is said to be *complete with respect to* G if $G' \sqsubseteq G$ and G' is complete. For unambiguous CFGs, we can show the uniqueness of characteristic examples of sub-grammars by the following proposition.

PROPOSITION 3.1.2. Let G be a parenthesis grammar and let $G_1, G_2, \cdots, G_m$ be any set of complete grammars with respect to G obtained by Proposition 3.1.1. Let W_i, $1 \leq i \leq m$ be the sets of characteristic examples for G_i. Then for all $1 \leq i, j \leq m$: $W_i \cap W_j = \emptyset$ iff $i \neq j$.

PROOF. Since $i = j$ implies $W_i = W_j$, we assume $i \neq j$. Let P_i and P_j be sets of productions of G_i and G_j, respectively. By Proposition 3.1.1, $G_i \not\sqsubseteq G_j$ and $G_j \not\sqsubseteq G_i$. Thus there exists a production $r \in P_i \backslash P_j$. Every $w_i \in W_i$ is derived using all production rules in P_i and every $w_j \in W_j$ is not derived using the production rule $r \in P_i$. Since G is unambiguous, G_i and G_j both are unambiguous. Hence, $W_i \cap W_j = \emptyset$. **Q.E.D.**

We next deal with the following problem for CFGs.

PROBLEM 3.1.1. Given a CFG G, decide whether there exists a characteristic example of G.

For the problem, let us use the following notations. Let $G = (N, \Sigma, P, S)$ be any CFG. A sentential form of G containing a nonterminal $A \in N$ is denoted by $\beta[A]$. For a nonterminal $A \in N$, we write $\alpha \Rightarrow_A^* \beta$ if there exists a derivation from α to β such that $\alpha \Rightarrow^* \gamma_1 A \gamma_2 \Rightarrow^* \beta$. For a production rule $A \rightarrow w$ in P, we write $\alpha \Rightarrow_{A \rightarrow w}^* \beta$ if there exists a derivation from strings α to β such that $\alpha \Rightarrow^* \gamma_1 A \gamma_2 \Rightarrow \gamma_1 w \gamma_2 \Rightarrow^* \beta$.

For a derivation tree T of G, let $d(T)$ denote the depth of T. For a nonterminal $A \in N$, let $n(T)_A$ denote the number of internal nodes of T labeled by A. For a production rule $A \rightarrow w \in P$, let $n(T)_{A \rightarrow w}$ denote the number of internal nodes of T labeled by A such that w is the concatenation of labels of its children. For a set $P' \subseteq P$ of production rules, let $T_{P'}$ denote a derivation tree such that for every $r \in P'$, $n(T_{P'})_r \geq 1$.

DEFINITION 3.1.4. Let A be a nonterminal of a CFG G. We call A *bounded* if there exists a constant k such that for every derivation tree T of G, $n(T)_A \leq k$.

DEFINITION 3.1.5. A production rule r of a CFG G is said to be *bounded* if there exists a constant k such that for every derivation tree T of G, $n(T)_r \leq k$.

LEMMA 3.1.1. It is decidable whether or not a nonterminal of a CFG is bounded.

PROOF. We first prove that the following conditions are equivalent for any CFG $G = (N, \Sigma, P, S)$.

1. A nonterminal $A \in N$ is not bounded.

2. For a nonterminal $A \in N$, there exists a $B \in N$ such that $B \Rightarrow_A^* \beta[B]$.

It is clear that Condition 2 implies 1. Now, we assume that there exists no $B \in N$ that satisfies Condition 2. Let T be a derivation tree of G. For any subtree of T whose root a_1 is labeled by A, it has no internal node labeled by A except a_1. If the length of the path of T from the root to a_1 is greater than $|N|$, then there exist two or more internal nodes of T labeled by $B \in N$ in the path. Let b_1 and b_2 be such internal nodes in root-to-leaf order. Let T_1 and T_2 be subtrees of T whose root are b_1 and b_2, respectively. Since both T_1 and T_2 have exactly one internal node labeled by A, and since T_2 is a subtree of T_1, the tree obtained by replacing T_1 of T by T_2 is a derivation tree of G. Thus, for any derivation tree T of G, there exists a derivation tree T' of G such that $n(T)_A = n(T')_A$ and $d(T') \leq |N|$. Hence, the A is bounded. **Q.E.D.**

We note that for every CFG $G = (N, \Sigma, P, S)$ and all strings $\alpha, \beta \in (N \cup \Sigma)^*$, it is decidable whether or not $\alpha \Rightarrow^* \beta$. Let m be the maximum length of right sides of production rules in P. For every $B \in N$ and $\beta[B]$, we can decide whether or not $B \Rightarrow^* \beta[B]$, where the length of $\beta[B]$ is at most $m^{\|P\|}$. For each nonterminal $A \in N$, the A is not bounded if $B \Rightarrow_A^* \beta[B]$, and the A is bounded otherwise.

LEMMA 3.1.2. It is decidable whether or not a production rule of a CFG is bounded.

PROOF. Let $G = (N, \Sigma, P, S)$ be a CFG. Clearly, every production rule of the form $S \to w_S$ ($w_S \in (N \cup \Sigma)^*$) is bounded. Without loss of generality, let a production rule $r \in P$ be of the form $A \to w_A$, where $A \in N \setminus \{S\}$ and $w_A \in (N \cup \Sigma)^*$. We prove that for such an $r \in P$, the following conditions are equivalent.

1. $r \in P$ is not bounded.

2. There exists a $B \in N$ such that one of the following conditions is satisfied.

   (a) $B \Rightarrow^* \beta[B]$ and $\beta[B]$ contains A.

   (b) $B \Rightarrow^* \beta_1[A]$ and $A' \Rightarrow^* \beta_2[B]$, where $A' \in N$ is contained in w_A.

It is clear that Condition 2 implies 1. Now, we contrarily assume Condition 1. Since A is not bounded, there exists $B \in N$ such that $B \Rightarrow_A^* \beta[B]$, that is, it holds that $B \Rightarrow^* \alpha_1 A \alpha_2 \Rightarrow^* \beta[B]$ for some strings $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$. If B is derived from a nonterminal contained in α_1 or α_2, then Condition (a) of 2 holds. If B is derived from a nonterminal contained in w_A, then Condition (b) of 2 holds. ●therwise, for any derivation tree T of G, the number $n(T)_r$ is less than the number of internal nodes of a derivation tree of G having depth less than or equal to $||N||$. Thus, the r is bounded. This contradicts that r is not bounded. Hence, Condition 2 holds.

Similarly as above, for any production rule in P, we can decide whether or not one of Condition (a) and (b) of 2 holds. **Q.E.D.**

Now, we are ready to solve the decision problem for characteristic examples and prove this problem in the following theorem.

THEOREM 3.1.1. It is decidable whether or not a CFG has a characteristic example.

PROOF. Let $P' \subseteq P$ be a set of bounded production rules of a CFG $G = (N, \Sigma, P, S)$. By Lemma 3.1.2, membership in P' is decidable. First we prove that the following conditions are equivalent.

1. The G has a characteristic example.

2. There exists a derivation tree $T_{P'}$ of G.

It is clear that Condition 1 implies 2. For the converse direction, assume that Condition 2 holds. Let $r \in P \setminus P'$. There exists a nonterminal $A_1 \in N$ such that $A_1 \Rightarrow_r^* \beta_1[A_1]$. If $T_{P'}$ has an internal node labeled by A_1, then there exists a derivation tree T of G such that $n(T_{P'})_{r' \in P'} \leq n(T)_{r' \in P'}$ and $n(T)_r \geq 1$. If not, any production rule $r_1 \in P$ containing A_1 is not bounded. If there exists no $A_2 \in N$ such that

$A_2 \Rightarrow^*_{r_1} \beta_2[A_2]$ and $A_2 \neq A_1$, then A_1 must be contained in the right side of a production rule of the form $S \to w_S \in P'$. This contradicts that $T_{P'}$ has no internal node labeled by A_1. Thus, $A_2 \neq A_1$.

Since there are only finitely many nonterminals, we can find an $A_k \in N$ such that $A_k \Rightarrow^*_{r_{k-1}} \beta_k[A_k]$, $A_{k-1} \Rightarrow^*_{r_1} \beta_2[A_{k-1}]$, ..., $A_1 \Rightarrow^*_r \beta_1[A_1]$ and the tree $T_{P'}$ has an internal node labeled by A_k, namely, there exists a derivation tree T of G such that $n(T_{P'})_{r' \in P'} \leq n(T)_{r' \in P'}$ and $n(T)_r \geq 1$. Thus, Condition 2 implies 1.

Finally, we prove that it is decidable whether or not there exists $T_{P'}$. Let T be a derivation tree of G. If $d(T) > ||N||$, then for a production rule $(A \to w) \in P$, there exists an internal node a_1 of T labeled by A such that w is the concatenation of labels of its children and $n(T_1)_{A \to w} \geq 2$, where T_1 is the subtree of T whose root is a_1. Since $n(T_1)_{A \to w} \geq 2$, there exists an internal node a_2 $(\neq a_1)$ of T_1 labeled by A such that w is the concatenation of labels of its children. Let T_2 be a subtree of T_1 whose root is a_2. A tree obtained by replacing T_1 of T by T_2 is also a derivation tree of G. Thus, there exists a derivation tree T' of G such that $n(T)_{r' \in P'} = n(T')_{r' \in P'}$ and $d(T') \leq ||N||$. Hence, we can decide whether or not there exists $T_{P'}$ by enumerating all derivation trees of depth less than or equal to $||N||$. **Q.E.D.**

3.2 Learning Parenthesis Languages

In this section, we study the learnability of the class of all parenthesis grammars. The scenario is as follows. An unknown target parenthesis language L represented by a parenthesis grammar G has to be learned. We assume that G is in normal form, that is, a reduced, invertible parenthesis grammar. The learner receives finitely many characteristic examples for G as input. Additionally, our learning algorithm has access to an oracle that answers membership queries.

Next, we describe the general idea behind our algorithm. The algorithm is denoted by $\boldsymbol{A}$. In any derivation of G, exactly one pair of parentheses '(' and ')' is introduced at every application of every production rule. Thus, we can describe a 'silhouette' of a derivation tree according to the derivation, that is, a tree having no label for its internal nodes. However, these parentheses are of just one species and have no subscript. Therefore, the parentheses do not tell us which production introduced them,

or even from which nonterminal they came from. Thus, the main part of the $\boldsymbol{A}$ is to restore the original derivation tree from the silhouette.

We first define an equivalence relation over nodes of derivation trees of a parenthesis grammar. The algorithm uses this relation as its success criterion: if this relation holds on two nodes, then the $\boldsymbol{A}$ assigns them the same label (nonterminal), otherwise he assigns them different labels. The correctness of this assignment is proved in the next section. Moreover, we also prove that the running time of our algorithm is bounded by a polynomial in the number of production rules of the target G and in the length of a longest characteristic example provided.

For the discussion below, we give the definition of replacement of *subtrees* or *co-subtrees* of trees. We also define the specific membership queries used by our learning model.

For any tree t, let us denote the root of t by $rt(t)$. The label of a node x of t is denoted by $t(x)$. The frontier of t is denoted by $fr(t)$. Let N and E be the sets of nodes and edges of t, respectively. The *subtree* of t on a node $x \in N$, written t/x, is the tree with $N_{t/x} \subseteq N$ and $E_{t/x} \subseteq E$ such that

1. a node $y \in N$ is in $N_{x/t}$ iff there exist $(x, x_1), (x_1, x_2), \cdots, (x_n, y) \in E$ for some $n \geq 1$ or $x_1 = y$, and
2. an edge $(x_i, x_j) \in E$ is in $E_{t/x}$ iff $x_i, x_j \in N_{t/x}$ and $(x_i, x_j) \in E$.

The *co-subtree of* t on a node $x \in N$, written $t\backslash x$, is the tree with $N_{t\backslash x} \subseteq N$ and $E_{t\backslash x} \subseteq E$ such that

1. $N_{t\backslash x} = (N \backslash N_{t/x}) \cup \{x\}$ and
2. $E_{t\backslash x} = E \backslash E_{t/x}$.

It is easy to see that if t is a tree, then $rt(t/x) = x$ and $rt(t\backslash x) = rt(t)$. In Figure 3.1, we display an example for a tree, a subtree, and a co-subtree, respectively.

Let $t_1 = (N_1, E_1)$ and $t_2 = (N_2, E_2)$ be trees such that $N_1 \cap N_2 = \emptyset$. Let x be a leaf of t_1. Then, we define the tree $t_1 \#_x t_2 = (N_\#, E_\#)$ as follows.

1. $N_\# = (N_1 \cup N_2) \backslash \{x\}$, and

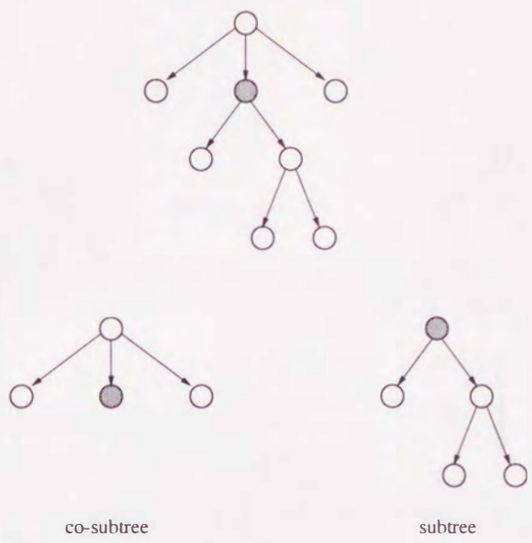

Figure 3.1: A subtree and co-subtree on a node of a tree.

2. $E_{\#} = (E_1 \cup E_2 \backslash \{(y, x)\}) \cup \{(y, rt(t_2))\}$.

Let $x_1 \in N_1$, and let $x_2 \in N_2$. More generally, we define the tree $t\backslash x_1 \#_{x_1} t_2/x_2$ obtained by replacing the subtree t_1/x_1 by the subtree t_2/x_2. Given trees t_1 and t_2, it is clear that x_1 is a leaf of the tree $t_1\backslash x_1$. In this case we omit the subscript of #, that is, instead of $t\backslash x_1 \#_{x_1} t_2/x_2$ we write $t\backslash x_1 \# t_2/x_2$.

Next, we recall the definition of a 'skeleton', a special type of trees.

DEFINITION 3.2.1. (Sakakibara [53]). Let $t = (N, E)$ be a tree and V be the set of labels of nodes of t. The *skeletal description* of t, written s_t, is a tree with N and E such that for each $x \in N$,

$$s_t(x) = \begin{cases} t(x), & \text{if } x \text{ is a leaf,} \\ \$, & \text{otherwise,} \end{cases}$$

where \$ is a special symbol not in V.

Intuitively, the skeletal description of a tree is with labeled leaves, and all internal nodes labeled by \$. In this thesis, the term '*skeleton*' is used for the skeletal description of a derivation tree of a parenthesis grammar.

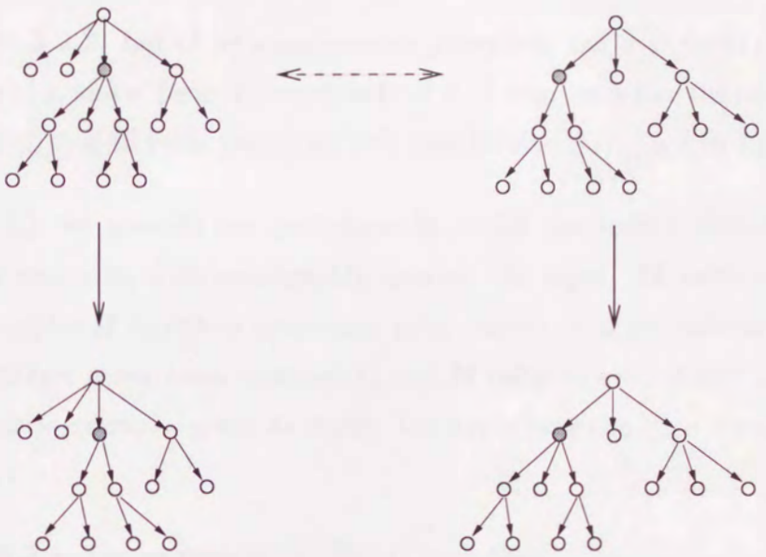

Figure 3.2: Replacement of subtrees on a node.

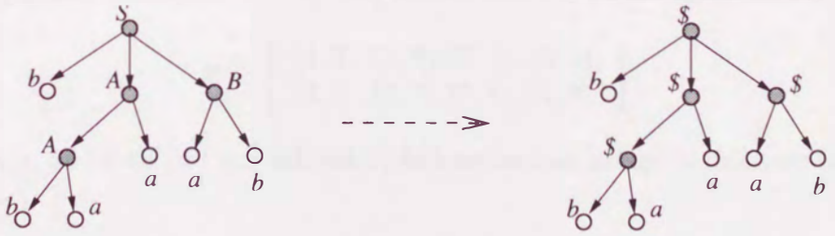

Figure 3.3: The skeletal description of a tree.

Let G be a parenthesis grammar. The set of derivation trees of G is denoted by $T(G)$. The set of skeletons of $t \in T(G)$ is denoted by $s(T(G))$. Next, we define a relation over $s(T(G))$.

DEFINITION 3.2.2. Let G be a parenthesis grammar. Let $s, s' \in s(T(G))$, a be an internal node of s, and a' be an internal node of s'. Then, we define the relation $\equiv_{G(s,s')}$ such that $a \equiv_{G(s,s')} a'$ iff both $fr(s\backslash a\#s'/a')$ and $fr(s'\backslash a'\#s/a)$ are in $L(G)$.

In Table 3.1, we provide the procedure $\boldsymbol{M}$ which determines derivation trees of characteristic examples with membership queries. As input, $\boldsymbol{M}$ takes a set of characteristic examples of complete grammars with respect to a parenthesis grammar G generating a target parenthesis language L, and $\boldsymbol{M}$ outputs a set of derivation trees for the characteristic examples given as input. For explaining the basic ideas, we include Example 3.2.1.

EXAMPLE 3.2.1. Let us consider a CFG G such that

$$G = \{\{S, A, B\}, \{a, b, (,)\}, P, S\}, \text{ where}$$

$$P = \{S \to (AB), A \to (aB)|(a), B \to (bA)|(b)\}.$$

The grammar G is an invertible parenthesis grammar. There exists a derivation of G for the string $w = ((a(b))(b(a)))$ in which every production is applied. Thus, the G is also complete. From this characteristic example, we can compute the unique tree $t = (N, E)$, where $N = \{1, 2, \cdots, 9\}$ and E contains the following elements.

$$E = \left\{ \begin{array}{l} (1,2), (1,3), (2,4), (2,5), \\ (3,6), (3,7), (5,8), (7,9) \end{array} \right\}$$

Moreover, all labels of t are defined as follows and an image of this tree is displayed in Figure 3.4.

$$t(i) = \left\{ \begin{array}{ll} a, & \text{if } i = 4 \text{ or } 9, \\ b, & \text{if } i = 8 \text{ or } 6, \text{ and} \\ \$, & \text{otherwise.} \end{array} \right.$$

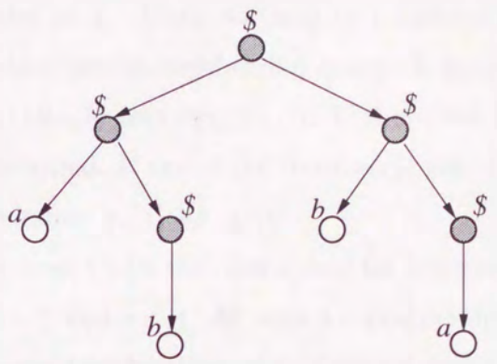

Figure 3.4: An image of skeleton.

Procedure $\boldsymbol{M}(\hat{s})$; $\hat{s} = \{s_1, \cdots, s_m\}$ of skeletons of characteristic examples

begin
  for each $i = 1, \ldots, m$ /* First loop */
    for each nodes j and k of s_i
      if $j \equiv_{G(s_i,s_i)} k$, **then** /* by membership queries */
        rename $s_i(k)$ by $s_i(j)$
      else;
  for each $i = 1, \ldots, m-1$ and $j = i+1, \ldots, m$ /* Second loop */
    for each nodes i' of s_i and j' of s_j
      if $i' \equiv_{G(s_i,s_j)} j'$, **then** /* by membership queries */
        rename $s_j(j')$ by $s_i(i')$
      if $i' \not\equiv_{G(s_i,s_j)} j'$ and $i' = j'$, **then**
        rename $s_j(j')$ by a new label k
      else;
  output $\hat{s}$;
end

Table 3.1: The procedure $\boldsymbol{M}$ to decide derivation trees.

Now, we return to the explanation of the general behavior of our procedure $\boldsymbol{M}$. Let L be a target grammar and let $\hat{s} = \{s_1, s_2, \cdots, s_m\}$ be a set of skeletons for m characteristic examples of L. First, for any two nodes i and j of a skeleton s_k $(1 \leq k \leq m)$, our procedure uses a membership query. A membership query proposes two frontiers of trees $s_k\backslash i\#s_k/j$ and $s_k\backslash j\#s_k/i$. If these two frontiers are both in L, then the answer *yes* is returned. If one of the frontiers is not in L, then *no* is returned. In case of *yes*, the $\boldsymbol{M}$ renames $s_k(j)$ by $s_k(i)$.

Furthermore, for any node i' of a skeleton s_i and for any node j' of another skeleton s_j such that $1 \leq i \leq m-1$ and $i < j$, $\boldsymbol{M}$ uses a membership query. In this stage, a membership query proposes two frontiers of $s_i\backslash i'\#s_j/j'$ and $s_j\backslash j'\#s_i/i'$. If these two frontiers are both in L, then the answer *yes* is returned. If one of the frontiers is not in L, then *no* is returned. In case of *yes*, $\boldsymbol{M}$ renames $s_j(j')$ of s_j by $s_i(i')$ of s_i. In case of *no* and $i' = j'$, procedure $\boldsymbol{M}$ introduces a new label k and renames $s_j(j')$ of s_j by k.

Finally, the procedure outputs a refined set $\hat{s}$ as a set of derivation trees of a grammar for the target parenthesis language L. Our algorithm $\boldsymbol{A}$ computes a parenthesis grammar $G' = (N, \Sigma, P, S)$ using such a refined $\hat{s} = \{s_1, s_2, \cdots, s_m\}$. Since each s_i $(1 \leq i \leq m)$ is a derivation tree for a characteristic example, the N, \$, and P are effectively computable. For example, if $s \in \hat{s}$ has an internal node i such that its children are $j_1, j_2, \cdots, j_k$ in left-to-right order, then the algorithm $\boldsymbol{A}$ makes a production rule $s(i) \rightarrow s(j_1)s(j_2)\cdots s(j_k)$.

LEMMA 3.2.1. Let a and b be internal nodes of a derivation tree T of a reduced invertible parenthesis grammar G. Then $T(a) = T(b)$ iff $a \equiv_{G(T,T)} b$.

PROOF. Clearly, if two labels of a and b are equal, then $a \equiv_{G(T,T)} b$. We assume $a \equiv_{G(T,T)} b$. Since G is invertible, if two frontiers of s/a and s/b are equal, then $T(a)$ and $T(b)$ are also equal.

Let two frontiers of s/a and s/b be not equal. The grammar G has two production rules of the form $A \rightarrow (w_1T(a)w_2)$ and $A' \rightarrow (w_1'T(b)w_2')$, where A and A' are nonterminals and w_1, w_2, w_1' and w_2' are sentential forms. G also has two production rules of the form $A \rightarrow (w_1T(b)w_2)$ and $A' \rightarrow (w_1'T(a)w_2')$.

Hence, it follows that any sentential from of G containing $T(a)$ or $T(b)$ are of the form $W_1(w_1Bw_2)W_2$ or $W_1'(w_1'Bw_2')W_2'$ for a nonterminal $B \in \{T(a), T(b)\}$, where W_1, W_2, W_1' and W_2' are sentential forms. Thus, a string $\alpha T(a)\beta$ is derived from G iff the string $\alpha T(b)\beta$ is derived from G. Since no two distinct nonterminals of G are equivalent, we conclude that the strings $T(a)$ and $T(b)$ are equal. **Q.E.D.**

LEMMA 3.2.2. Let G be an invertible parenthesis grammar. Let a and a' be internal nodes of derivation trees T and T' of G, respectively. Then, $T(a) = T'(a')$ iff $a \equiv_{G(T,T')} a'$.

PROOF. Analogous to Lemma 3.2.1. **Q.E.D.**

From Lemma 3.2.1 and 3.2.2, we conclude that for a target parenthesis language, our algorithm eventually terminates and outputs a parenthesis grammar which generates the target parenthesis language. We now analyze the time complexity of our algorithm.

LEMMA 3.2.3. The time used by the procedure $\boldsymbol{M}$ is bounded by a polynomial in the number of characteristic examples initially given and in the length of a longest characteristic example returned.

PROOF. It is sufficient to show that the total number of membership queries is bounded by a polynomial in the number of characteristic examples, denoted by m, and in the length of a longest characteristic example by n.

For any characteristic example w, its skeleton has at most $c|w|^2$ internal nodes, where c is a constant. In order to decide whether or not any two labels of internal nodes of the skeleton are equal, the procedure $\boldsymbol{M}$ uses at most $(c|w|^2 - 1) + (c|w|^2 - 2) + \cdots + (c|w|^2 - (c|w|^2 - 1)) = \frac{1}{2}(c|w|^2 + 2)(c|w|^2 - 1)$ many membership queries.

For any two characteristic examples w_1 and w_2 such that $|w_1| \leq |w_2|$, their skeletons have at most $c|w_2|^2$ internal nodes. In order to decide whether or not any two labels of internal nodes of the skeletons are equal, the $\boldsymbol{M}$ uses at most $c^2|w_2|^4$ many membership queries.

Thus, the total number of membership queries used by the procedure is at most $\frac{1}{2}m(cn^2 + 2)(cn^2 - 1) + \frac{1}{2}c^2(m-1)(m-2)n^4 = \mathcal{O}(m^2n^4)$. **Q.E.D.**

LEMMA **3.2.4.** The total number of characteristic examples to decide a parenthesis grammar for a target language is bounded by the number of production rules of a minimal invertible parenthesis grammar.

PROOF. Let $G = (N, \Sigma, P, S)$ be an invertible parenthesis grammar for a target. By Proposition 3.1.1, there exist complete grammars $G_1, \cdots, G_k$ with respect to G such that $\cup_{i=1}^{k} P_i = P$ and $k \leq ||P||$, where P_i is the set of productions of G_i for all $i = 1, \ldots, k$.

Let w_i be a characteristic example of G_i. By Proposition 3.1.2, the string w_i is not a characteristic example of any other $G_j \in \{G_1, \cdots, G_k\} \setminus \{G_i\}$. By Lemma 3.2.1 and 3.2.2, a grammar G_i' is decidable such that $L(G_i) = L(G_i')$ and $L(G_i') \neq L(G_j)$ for any other $G_j \in \{G_1, \cdots, G_k\} \setminus \{G_i\}$.

Thus, given characteristic examples $w_1, \cdots, w_k$ of the grammars $G_1, \cdots, G_k$, we can compute parenthesis grammars $G_1' \cdots, G_k'$ such that $k \leq ||P||$ and $L(G_i) = L(G_i')$ for all $i = 1, \ldots, k$. **Q.E.D.**

Putting it all together, we obtain the following theorem.

THEOREM **3.2.1.** The class of parenthesis grammars is learnable with respect to reduced, invertible parenthesis grammars as hypothesis space using membership queries and characteristic examples in time polynomial in the length of a longest characteristic example provided and in the number of production rules of a minimal grammar for an unknown target.

3.3 Discussion

We have introduced the notion of characteristic examples for CFGs. and discussed their properties. In particular, it can be effectively decided whether or not a CFG has a characteristic example. Consequently, in our learning model for parenthesis grammars, we assumed that characteristic examples are given for the learning algorithm as input instead of using equivalence queries. The class of parenthesis grammars is learnable in time polynomial in the length of the longest characteristic example and in the size of a minimum grammar for the target using membership queries and characteristic examples.

However it remains open whether the time complexity of our model is polynomial in the sense of Angluin (cf. [6]). That is, when we consider the length of examples, characteristic examples may be very long compared with counterexamples returned in response to equivalence queries.

As future work we consider learning of CFGs using characteristic examples and structural membership queries: The teacher gives 'plain' characteristic examples (that have no additional information) as input and answers the membership queries for structural strings. It is necessary to show that an algorithm itself can decide a skeletal description of each characteristic example of a CFG using structural membership queries.