Vol. 34 No. 3 Transactions of Information Processing Society of Japan Mar. 1993

Regular Paper

Polynomial-time MAT Learning of C-Deterministic Context-free Grammars

HIROMI SHIRAKAWA † and TAKASHI YOKOMORI ††

This paper concerns the learning of context-free grammars, and introduces a subclass which we call *context-deterministic* (*c-deterministic*) grammars. The corresponding language class properly contains the classes of regular languages and of even linear languages, and is incomparable to the classes of simple deterministic languages and of one-counter languages. We show that the class of c-deterministic grammars is learnable in polynomial time from (extended) minimally adequate teacher (MAT), which gives a generalization of the corresponding results on regular languages in Ref. 3) and even-linear languages in Ref. 9).

1. Introduction

An inductive inference can be intuitively viewed as the process of extracting a small finite representation from a large number of data. In an abstract setting, a well-defined class of objects (the domain) is fixed to be inferred, and a representation space for the domain is assumed. Further, a protocol for presenting examples of an object in the domain is also specified. Given a targeted object L, the goal here is to find a correct representation G in the space for L using a finite set of examples of L. Sometimes several oracles playing a role of a teacher is assumed to make the inference efficient. We discuss a problem of grammatical inference, where the domain consists of (formal) languages, and objects are represented by grammars.

In Ref. 3), Angluin introduced the model of learning called *minimally adequate teacher* (MAT), that is, learning from membership queries and equivalence queries. Since she presented an algorithm which learns regular languages using deterministic finite automata (DFAs) in polynomial time from MAT, several extended results about polynomial-time MAT learnability for learning subclasses of context-free grammars have been shown[4),7)]; however, the polynomial-time learnability of the whole class of context-free grammars is still open.

In this paper, we introduce the class of *context-deterministic* (*c-deterministic*) grammars, which is a subclass of the class of context-free grammars, and show that the class is learnable in polynomial time from MAT. Since the class of languages generated by c-deterministic grammars properly contains the classes of regular languages and of even linear languages, this gives a generalization of the corresponding results for regular languages in Ref. 3) and even-linear languages in Ref. 9).

The idea used in this paper is that all the possible nonterminals and rules are introduced from positive counterexamples, and among these rules, wrong ones are removed by negative counterexamples. A similar idea is employed in Ref. 2), 7). The paper[2)] discusses the problem of learning of k-bounded context-free grammars in which any production rule has at most k nonterminals on the righthand side. In her setting, the set of nonterminals is known by the algorithm, and only production rules are to be learned. The learning algorithm presented there is similar to our algorithm described later, except for using *noneterminal membership* queries to find such wrong rules, which is a fairly strong requirement for the teacher. Hence, the most important issue is how to determine the wrong rules without nonterminal membership queries. In fact, the latter paper (Ref. 7)) discusses such a method for learning a subclass called simple deterministic grammars, in which a single nonterminal membership query is replaced by a number of membership queries. Here, we introduce c-deterministic grammars whose properties make it possible to determine wrong rules without using nonterminal membership queries.

† Department of Informatics, Sanno College
†† Department of Computer Science and Information Mathematics, University of Electro-Communications

In a c-deterministic grammar, intuitively, for each nonterminal appearing in the derivation for a terminal string, there exists a pair of prefix and suffix of the string which specifies the nonterminal. For example, consider the following grammar $G=(N, \Sigma, P, S)$, that is a c-deterministic grammar, where $N=\{S, I, C, T, E\}$, $\Sigma=\{$*if, then, else,* ⟨*condition*⟩, ⟨*statement*⟩$\}$ and P consists of the following rules.

$S \rightarrow ICTSES \mid$⟨*statement*⟩
$I \rightarrow$ *if*
$C \rightarrow$⟨*condition*⟩
$T \rightarrow$ *then*
$E \rightarrow$ *else*

It is easy to see that this grammar generates all the statements of "if-then-else" structures. Let us see the concept of c-determinism through the property of this grammar. Suppose that an unknown string "*if* ⟨*condition*⟩ α ⟨*statement*⟩ *else* ⟨*statement*⟩" is given, where α indicates an unknown substring. Then, it is easy to guess that the unknown substring is "*then*", that is, the string derived from the nonterminal T. We can see that for any terminal string derived by this grammar, any nonterminal which appears in the derivation is specified by the prefix and suffix of the string, that is, they specify all the strings derived from the nonterminal. Because of this property, the learning algorithm for the class of c-deterministic grammars becomes simpler than those previously mentioned.

In the next section, we give some definitions and notation used in this paper. The formal definition of c-deterministic grammars is given in Section 3. In Section 4. 1, a learning algorithm for c-deterministic grammars is given, and its correctness and time complexity are discussed in Section 4. 2. Further, we entablish the relationships between several classes of languages and the class of c-deterministic context-free languages in Section 5.

2. Preliminaries

2. 1 Definitions and Notation

We assume the reader to be familiar with the rudiments of formal language theory (See, e. g., Ref. 5)), and we only give some basic definitions and notation used in this paper.

For a given finite alphabet Σ, the set of all strings with finite length (including *zero*) is denoted by Σ^*. (An *empty* string is denoted by λ.) $lg(w)$ denotes the length of a string w. Σ^+ denotes $\Sigma^*-\{\lambda\}$. A *language* L over Σ is a subset of Σ^*. A language L is *λ-free* iff λ is not contained in L. If two languages L and L' are equivalent ignoring λ, two languages are said to be *equivalent modulo* λ and denoted by $L \underset{\lambda}{=} L'$. For any strings $x, y \in \Sigma^*$ and any languge L over Σ, let $x\backslash L=\{y \mid xy \in L\}(L/x=\{y \mid yx \in L\})$. A language $x\backslash L$ (resp. L/x) is called *left* (resp. *right*) *derivative of* L *with respect to* x.

A *context-free grammar* is denoted by $G=(N, \Sigma, P, S)$, where N and Σ are alphabets of *nonterminals* and *terminals,* respectively, such that $N \cap \Sigma=\phi$. P is a finite set of rules: each rule is of the form $A \rightarrow \alpha$, where A is a nonterminal and α is a string of symbols from $N \cup \Sigma$. S is a special nonterminal called the *start symbol.* A grammar G is *λ-free* iff no rule has λ on its righthand side. Two grammars G and G' are *equivalent* iff $L(G)=L(G')$ holds.

If $A \rightarrow \beta$ is a rule of P, and α and γ are arbitrary strings in $(N \cup \Sigma)^*$, then we may write $\alpha A \gamma \Rightarrow_G \alpha \beta \gamma$. The notation $\overset{*}{\Rightarrow_G}$ is the reflexive and transitive closure of $\Rightarrow_G$. For $A \in N$, let $L_G(A)=\{w \in \Sigma^* \mid A \overset{*}{\Rightarrow_G} w\}$. (In each case, the subscript G is abbreviated as in $\Rightarrow$ and $L(A)$, respectively, when it is clear from the context.) In particular, $L(S)$, equivalently denoted by $L(G)$, is called the language generated by G. Two nonterminals A and B are *distinct* iff $L(A) \neq L(B)$; otherwise, they are *equivalent.* A language L is *context-free* if there exists a context-free grammar G such that $L=L(G)$.

Since we are concerned with the learning problem of context-free grammars, without loss of generality, we restrict our consideration to only λ-free context-free grammars.

A context-free grammar $G=(N, \Sigma, P, S)$ is in *Chomsky normal form* (abb., *CNF*) if each rule is of one of the following forms: $A \rightarrow BC$ (called *binary rule*), $A \rightarrow a$ (called *terminal rule*), where A, B and C are nonterminals and a is a terminal. It is known that any λ-free context-free language can be generated by some context-free grammar in CNF, that is, any λ-free context-free grammar can be transformed into an equivalent CNF. A context-free grammar $G=(N, \Sigma, P, S)$ is *k-restricted* if each rule is of the form either $A \rightarrow a$ or $A \rightarrow B_1 B_2 \cdots B_j$, where $A, B_j \in N (1 \leq \forall i \leq j)$, $a \in \Sigma$ and $2 \leq \exists j \leq k$. Note that any grammar in CNF is 2-restricted. A

CFG G is *reduced* if (1) for any $A, B \in N$, $L(A) \neq L(B)$, (2) for any $A \in N$, there are derivations such that $S \Rightarrow^* \alpha A \beta$ and $A \Rightarrow^* w$, where α, $\beta \in (N \cup \Sigma)^*$ and $w \in \Sigma^*$.

Proposition 1 (Ref. 5)) *For any λ-free context-free language L, there exists a context-free grammar G in CNF such that G is reduced and* $L = L(G)$. □

In what follows, we may assume that *any grammar G is a λ-free, reduced CFG.* Further, a derivation by the relation $\Rightarrow$ indicates the *left-most* one.

By $|N|+|P|$, we measure the *size* of the grammar, where $|N|(|P|)$ is the cardinarity of $N(P)$, and denoted by $size(G)$. G is *minimal* iff *size* (G) is minimum, that is, for any *CFG* G' that is equivalent to G, $size(G) \leq (G')$ holds.

2.2 MAT Learning

Let L be a target *CFL* over a fixed alphabet Σ. We assume the following types of queries in the learning process.

A *membership query* proposes a string $x \in \Sigma^*$ and asks whether $x \in L$ or not. The answer is either *yes* or *no.* An *equivalence query* proposes a grammar G and asks whether $L = L(G)$ or not. The answer is *yes* or *no,* and in the latter case together with a counterexample w in the symmetric difference of L and $L(G)$. A counterexample w is *positive* if it is in $L - L(G)$, and *negative* otherwise.

The learning protocol consisting of membership queries and equivalence queries is called *minimally adequate teacher* (*MAT*). The purpose of the learning is to find a *CFG* $G = (N, \Sigma, P, S)$ such that $L = L(G)$ with the help of minimally adequate teacher.

This definition is slightly different from the original one. In Ref. 3), equivalence queries are done only with conjectures of the same type as the target class. However, our model allows to query with arbitrary context-free grammars, that is, conjectured grammars constructed in the algorithm can be any (context-free) grammars. Equivalence queries modified as above are called *extended equivalence queries,* and the learning protocol consisting of membership queries and extended equivalence queries is called *extended minimally adequate teacher,* and is employed in Ref. 7) to learn the class of simple deterministic grammars. Hereafter, we assume that MAT means extended MAT.

In Ref. 2), Angluin employs a strong query called *nonterminal membership queries* which, given a string $x \in \Sigma^*$ and a nonterminal A of G with unknown set of rules, can ask whether $x \in L(A)$ or not, and the answer is *yes* or *no.* The primary issue here is how one can replace nonterminal membership queries by membership queries at the sacrifice of some sort of restriction on the class of grammars to be learned.

3. c-Deterministic Grammars and Languages

Let $G = (N, \Sigma, P, S)$ be a *CFG*. Then, a grammar G is *context-deterministic* (abb., *c-deterministic*) iff whenever the derivation $S \Rightarrow^* xAz$ exists, $L(A) \overset{=}{\lambda} x \backslash L(G) / z$ holds, where $x, z \in \Sigma^*$ and $A \in N$. A language L is *c-deterministic* iff there exists a *c*-deterministic grammar G such that $L = L(G)$. Note that for *any* context-free grammar $G' = (N', \Sigma, P', S)$, $L(A) \subseteq x \backslash L(G') / z$ holds whenever the derivation $S \Rightarrow^* xAz$ exists. For example, the language L generated by the grammar G introduced in Section 1 is c-deterministic.

Let $G = (N, \Sigma, P, S)$ be a context-free grammar. For any nonterminal $A \in N$, define the *context of* A, written as $Cont(A)$, by $Cont(A) = \{(x, z) \in \Sigma^* \times \Sigma^* | S \Rightarrow^* xAz\}$. Then, an alternative definition for c-deterministic grammar is that for any $A \in N$, for every pair $(x, z) \in Cont(A)$, $x \backslash L(G) z \overset{=}{\lambda} L(A)$ holds.

As an example, consider a context-free grammar $G' = (N', \Sigma, P', E)$, where

$N' = \{E\}$,

$\Sigma = \{+, *, (,), \boldsymbol{id}\}$,

$P' = \{E \to E + E | E * E | (E) | \boldsymbol{id}\}$

(As easily seen, G' is a grammar which generates all the arithmetic expressions with operators $+$ and $*$ [6].)

Transforming G' into CNF grammar, we obtain $G = (N, \Sigma, P, E)$, where

$N = \{E, X_1, X_2, X_3, X_4, X_5\}$

$P = \{E \to EX_1 | X_3X_4 | \boldsymbol{id},$

$X_1 \to X_2E,$

$X_2 \to + | *,$

$X_3 \to (,$

$X_4 \to EX_5,$

$X_5 \to) \}$,

It is easy to see that $L(G) = L(G') (= L)$. We claim that G is c-deterministic. By simple calculation, we get the following equations :

$L(X_1)=\{+w|w\in L\}\cup\{*\ w|w\in L\}$
$L(X_2)=\{+,\ *\}$
$L(X_3)=\{(\}$
$L(X_4)=\{w)|w\in L\}$
$L(X_5)=\{)\}$

Let $(x, z)\in Cont(x)$ and $X\in N$. To prove that $x\backslash L/z=L(X)$, it is sufficient to show that $x\backslash L/z\subseteq L(X)$, that is, for any string $y\in\Sigma^+$ such that $xyz\in L$, we have that $y\in L(X)$.

Consider the nonterminal X_1. By the form of the grammar, the context of X_1 is defined recursively as follows:

(1) (w, λ) is in $Cont(X_1)$.
(2) If (u, v) is in $Cont(X_1)$, then both of $(u, v+w)$, $(w+u, v)$ are in $Cont(X_1)$.
(3) If (u, v) is in $Cont(X_1)$, then both of $(u, v*w)$, $(w*u, v)$ are in $Cont(X_1)$.
(4) If (u, v) is in $Cont(X_1)$, then $((u, v))$ is in $Cont(X_1)$.
(5) Those and only those are in $Cont(X_1)$, where w is a string in L. Let $y\in\Sigma^+$ be any string such that $xyz\in L$ for some $(x, z)\in Cont(X_1)$. Then, (x, z) follows the above definition, and there exists a string $w\in L$ such that $x'w=x$ for some $x'\in\Sigma^*$ and $wy\in L$. This implies that $y\in\{+, *\}L=L(X_1)$, and therefore, $x\backslash L/z\underset{\lambda}{=}L(X_1)$ for all $(x, z)\in Cont(X_1)$. Similarly from the contexts of the nonterminals E, X_2, X_3, X_4 and X_5, which are also recursively defined, we see that for any pair of string (x, z), if $(x, z)\in Cont(X)$, $x\backslash L/z\underset{\lambda}{=}L(X)$ holds. Hence, G is a c-deterministic grammar and L is a c-deterministic language.

4. Main Results

4.1 Learning c-Deterministic Grammars

(1) Introducing the set new nonterminals

Given a *positive* counterexample w, define a set of new nonterminals $N(w)$ as the set of all the triples (x, y, z) such that $w=xyz$, where y is not empty, that is,

$$N(w)=\{(x, y, z)|x, z\in\Sigma^*, y\in\Sigma^+ \text{ and } w=xyz\}.$$

The following lemma obviously holds.

Lemma 2 *Let $G=(N, \Sigma, P, S)$ be a c-deterministic grammar such that $L=L(G)$, and $w\in L$. Then, for any nonterminal A that appears in the derivation $S\Rightarrow_G^* w$, there is a triple (x, y, z) in $N(w)$ such that $L(A)\underset{\lambda}{=}x\backslash L/z$.* □

(2) Constructing new candidate rules

From the set of nonterminals N together with the new ones $N(w)$ produced above, we newly construct the set of nonterminals $N=N\cup N(w)$. Then, the set of new candidate rules P_{new} is constructed as

$$P_{new}=\{A\to a|A\in N(w), a\in\Sigma\}\cup\{A\to BC|A, B, C\in N, \text{ at least one of } A, B, C \text{ is in } N(w)\}.$$

(3) Learning algorithm LA

The following is the learning algorithm LA for c-deterministic grammars:

Input: a c-deterministic *CFL* L over fixed Σ.
Output: a *CFG* G in CNF such that $L=L(G)$;
Procedure:
intialize $G=(\{S\}, \Sigma, \emptyset, S)$;
repeat
  make an equivalence query to $G=(N, \Sigma, P, S)$;
  if the answer in *yes* **then** output G and halts
  else if the answer is a *positive* counterexample w
  then introduce the set of new nonterminals $N(w)$ from w;
    $N:=N\cup N(w)$;
    construct the set of new rules P_{new};
    $P:=P\cup P_{new}$;
  else (the answer is a *negative* counterexample w')
    parse w' with conjectured grammar G;
    construct the derivation tree $T_{S,w'}$;
    diagnose P and find an incorrect rule r;
    $P:=P-\{r\}$

(4) Diagnosing the set of rules P

Before describing the diagnosing algorithm, we need some more definitions.

A derivation tree for a derivation $A\Rightarrow_G^* w$ is denoted by $T_{A,w}$. We call such a string w a *leaf string* of $T_{A,w}$.

For a target c-deterministic *CFL* L, let $G_*=(N_*, \Sigma, P_*, S)$ be a c-deterministic *CFG* such that $L=L(G_*)$, and $G=(N, \Sigma, P, S)$ be a conjectured grammar. For nonterminals $A=(x, y, z)$ in N and $\tilde{A}\in N_*$, we say that A is *well-corresponding to* $\tilde{A}$ iff $x\backslash L/z\underset{\lambda}{=}L(\tilde{A})$ holds. Note that, in this case, $A=(x, y, z)$ can be identified as $\tilde{A}$ and a nonterminal membership query to $\tilde{A}$ is replaced with a membership query using the relation $L(\tilde{A})\underset{\lambda}{=}x\backslash L/z$. A nonterminal A is *non-corresponding to* G_* iff there is no nonterminal $\tilde{A}\in N_*$ to which A is well-corresponding. A binary rule $A\to B_1B_2$ in P is

well-corresponding to $\tilde{A}\to\tilde{B}_1\tilde{B}_2$ in P_* iff A, B_1 and B_2 are well-corresponding to $\tilde{A}$, $\tilde{B}_1$ and $\tilde{B}_2$, respectively. Similarly, a terminal rule $A\to a$ in P is *well-corresponding to* $\tilde{A}\to a$ in P_* iff A is well-corresponding to $\tilde{A}$.

Let N be the set of nonterminals of a conjectured grammar $G=(N, \Sigma, P, S)$. A *replacement* σ is a construct of ordered pairs $[(w_1, B_1), (w_2, B_2)]$, where $w_i\in\Sigma^*$ and $B_i=(x_i, y_i, z_i)\in N$ for $i=1, 2$. Suppone $\beta=B_1B_2$. Then, an *instance* of β by σ, denoted by $\sigma[\beta]$, is a terminal string obtained by replacing B_i of β with a terminal string w_i for $i=1, 2$.

Let r be a binary rule $A\to B_1B_2$, where $A=(x, y, z)$. Then, r is *incorrect for* L iff there exists a replacement $\sigma=[(w_1, B_1), (w_2, B_2)]$ such that for $i=1, 2$, $w_i\in x_i\backslash L/z_i$, but $\sigma[B_1B_2]\notin x\backslash L/z$. A similar definition is given for a terminal rule, that is, a rule $A\to a$ is *incorrect* for L iff $a\notin x\backslash L/z$. A rule is *correct* for L iff it is not incorrect for L. These definitions are a natural extension of those in Ref. 2), 7). Note that the rule containing non-corresponding nonterminals may be incorrect for L or may not.

It is clear that every rule well-corresponding to a rule of G_* is correct for L. We show that if a string w' which is not in L is derived by the conjectured grammar G, then there exists at least one incorrect rule in P, that is, if P has no incorrect rule, then G derives no string w' such that $w'\notin L$.

Lemma 3 *Let $G=(N, \Sigma, P, S)$ be a conjectured grammar and L be a target c-deterministic language. If all the rules in P are correct for L, then for all $A=(x, y, z)\in N$, $L(A)\subseteq x\backslash L/z$ holds.*

Proof. Let w be a string such that $w\in L(A)$. We show that w is also in $x\backslash L/z$ by induction on the maximum length i of the path of derivation tree for w. Suppose $i=1$, that is, $A\Rightarrow w$. Then, $w=a$ must be derived using a terminal rule $A\to a$ correct for L, and therefore, $a\in x\backslash L/z$.

Next, suppose that the claim holds for $j\leq k$. Let T_{B_1, w_1}, T_{B_2, w_2} be two derivation trees, where $B_i=(x_i, y_i, z_i)\in N$ for $i=1, 2$, such that the maximum length of the path in each T_{B_i, w_i} is at most k. By the induction hypothesis, $w_i\in x_i\backslash L/z_i$. Let $A\to B_1B_2$ be a correct rule in P. Then, we obtain a derivation tree T_{A, w_1w_2} for w_1w_2, where the maximum length of the path is at most $k+1$. Suppose w $(=w_1w_2)$ is not a string in $x\backslash L/z$. Then, there exists a replacement $\sigma=[(B_1, w_1), (B_2, w_2)]$ such that for $i=1, 2$, $w_i\in x_i\backslash L/z_i$ but $\sigma[B_1B_2]=w_1w_2\notin x\backslash L/z$, from which we have that the rule $A\to B_1B_2$ is incorrect for L. This contradicts the assumption, and therefore, the lemma holds. □

Note that by replacing A of the above lemma by S, we have that if all the rules in P are correct for L, then $L(G)=L(S)\subseteq\lambda\backslash L/\lambda=L$.

Given a *negative* counterexample w', the algorithm has to prevent the conjectured grammar from producing w' by removing wrong rules. In order to determine such wrong rules, the algorithm calls the diagnosing procedure whenever a negative counterexample is provided.

The diagnosing algorithm is a modification of Shapiro's *contradiction backtracing procedure* (Ref. 8)). This procedure takes as an input a subtree $T_{A, w'}$, a derivation tree of w' rooted with A, and outputs a rule r which is incorrect for L.

The diagnosing algorithm in turn deals with each child of the root A of $T_{A, w'}$. If A has only one child, which must be labeled with a symbol a $(=w')\in\Sigma$, then the algorithm returns the terminal rule $r: A\to a$. Otherwise, that is, if A has two children, the algorithm proceeds as follows.

Assume that the left (resp. right) child of the root $A=(x, y, z)$ is labeled with $B_1=(x_1, y_1, z_1)$(resp. $B_2=(x_2, y_2, z_2)$), and w_i $(i=1, 2)$ is the leaf string of the subtree rooted by B_i. First, the algorithm queries with B_1 whether $x_1w_1z_1\in L$ or not. If the answer of the membership query is "no", then it recursively calls with the subtree T_{B_1, w_1}. Otherwise (the answer is "yes"), then it goes on the right child B_2 and performs it in the same manner. If the both queries are answered with "yes", then the diagnosing procedure returns the binary rule $A\to B_1B_2$ at the top of $T_{A, w'}$, which is incorrect for L. This will be discussed in more detail later.

Procedure *diag*$(T_{A, w'})$
  if $w'=a\in\Sigma$, **then** output $r: A\to a$ and halts
  else for $i=1, 2$ (each child of A), **do**
    make a membership query "$x_iw_iz_i\in L$?";
    if the answer is *no*
    then *diag*(T_{B_i, w_i});
  output $r=A\to B_1B_2$ and halts

We show an example of the run of the

diagnosing algorithm. Let $L=\{a^ncb^n|n\geq 0\}$ be the target language. Note that the grammar $G_*=(\{S, A, B, X\}, \{a, b, c\}, P_*, S)$, where P_* is composed of the rules

$S\rightarrow AX|c$
$A\rightarrow a$
$B\rightarrow b$
$X\rightarrow SB$,

is a c-deterministic grammar such that $L=L(G_*)$. Further, suppose that the current conjecture G has the rules

$\hat{S}\rightarrow\hat{A}\hat{X}|c$
$\hat{A}\rightarrow a$
$\hat{B}\rightarrow b$
$\hat{X}\rightarrow\hat{S}\hat{B}|\hat{S}\hat{A}$,

where nonterminals $\hat{A}$, $\hat{B}$ and $\hat{X}$ are the triples (a, a, cbb), $(aaacb, b, b)$ and $(aa, acbb, b)$, respectively.

Let $w'=aacab$ be a negative counterexample returned in the learning algorithm. Then, LA constructs the derivation tree $T_{\hat{S}, w'}$ using the conjectured grammar, which contains at least one incorrect rule. In **Fig. 1**, the derivation tree $T_{\hat{S}, w'}$ for this derivation is shown.

Since the node labeled with S has two children, the diagnosing algorithm first considers the subtree $T_{\hat{A}, a}$ and queries whether $a\cdot a\cdot cbb\in L$ or not. Since $aacbb\in L$, the answer "yes" is returned, and the algorithm considers the next subtree $T_{\hat{X}, acab}$ making the next membership query "$aa\cdot acab\cdot b\in L$?". Then, the answer is "no". Therefore, the diagnosing algorithm is recursively called for the subtree $T_{\hat{X}, acab}$. The algorithm considers the subtree $T_{\hat{S}, aca}$, where $\hat{S}$ is the label of the left child of $\hat{X}$, and queries whether $aca\in L$ or not. Again, the answer is "no", and the algorithm is called for $T_{\hat{S}, aca}$. Then, it eventually makes two membership queries "$a\cdot a\cdot cbb\in L$?", and "$aa\cdot ca\cdot b\in L$?", for two subtrees $T_{\hat{A}, a}$ and $T_{\hat{X}, ca}$. While the answer for the former is "yes", the answer for the latter query is "no". Therefore, the diagnosing algorithm is applied for the subtree $T_{\hat{X}, ca}$. At this time, two membership queries "$c\in L$?" and "$a\cdot a\cdot cbb\in L$?" are made, but both answers for the queries are "yes". Then, the diagnosing algorithm returns the rule $\hat{X}\rightarrow\hat{S}\hat{A}$ as an incorrect rule for L and halts.

Note. The diagnosing algorithm correctly works even for a derivation in which some non-corresponding nonterminals appear.

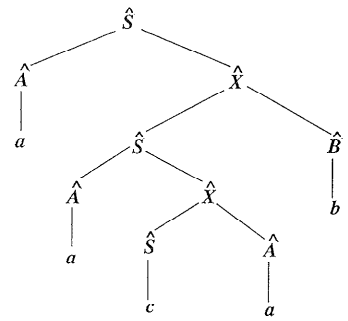

Fig. 1 A derivation tree for $w'=aacab$.

Although a rule has some non-corresponding nonterminals, the rule may not be used in any derivation of a negative example.

We are now in a position to prove the correctness of the diagnosing algorithm.

Lemma 4 *Given a negative counterexample w' and a conjectured grammar G, the diagnosing algorithm always halts and outputs a rule incorrect for L.*

Proof. It is clear that since the procedure *diag* is recursively called with a proper subtree of the initial input tree $T_{S, w'}$, the algorithm always halts and outputs some rule in P.

Let us assume that a rule $r: A\rightarrow\alpha$ ($\alpha\in N^2\cup\Sigma$) is returned by this algorithm,where $A=(x, y, z)$ and w is the leaf string of a derivation tree $T_{A, w}$. Note that $xwz\notin L$.

Suppose that r is a terminal rule $A\rightarrow a$. Then, it is returned with the answer "no" to the membership query "$xaz\in L$?", which implies that $xaz\notin L$. Thus, r is incorrect for L.

Now, suppose r is a binary rule, that is, $\alpha=B_1B_2\in N^2$. Let $B_i=(x_i, y_i, z_i)$ and the string w_i ($i=1, 2$) be the leaf string of a subtree rooted B_i. When $r: A\rightarrow B_1B_2$ is returned, from the property of *diag*, it must hold that $x_iw_iz_i\in L$ for $i=1, 2$. Hence, r is incorrect for L. □

Thus, it is seen that the diagnosing algorithm finds not a rule containing non-corresponding nonterminals but an incorrect rule, which may contain some non-corresponding nonterminals or may not.

4.2 The Correctness and Time Analysis of *LA*

Let $G_0=(N_0, \Sigma, P_0, S)$ be a minimal c-deterministic grammar such that $L=L(G_0)$ and l be the maximum length of all counterexamples provided through this algorithm.

Lemma 5 *Let $G_0=(N_0, \Sigma, P_0, S)$ be a minimal context-free grammar. Then, the number of positive examples needed to identify a correct grammar is at most $|N_0|$.*

Proof. Let $G=(N, \Sigma, P, S)$ be the conjectured grammar from LA, and suppose a positive counterexample w is given. Then, we claim that at least one new triple well-corresponding to some $\tilde{A}$ in N_0 is introduced.

For w, let $N_0(w)$ be the set composed of all the nonterminals in N_0 used in the derivation of G_0 for w. By the nature of LA, if a positive counterexample w is given, then there exist some rules or nonterminals needed to derive w but not contained in P or N. By Lemma 2, all triples well-corresponding to the nonterminals in N_0 that appear in the derivation $S \Rightarrow^{*}_{G_0} w$ are in $N(w)$. Further, for each new nonterminal A, all the rules containing A are added to the rule set P, and only incorrect rules in P are removed by the diagnosing procedure.

To sum up, whenever a positive counterexample w is given, there exists at least one nonterminal $A=(x, y, z)$ in $N(w)-N$ that is needed to derive w and is well-corresponding to $\tilde{A}$. That is, satisfying that $x\backslash L/z = L(\tilde{A})$ for some $\tilde{A} \in N_0$, and these are added to N. This justifies the claim, and therefore, the lemma holds. □

This means that after receiving at most $|N_0|$ positive counterexamples, N includes a sufficient number of nonterminals to generate the target language L.

For the number of nonterminals produced from each w, the following lemma is easily shown.

Lemma 6 *Let $w \in \Sigma^*$ be a positive counterexample given in the algorithm. Then, $|N(w)|$, that is, the number of nonterninals introduced is at most $(1/2)\lg(w)(\lg(w)-1)$.* □

From Lemma 6, the number of rules constructed from $N(w)$ is also bounded.

Lemma 7 *Let $G=(N, \Sigma, P, S)$ be the conjectured grammar, where N is the set of nonterminals incremented by adding new nonterminals. Then, the number of new rules P_{new} is at most $(|N|\times|\Sigma|)+(|N|)^3$.* □

Let $|N_{\max}|$ be the maximum number of nonterminals of conjectured grammars, and $|P_{\text{total}}|$ be the total number of rules introduced in LA. From Lemmas 5 and 6, the following two lemmas immediately hold.

Lemma 8 *Through running the learning algorithm, the maximum number $|N_{\max}|$ of nonterminals is at most $(1/2)|N_0|l(l-1)$.* □

Lemma 8 together with Lemma 7 immediately leads to the following.

Lemma 9 *The number $|P_{\text{total}}|$ of rules introduced in the whole of the learning algorithm is bounded by $((1/2)|N_0|l(l-1)\times|\Sigma|)+((1/2)|N_0|l(l-1))^3$.* □

Now, suppose a negative counterexample w' is given. Then, the algorithm constructs its derivation tree using the conjectured grammar $G=(N, \Sigma, P, S)$ at that time. Since there exists an algorithm (e.g., Cocke-Kasami-Younger algorithm[5]) that constructs the derivation tree for w' is time proportional to $(|N|+|P|)(\lg(w'))^3$, each parsing procedure requires at most $(|N_{\max}|+|P_{\text{total}}|)l^3$ steps, where $|N| \leq |N_{\max}|$, $|P| \leq |P_{\text{total}}|$ and $\lg(w') \leq l$. From these observations, the next lemma immediately follows;

Lemma 10 *When a negative counterexample is given, the time complexity of constructing its derivation tree is at most*

$$\left\{(|\Sigma|+1)\frac{1}{2}|N_0|l(l-1)+\left(\frac{1}{2}|N_0|l(l-1)\right)\right\}^3 l^3.$$ □

As for the number of membership queries performed in one calling of diagnosing algorithm, the next lemma holds.

Lemma 11 *In the diagnosing algorithm, the number of required membership queries is at most $2\lg(w')-1$, where w' is a negative counterexample given in the algorithm LA.*

Proof. It suffices to show by the induction on n $(=\lg(w'))$, the length of w', that the number of internal nodes of each derivation tree for w', using the grammar $G=(N, \Sigma, P, S)$, is $2\lg(w')-1$, because the membership queries are made for each nonterminal appearing in the derivation of w'. First, let $n=1$. Then, the claim holds, since the derivation tree has one internal node, namely, the root node. Next, assuming that the claim holds for all n less than or equal to k, suppose $S \to AB$ is a rulle in P and let $w_1' \in L(A)$, $w_2' \in L(B)$ be two strings such that $\lg(w_1')+\lg(w_2')=k+1$. Then, consider a derivation tree $T_{S, w'}$. By the induction hypothesis, both $\lg(w_1') \leq k$ and $\lg(w_2') \leq k$ holds. Therefore, the numbers of internal nodes of the two derivation trees $T_{A, w_1'}$ and $T_{B, w_2'}$ are $2\lg(w_1')-1$ and

$2\lg(w_2')-1$, respectively. Then, the number of internal nodes of the derivation tree $T_{S,w'}$ is $(2\lg(w_1')-1)+(2\lg(w_2')-1)+1=2(k+1)-1$. Thus, the claim holds for $n=k+1$.

This means that the number of membership queries required in the diagnosing algorithm for the negative counterexample w' is at most $2\lg(w')-1$. □

When $|N_0|$ positive counterexamples are given, the set of rules P_{total} includes all the rules that are well-corresponding to the ones in P_0, that is, any string in L is derivable using P_{total}. This implies that $L\subseteq L(G)$ for a conjectured grammar G at that time, and hence no more positive counterexample is given. By Lemma 4, each time a negative counterexample is given, one incorrect rule is determined and removed. Therefore, the number of required negative counterexamples is at most $|P_{\text{total}}|$, the maximum number of rules of conjectured grammar. By Lemma 3, if all incorrect rules are removed, the resulting conjectured grammar derives no string which is not in L, that is, $L(G)\subseteq L$. From these facts described above, we conclude that LA always converges and outputs a correct grammar G.

From a series of lemmas above, we obtain the main theorem;

Theorem 12 *For any c-deterministic language L, the learning algorithm LA identifies a CFG G such that $L=L(G)$ in time polynomial in the maximum length l of counterexamples and the size of a minimal grammar G_0 for L, where the size of the grammar G_0 is the sum of the numbers of nonterminals and rules.* □

5. Relationships between Other Subclasses of *CFG*s

In this section, we discuss the relationships between the class of c-deterministic languages and other subclasses of context-free languages which are learnable from MAT.

First, the class of regular languages is properly contained in the class of c-deterministic languages.

Lemma 13 *For any regular language L, there exists a c-deterministic grammar such that $L=L(G)$.*

Proof. The grammar corresponding to a minimal DFA which accepts L is obviously c-deterministic. □

Furthermore, there exists a non-regular language L' which is c-deterministic, such as the language L introduced in Section 3.

As mentioned previously, the class of simple deterministic languages and of one-counter languages is also polynomial-time learnable from MAT. We claim that the class of c-deterministic languages is incomparable to the class of simple deterministic languages and of one-counter languages. In order to show this, we need the next easy lemma.

Lemma 14 *Let L be a c-deterministic language. Then, there exists a constant $p_L>0$ such that for two strings $z=uvwxy$ and $z'=uv'w'x'y$ in L, if $\lg(z)\geq p_L$, $\lg(z')\geq p_L$ and for all $i\geq 0$, both uv^iwx^iy and $uv'^iw'x'^iy$ are in L, then $uv^iw'x^iy$ and $uv'^iwx'^iy$ are also in L for all $i\geq 0$.*

Proof. Let $G=(N, \Sigma, P, S)$ be a c-deterministic grammar such that $L=L(G)$. Then, by what is called "Pumping Lemma", there exists $p_L(>0)$ depending only on L and satisfying the conditions of the lemma. Let z, z' be two strings in L such that $z=uvwxy$, $z'=uv'w'z'y$ and both $\lg(z)\geq p_L$ and $\lg(z')\geq p_L$ hold. Assume that both $uv^iw'x^iy$ and $uv'^iwx'^iy$ are in L for all $i\geq 0$. Then, there exist the derivations

$$S\Rightarrow_G^* uAy,\ A\Rightarrow_G^* vAx,\ A\Rightarrow_G^* w,$$
$$S\Rightarrow_G^* uBy,\ B\Rightarrow_G^* v'Bx',\ B\Rightarrow_G^* w'$$

for z and z', where $A, B\in N$. Since G is c-deterministic $A=B$ holds. Hence, both

$$S\Rightarrow_G^* uAy\rightarrow_G^* uv'^iAx'^iy\Rightarrow uv'^iwx'^iy$$

and

$$S\Rightarrow_G^* uAy\Rightarrow_G^* uv^iAx^iy\Rightarrow^* uv^iw'x'y$$

hold for all $i\geq 0$. □

Example 1 Consider the two languages $L_0=\{a^mb^ncb^na^m|m\geq 1, n\geq 0\}$ and $\tilde{L}_0=\{a^md^ned^na^m|m\geq 1, n\geq 0\}$ over an alphabet $\Sigma=\{a, b, c, d, e\}$. Let $G_0=(\{S_0, A, B\}, \Sigma, P_0, S_0)$ and $\tilde{G}_0=(\{\tilde{S}_0, \tilde{A}, \tilde{B}\}, \Sigma, \tilde{P}_0, \tilde{S}_0)$, where

$$P_0: S_0\rightarrow aAa$$
$$A\rightarrow c|aAa|bBb$$
$$B\rightarrow c|bBb$$

and

$$\tilde{P}_0: \tilde{S}_0\rightarrow a\,\tilde{A}\,a$$
$$\tilde{A}\rightarrow e|a\,\tilde{A}\,a|d\,\tilde{B}\,d$$
$$\tilde{B}\rightarrow e|d\,\tilde{B}\,d.$$

Then, $L_0=L(G_0)$ and $\tilde{L}_0=L(\tilde{G}_0)$, and moreover, both G_0 and $\tilde{G}_0$ are c-deterministic.

Let $L_1=L_0\cup\tilde{L}_0$. Then, L_1 is not c-deterministic, because from Lemma 14, if it is

c-deterministic, the strings such as *addcdda* or *aabebaa* must also be in L_1, which contradicts the fact. □

Lemma 15 *The class of c-deterministic languages is incomparable to the class of simple deterministic languages.*

Proof. The language proposed in Section 3 (the language consists of all the arithmetic expressions with operators + and *) is a c-deterministic language, but not a simple deterministic language.

Conversely, we show that the language L_1 introduced in Example 1, which is not a c-deterministic language, it a simple deterministic language. Let $G_1'=(\{S, A, B, C, D, E, F\}, \Sigma, P_1', S)$ *be a context-free grammar, where* P_1' consists of the rules

$S \to aAB$
$A \to c|e|aAB|bCD|dEF$
$B \to a$
$C \to c|bCD$
$D \to b$
$E \to e|dEF$
$F \to d$.

It is easy to show that $L_1=L(G_1')$. Since G_1' is a simple deterministic grammar, L_1 is a simple deterministic language. □

A deterministic one-counter automaton M is described by a 6-tuple $(\Sigma, Q, q_0, F, \delta, \delta')$, where Q is a finite set of states, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. The two functions $\delta : Q \times \Sigma \times \{0, 1\} \to Q \times (Z \cup \{clear\})$ and $\delta' : Q \times \Sigma \times N \to Q \times N$ together define the transition function[4]. For example, $\delta(p, a, 1)=(q, +1)$ means that if the current state is p, the input is a and the stack (counter) is not empty, it changes its state to q and add 1 to the counter. The alternative function δ' is used instead of δ only when the value of the counter becomes negative after applying δ.

Lemma 16 *The class of c-deterministic languages is incomparable to the class of one-counter languages.*

Proof. The language L_0 introduced in Example 1 is a c-deterministic language, but it is not a one-counter language.

Consider a one-counter automaton $M=(\Sigma, Q, q_0, F, \delta, \delta')$, where $Q=\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$, $F=\{q_6, q_7\}$, and define δ as

$\delta(q_0, a, 0)=(q_1, 0)$ $\quad$ $\delta(q_4, a, 0)=(q_4, 0)$
$\delta(q_1, a, 0)=(q_1, 0)$ $\quad$ $\delta(q_5, d, 1)=(q_5, +1)$
$\delta(q_1, b, 0)=(q_2, +1)$ $\quad$ $\delta(q_5, e, 1)=(q_6, 0)$
$\delta(q_1, d, 0)=(q_3, +1)$ $\quad$ $\delta(q_6, d, 1)=(q_6, -1)$
$\delta(q_2, b, 1)=(q_2, +1)$ $\quad$ $\delta(q_6, a, 0)=(q_7, 0)$
$\delta(q_2, c, 1)=(q_3, 0)$ $\quad$ $\delta(q_7, a, 0)=(q_7, 0)$.
$\delta(q_3, b, 1)=(q_3, -1)$
$\delta(q_3, a, 0)=(q_4, 0)$

The language L_2 accepted by M is $L_2=\{a^m b^i c d^i a^n | i, m, n \geq 1\} \cup \{a^m d^i e d^i a^n | i, m, n \geq 1\}$. This is not a c-deterministic language, proved by Lemma 14. □

Finally, we show that the class of even linear languages is properly contained in the class of c-deterministic languages.

An even linear grammar (abb., *ELG*) is a linear grammar whose production rules are of the form either $A \to uBv$ such that $\lg(u)=\lg(v)$ or $A \to w$, where $A, B \in N$ and $u, v, w \in \Sigma^+$. (Recall the assumption that any grammar G considered in this article is λ-free.) It is known that the class of even linear languages properly contains the class of regular languages[1]. The Lemmas 17, 18 and 19 are shown in Ref. 9).

Lemma 17 (Ref. 9)) *Any even linear language is generated by an even linear grammar* $G=(N, \Sigma, P, S)$ *such that each rules in* P *is of the form*

$A \to a$, $A \to ab$, $A \to aBb$,

where $A, B \in N$ *and* $a, b \in \Sigma$. □

Let $G^0=(\{S^0\}, \Sigma, P^0, S^0)$ be an even linear grammar, where

$P^0=\{S^0 \to a | a \in \Sigma\}$
$\cup \{S^0 \to ab | a, b \in \Sigma\}$
$\cup \{S^0 \to aS^0b | a, b \in \Sigma\}$.

We call G^0 the *universal ELG* for Σ. Let C be a language on the labels of P^0, that is, C is a language composed of the strings of rule labels of G^0. Then, we define

$L_{[C]}(G^0)=\{w \in \Sigma^* | S \Rightarrow^{\alpha} w, \alpha \in C\}$,

where $S \Rightarrow^{\alpha} w$ denotes a derivation using the string of rules α. We call C a *control set.*

Lemma 18 (Ref. 9)) *For any even linear language* L, *there exists a regular control set* C *such that* $L=L_{[C]}(G^0)$. □

Lemma 19 (Ref. 9)) *Let* C *be a regular control set. Then,* $L=L_{[C]}(G^0)$ *is an even linear language.* □

From above results, we show that any even linear language is c-deterministic.

Lemma 20 *For any even linear grammar* $G=(N, \Sigma, P, S)$, *there exists an even linear grammar* $G'=(N', \Sigma, P', S)$ *such that* $L(G')=$

$L(G)$ and G' is c-deterministic.

Proof. Without loss of generality, we assume that L does not contain $\{\lambda\}$. Let C be the control set such that $L=L_{[C]}(G^0)=L(G)$. Since C is regular, there exists a DFA $M=(N, P^0, \delta, S, F)$, which accepts C. Note that the alphabet N is used as the set of states and S as the initial state. From M, we construct a grammar $G'=(N, \Sigma, P', S)$, where P' is defined as follows:

(1) For $p: S^0 \to a$ such that $\delta(A, p)\in F$, a rule $A \to a$ is added to P'.

(2) For $p: S^0 \to ab$ such that $\delta(A, P)\in F$, a rule $A \to ab$ is added to P'.

(3) For $p: S^0 \to aS^0b$ such that $\delta(A, p)=B$, a rule $A \to aBb$ is added to P'.

(4) P' is composed of only the rules defined above.

It is assured that $L=L(G')$ [9]. We show that G' is c-deterministic. By the determinicity of M, it is seen that if the two production rules $A \to aBb$ and $A \to aCb$ exist, then $B=C$ holds. This implies that if there exist two derivations $S \Rightarrow^* xAz$ and $S \Rightarrow^* xBz$ in G', then $A=B$ holds, and by the form of the grammar, $L(A)=x\backslash L/z$ holds for every $A\in N$. □

6. Conclusion

We have presented an algorithm that learns the class of c-deterministic languages, a subclass of the class of context-free languages, from minimally adequate teacher. This algorithm is based on the characterization results of nonterminals using derivatives of a target language. It has been shown that the algorithm learns a correct grammar in polynomial time from minimally adequate teacher, which gives a generalization of the corresponding result on regular languages in Ref. 3). It is noted that the definition of the minimally adequate teacher used in this paper is slightly different from the original one, proposed by Angluin[3]. In her problem setting, the class of conjectures consists of deterministic finite automata, and the target class is the class of regular languages, that is, the class of conjectures exactly represents the targeted class; however, we used the teacher which allows arbitrary context-free grammars in CNF as conjectures to learn the class of c-deterministic grammars. This kind of teacher is called *extended* minimally adequate teacher[7].

Finally, we have discussed the relationships between several classes of context-free languages and the class of c-deterministic languages. It was shown that the class of c-deterministic languages properly includes the class of regular languages and even the class of even linear languages, and is incomparable to the class of simple deterministic languages which is also learnable in polynomial time from extended minimally adequate teacher. The class of one-counter languages, which is learnable in polynomial-time from minimally adequate teacher (Ref. 4)), has been shown to also be incomparable to the class of c-deterministic languages.

As for the closure properties of c-deterministic languages, it is shown that the class is closed under union, intersection and homomorphism, while there still remains open whether or not the class of c-deterministic languages is closed under complement, concatenation, Kleene closure and intersection with regular languages.

We have considered the learning problem on the class of 2-restricted (i.e., in CNF) c-deterministic grammars, in order to discuss the essence of the problem. Note that given a fixed $k\geq 2$, the algorithm LA is casily extended for learning of the class of k-restricted c-deterministic grammars, by modifying the procedure of constructing new rules. With the fixed k, the time complexity of modified LA is still bounded by a polynomial of $|N_0|$ and l.

Although a little is known about the class of c-deterministic languages, the class is moderately powerful so that it may generate some of the well-known context-free languages such as the sentenses of if-then-else structure, arithmetic expressions, and so on.

Acknowledgements The authors would like to thank the referees for their helpful comments and suggestions which greatly improved the completeness of this paper. This work is supported in part by Grants-in-Aid for Scientific Research Nos. 02650261 and 03245104 from the Ministry of Education, Science and Culture, Japan.

References

1) Amar, V. and Putzolu, G.: Generalizations of Regular Events, *Inf. Control*, Vol. 8, pp. 56-63 (1965).

2) Angluin, D.: Learning k-bounded Context-free Grammars, Res. Rep. 557, Dept. of Comput. Sci.,

Yale University (1987).
3) Angluin, D.: Learning Regular Sets from Queries and Counterexamples, *Inf. Comput.*, Vol. 75, pp. 87-106 (1987).
4) Berman, P. and Roos, R.: Learning One-counter Languages in Polynomial Time, *28th IEEE Symp. on FOCS*, pp. 61-67 (1987).
5) Harrison, M. A.: *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA (1978).
6) Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA (1979).
7) Ishizaka, H.: Polynomial Time Learnability of Simple Deterministic Languages, *Machine Learning*, Vol. 5, pp. 151-164 (1990).
8) Shapiro, E.: Inductive Inference of Theories from Facts, Res. Rep. 192, Dept. of Comput. Sci., Yale University (1981).
9) Takada, Y.: Grammatical Inference for Even Linear Languages Based on Control Sets, *Inf. Process. Lett.*, Vol. 28, pp. 193-199 (1988).

(Received April 6, 1992)
(Accepted December 3, 1992)

Hiromi Shirakawa was born in Tokyo, Japan on Dec. 8, 1967. She received the B.B.A degree from Sanno College in 1990 and M.E. in University of Electro-Communications in 1992. Currently she is a teaching assistant of Sanno College Graduate School of Management and Informatics. Her research interest includes database theory and algorithmic learning theory.

Takashi Yokomori received the B.S., the M.S. and the D.S. degrees from the University of Tokyo, in 1974, 1976 and 1979, respectively. After working for the Department of Informatics, Sanno College, he joined the IIAS-SIS, Fujitsu Limited in 1983. He is currently with the Department of Computer Science and Information Mathematics, University of Electro-Communications. He was a Postdoctoral Fellow at McMaster University, Canada in 1981-1982, and at University of Pennsylvania, U.S.A. in 1982-1983. His research interests include formal language theory and computational learning theory. Dr. Yokomori is a member of the ISPJ, the IEICE, the JSAI and EATCS.