ICOT Technical Report: TR-465

TR-465

Learning Simple Deterministic Languages

by
H. Ishizaka

April, 1989

ICOT

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

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Learning Simple Deterministic Languages

Hiroki Ishizaka

ICOT Research Center
21F, Mita Kokusai Bldg.
1-4-28 Mita, Minato-ku, Tokyo, 108, Japan
C.Mail Address: ishizaka%icot.jp@relay.cs.net

Abstract

This paper is concerned with the problem of learning simple deterministic languages. The algorithm described in this paper is essentially based on the theory of model inference given by Shapiro. In our setting, however, nonterminal membership queries, for nonterminals except the start symbol, are not used. Instead of them, extended equivalence queries are used. Nonterminals that are necessary for a correct grammar and their meanings are introduced automatically.

We show an algorithm that, for any simple deterministic language L, outputs a grammar G in 2-standard form, such that $L = L(G)$, using membership and extended equivalence queries. We also show that the algorithm runs in time polynomial in the length of the longest counterexample and the minimum number of nonterminals of a correct grammar.

1 Introduction

We consider the problem of learning simple deterministic languages using membership and extended equivalence queries. A simple deterministic language (SDL) is characterized as the language that is accepted by a 1-state deterministic push-down automaton by empty store. The class of SDLs is a proper sub-class of deterministic languages. Another characterization of SDL is as the language that is generated by a context-free grammar in a special form of Greibach normal form, called a simple deterministic grammar (SDG).

Angluin [Ang87a] shows that the class of k-bounded context-free grammars is learnable in polynomial time using equivalence and nonterminal membership queries. The algorithm described in this paper is based on her algorithm. Both algorithms are essentially based on the theory of model inference given by Shapiro [Sha81]. Our setting, however, differs from Angluin's and Shapiro's in available types of queries, that is, in our setting, the learning algorithm is allowed to use membership queries but not nonterminal membership queries. This difference leads to a problem of introducing new nonterminals that are not observed in interaction between a teacher and a learner.

This problem relates to the problem of introducing theoretical terms in learning of first order theories from facts. Recently, there have been several approaches to the problem [Ban88, MB88]. However, in such settings that the algorithm learns not only a concept but also a language for describing the concept, it becomes difficult to ensure the convergence of a learning process. Of course, if the concept is described by a sufficiently restricted language, then we can expect an algorithm that learns the concept even in such a setting. The result of this paper gives one such learning algorithm.

Another feature of our setting is that the algorithm is allowed to use extended equivalence queries. An equivalence query defined in [Ang88] is allowed to propose only element of an original hypothesis space. For example, if the target class of learning is a set of concepts $\{L_1, L_2, \ldots\}$, then the learning algorithm must make each equivalence query with L_i in the set. We do not assume this restriction. Thus, the learning algorithm described in this paper makes each equivalence query proposing a grammar which is simply in 2-standard form but may not necessarily be simple deterministic.

Yokomori [Yok88] gives another algorithm for learning SDLs in polynomial time. Our setting also differs from his. The difference will be described in Section 4.

2 Preliminaries

We shall give some basic notions and the notation needed in this paper. Most of them are from [Ang87a] and [Yok88].

2.1 Context-free Grammars and Languages

An *alphabet* is a finite non-empty set of distinct symbols. For a given alphabet X, the set of all finite strings of symbols from X is denoted X^*. The empty string is denoted ε.

X^+ denotes the set $X^* - \{\varepsilon\}$. For a string x, $|x|$ denotes the length of x. If S is a finite set, then $|S|$ denotes the cardinality of S.

Let Σ be an alphabet. A *language* L over Σ is a subset of Σ^*. For a string x in Σ^* and a language L over Σ, let $\overline{x}L = \{y \mid xy \in L\}$ ($L\overline{x} = \{y \mid yx \in L\}$). The set $\overline{x}L$ ($L\overline{x}$) is called the *left(right)-derivative of L with respect to x*. For a string $x = a_1 a_2 \cdots a_n$, $Pre_i(x)$ denotes the string $a_1 a_2 \cdots a_i$, and $Suf_i(x)$ denotes the string $a_{i+1} a_{i+2} \cdots a_n$.

A *context-free grammar* (CFG) is a 4-tuple $G = (N, \Sigma, P, S)$, where N is an alphabet of *nonterminals*, Σ is an alphabet of *terminals* such that $N \cap \Sigma = \emptyset$, $S \in N$ is the *start symbol*, and P is a finite set of *production rules* of the form $A \to \alpha$ ($A \in N, \alpha \in (N \cup \Sigma)^*$). The *size* of a grammar G is the sum of $|N|$, $|\Sigma|$, $|P|$, and the sum of the lengths of the right-hand sides of all the productions in P.

For $\beta, \gamma \in (N \cup \Sigma)^*$, a binary relation $\Rightarrow$ is defined as follows: $\beta \Rightarrow \gamma$ if and only if there exist $\delta_1, \delta_2 \in (N \cup \Sigma)^*$ and there exists a production rule $A \to \alpha \in P$ such that $\beta = \delta_1 A \delta_2$ and $\gamma = \delta_1 \alpha \delta_2$. A *derivation from β to γ* is a finite sequence of strings $\beta = \beta_0, \beta_1, \cdots, \beta_n = \gamma$ such that, for each i, $\beta_i \Rightarrow \beta_{i+1}$. If there is a derivation from β to γ, then we denote $\beta \Rightarrow^* \gamma$, that is, the relation $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$. In each step of the derivation, if the left-most occurrence of a nonterminal in β_i is replaced, then it is called the *left-most derivation*. In what follows, unless otherwise stated, a derivation $\beta \Rightarrow^* \gamma$ always means the left-most one.

The language of a nonterminal A, denoted $L(A)$, is the set of all $x \in \Sigma^*$ such that $A \Rightarrow^* x$. Similarly, for $\alpha \in N^*$, $L(\alpha)$ denotes the set of all $x \in \Sigma^*$ such that $\alpha \Rightarrow^* x$. (To emphasize the grammar being used, we use the subscript G, e.g., $S \Rightarrow_G x$ or $L_G(A)$.) The *language* of the grammar G, denoted $L(G)$, is just $L(S)$, where S is the start symbol in grammar G.

2.2 SDG and SDL

A context-free grammar in Greibach normal form G is *simple deterministic* if, for any $A \in N, a \in \Sigma, \alpha, \beta \in N^*$, there exist productions $A \to a\alpha$ and $A \to a\beta$ in P, then $\alpha = \beta$. Note that the definition does not imply that SDGs generate only ε-free languages. In this paper, however, our attention focuses on only ε-free SDGs. A language L is *simple deterministic* if there exists an SDG G such that $L(G) = L$.

For example, the grammar $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, where

$$P = \{S \to aA, A \to b, A \to aB, B \to aBC, B \to bC, C \to b\},$$

is one of the SDGs that generate an SDL $\{a^m b^m | 1 \leq m\}$.

The following propositions provide the features of SDGs and SDLs desired for our purpose (see, e.g., [Har79]).

Proposition 1 *Let $G = (N, \Sigma, P, S)$ be an SDG. For any $A \in N, x \in \Sigma^+$ and $\alpha \in N^*$, if there exists a derivation $A \Rightarrow^* x\alpha$, then $L(\alpha) = \overline{x}L(A)$.*

Proposition 2 *Let $G = (N, \Sigma, P, S)$ be an SDG. For any $A \in N$, $L(A)$ is prefix-free, that is, if $x \in L(A)$, then, for any $y \in \Sigma^+$, $xy \notin L(A)$.*

Proposition 3 *For any SDG G, there exists an equivalent SDG G' that is in 2-standard form, i.e., there exists an SDG $G' = (N', \Sigma, P', S)$ such that*

(1) $L(G) = L(G')$;

(2) *Each production in P' is of one of the following forms: $A \to a$, $A \to aB$, $A \to aBC$, where $A, B, C \in N', a \in \Sigma$.*

Proposition 3 allows us to consider only (ε-free) context-free grammars in 2-standard form.

2.3 Models and Incorrectness/Correctness

Our algorithm for learning SDL is based on Shapiro's model inference algorithm [Sha81] and Angluin's learning algorithm [Ang87a]. The most important component of these algorithms is the diagnosis routine. The diagnosis routine finds an incorrect element in a hypothesis that implies a negative example, so we need to define notions for incorrectness (or correctness) of the elements of a grammar, that is, incorrectness of productions. In order to do this, we shall introduce some model theoretical notions for grammars.

Let $G = (N, \Sigma, P, S)$ be a context-free grammar. For each nonterminal $A \in N$, a *model* of A, denoted $M(A)$, is a subset of Σ^+. A *model* M for the grammar G consists of the model of each nonterminal.

$$M = \{M(A_1), M(A_2), \ldots, M(A_{|N|})\}.$$

A *replacement* is a finite tuple $\langle (y_1, A_1), \ldots, (y_n, A_n) \rangle$, where $y_i \in \Sigma^*, A_i \in N$. Let $\rho = \langle (y_1, A_1), \ldots, (y_n, A_n) \rangle$ and $\beta \in (N \cup \Sigma)^*$. ρ is *compatible* with β if and only if there are finite strings $x_0, \ldots, x_n \in \Sigma^*$ such that $\beta = x_0 A_1 x_1 A_2 \cdots A_n x_n$. If ρ is compatible with β, then an *instance* of β by ρ, denoted $\rho[\beta]$, is the terminal string obtained from β by replacing each occurrence of A_i in β by the terminal string y_i.

Let M be a model for a grammar G. A production $A \to \alpha$ is *incorrect* for M if and only if there exists a replacement $\rho = \langle (y_1, A_1), \ldots, (y_n, A_n) \rangle$ that is compatible with α such that, for each i, $y_i \in M(A_i)$, but $\rho[\alpha] \notin M(A)$. A production is *correct* for M if and only if it is not incorrect for M.

Proposition 4 *Let $G = (N, \Sigma, P, S)$ be a CFG. Suppose a model M for G such that, for each nonterminal $A \in N$, $M(A) = L(A)$. Then every production in P is correct for M.*

2.4 Types of Queries

Let L be the unknown SDL that is intended to be learned by a learning algorithm. We assume a teacher who knows L and can answer the queries below. The algorithm is allowed to make two types of queries as follows.

A *membership query* proposes a string $x \in \Sigma^+$ and asks whether $x \in L$ or not. The reply is either *yes* or *no*.

An *extended equivalence query* proposes a grammar G in 2-standard form and asks whether $L = L(G)$. The reply is *yes* or *no*. If it is *no*, then a *counterexample* is also provided. A counterexample is a string x in the symmetric difference of L and $L(G)$. If $x \in L - L(G)$, x is called a *positive* counterexample, and if $x \in L(G) - L$, x is called a *negative* counterexample. The choice of a counterexample is assumed to be arbitrary.

Note the difference between extended equivalence queries and equivalence queries defined in [Ang88]. The equivalence query is allowed to propose only element of the original hypothesis space. Thus, in learning SDLs, the hypothesis proposed by an equivalence query is restricted to a grammar that generates an SDL. However, the hypothesis proposed by an extended equivalence query does not have to exactly generate an SDL. A teacher who answers equivalence queries and membership queries is called a *minimally adequate Teacher* [Ang87b], so we call a teacher who answers extended equivalence queries and membership queries an *extended minimally adequate Teacher*.

3 The Learning Algorithm

In what follows, unless otherwise stated, a grammar is in 2-standard form. Let L be the unknown SDL which should be learned by the algorithm and $G_0 = (N_0, \Sigma, P_0, S)$ be an SDG such that $L(G_0) = L$ and with the minimum number of nonterminals, that is, for any SDG $G' = (N', \Sigma, P', S')$ such that $L(G') = L$, $|N_0| \leq |N'|$. We assume that the terminal alphabet Σ and start symbol S are known to the learning algorithm, but that $N - \{S\}$, the set of nonterminals except S, and P, the set of productions, are unknown.

The main result of this paper is as follows.

Theorem 5 *There is an algorithm that, for any SDL L, learns a grammar G in 2-standard form such that $L(G) = L$ using extended equivalence queries and membership queries that runs in time polynomial in $|N_0|$ and the length of the longest counterexample.*

Note that the grammar learned by the algorithm may not be SDG. The grammar is simply in 2-standard form.

3.1 An Outline of the Algorithm

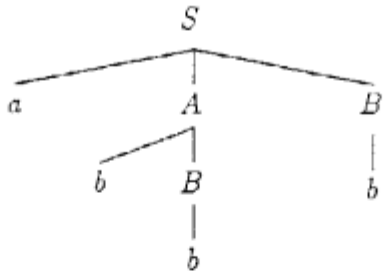
First, the algorithm initializes nonterminals N to $\{S\}$, and initializes productions P to the set of all productions consisting of only S. As a model M for G, we initially consider $\{M(S) = L\}$. Models for any other nonterminals that are introduced by the algorithm are defined in the next section. Then it iterates the following loop. An equivalence query is made, proposing G. If the reply is *yes*, then the algorithm outputs G and halts. Otherwise, a counterexample w is returned. The algorithm tries to parse w on G. If w can be parsed, that is, when w is negative, the algorithm diagnoses G on the parse-tree of w and finds an incorrect production for M. The incorrect production is removed from P. Otherwise, that is, when w is positive, new nonterminals are introduced and all new productions constructed from them are added to P.

The Learning Algorithm

Given: An extended minimally adequate Teacher for L and a terminal alphabet Σ.
Output: A grammar $G = (N, \Sigma, P, S)$ in 2-standard form such that $L(G) = L$.
Procedure:
$N := \{S\}$. $P := \{S \to aSS, S \to aS, S \to a | a \in \Sigma\}$. $G := (N, \Sigma, P, S)$.
repeat
  Make an extended equivalence query with G.
  If the reply is positive counterexample, *then*
    introduce new nonterminals with their models.
    Put all candidate productions into P.
  Else if the reply is negative counterexample, *then*
    diagnose G.
    Remove the incorrect production replied by the diagnosis routine from P.
until the reply is *yes*.
Output G.

We assume a parsing sub-procedure that runs in time polynomial in the size of a grammar G and w, e.g., Angluin's parsing procedure [Ang87a][1].

The diagnosis routine[2] finds an incorrect production for M on the input parse-tree that generates a string w and has its root node A such that $w \notin M(A)$. For example, consider the following parse-tree for a negative counterexample $abbb$.



Initially, $abbb \notin M(S) = L$ is known. Then the diagnosis routine considers in turn each child of S which is labeled with a nonterminal. In the example, first, the child labeled with A and generating the string bb is considered. The diagnosis routine inquires whether $bb \in M(A)$ or not. If $bb \notin M(A)$, then it calls itself recursively with the sub-tree rooted A. If $bb \in M(A)$, then it goes to the next child labeled with B and makes the same inquiry. If $b \notin M(B)$, then it returns the production $B \to b$. Otherwise, it returns the production $S \to aAB$.

In [Ang87a], such an inquiry is made through nonterminal membership queries. In

[1]Since G is in 2-standard form, Lemma 3 and Lemma 4 in [Ang87a] hold. In fact, the procedure returns a parse-DAG (directed acyclic graph) instead of a parse-tree. Our discussion, however, is not affected by the difference.

[2]See [Ang87a] for the precise definition of the diagnosis routine

our approach, however, it is made by using only membership queries. The next section shows how to introduce new nonterminals and replace nonterminal membership queries by membership queries.

Suppose that New is a set of all nonterminals that are newly introduced at a stage of learning. N is set to $N \cup New$. Let P_{New} be a set of all productions in 2-standard form constructed from N that have never appeared in P, that is, for each $a \in \Sigma$, P_{New} contains productions $A \to a\alpha$ such that $A\alpha \in N^+$, $|\alpha| \leq 2$ and $A\alpha$ contains at least one element of New. Then P is set to $P \cup P_{New}$.

Claim 1 *The set P_{New} is easily computed in time polynomial in $|N|$ and, at any stage of learning, P contains at most $|N| \times |\Sigma| \times (|N|+1)^2$ productions.*

3.2 Generating Nonterminals with Their Models

The key idea of the generating nonterminals routine has its roots in an extension of a model described in [Ish89].

First, we show an important feature of SDGs for describing the generating nonterminals routine.

Lemma 6 *Let $G = (N, \Sigma, P, S)$ be an SDG. Suppose that $A \Rightarrow^* rB\alpha$ for $A, B \in N, \alpha \in N^*, r \in \Sigma^+$, and that t is a string in $L(\alpha)$ such that $Suf_j(t) \notin L(\alpha)$ for any j $(1 \leq j \leq |t|-1)$ (if $\alpha = \varepsilon$ then $t = \varepsilon$). Then, for any $x \in \Sigma^+$, $x \in L(B)$ if and only if* (i) *$rxt \in L(A)$ and* (ii) *$rPre_i(x)t \notin L(A)$ for any i $(1 \leq i \leq |x|-1)$.*

Proof: Suppose $x \in L(B)$. Then $A \Rightarrow^* rB\alpha \Rightarrow^* rx\alpha \Rightarrow^* rxt$. Thus, $rxt \in L(A)$. Since $L(B)$ is prefix-free, $Pre_i(x) \notin L(B)$ for any i $(1 \leq i \leq |x|-1)$. Hence, if $rPre_i(x)t \in L(A)$, that is, $Pre_i(x)t \in \overline{r}L(A) = L(B\alpha)$, then there exists j $(1 \leq j \leq |t|-1)$ such that $Pre_i(x)Pre_j(t) \in L(B)$ and $Suf_j(t) \in L(\alpha)$. This contradicts the fact that $Suf_j(t) \notin L(\alpha)$ for any j $(1 \leq j \leq |t|-1)$. Thus, $rPre_i(x)t \notin L(A)$ for any i $(1 \leq i \leq |x|-1)$.

Conversely, assume that (i) and (ii) hold. From assumption (i), it holds that $xt \in \overline{r}L(A) = L(B\alpha)$. Since there is no strict suffix of t in $L(\alpha)$, there exists j $(1 \leq j \leq |x|)$ such that $Pre_j(x) \in L(B)$ and $Suf_j(x)t \in L(\alpha)$. On the other hand, from assumption (ii), $Pre_i(x)t \notin L(B\alpha)$ for any i $(1 \leq i \leq |x|-1)$. Hence, for any i $(1 \leq i \leq |x|-1)$, $Pre_i(x) \notin L(B)$. Thus, $j = |x|$. This concludes that $Pre_{|x|}(x) = x \in L(B)$. □

In the learning algorithm, whenever new nonterminals are introduced, there is a positive counterexample w. The generating nonterminals routine constructs nonterminals with their appropriate models from w.

Let w be a positive counterexample such that $|w| \geq 2$. *Nonterminals generated from a positive counterexample* w, denoted $N(w)$, are defined as follows:

$$N(w) = \{(r, s, t) | r, s \in \Sigma^+, t \in \Sigma^* \text{ and } rst = w\}.$$

Claim 2 *There are at most $|w|(|w|-1)/2$ elements of $N(w)$, and $N(w)$ is easily computed in time polynomial in $|w|$.*

For example, let $w = aabb$, then

$$N(w) = \{(a, abb, \varepsilon), (a, ab, b), (a, a, bb), (aa, bb, \varepsilon), (aa, b, b), (aab, b, \varepsilon)\}$$

For each triple $(r, s, t) \in N(w)$, let $\varphi(r, s, t)$ be the shortest suffix of t in $\overline{rs}L$, i.e.,

$$\varphi(r, s, t) = Suf_i(t) \quad \text{where} \quad i = \max_{0 \le j \le |t|-1} \{j \mid Suf_j(t) \in \overline{rs}L\}.$$

Claim 3 *The string $\varphi(r, s, t)$ is computed by making $|t|$ membership queries proposing $rsSuf_j(t)$ $(0 \le j \le |t| - 1)$.*

The intended model of each nonterminal in $N(w)$ is defined as follows. For each triple $(r, s, t) \in N(w)$, define

$$M((r, s, t)) = \{x \in \Sigma^+ \mid rx\varphi(r, s, t) \in L \text{ and } rPre_i(x)\varphi(r, s, t) \notin L \text{ for any } i\ (1 \le i \le |x| - 1)\}.$$

Lemma 7 *Let N be the set of known nonterminals. Suppose that w is a new positive counterexample. Then the time required for generating nonterminals and computing new productions is bounded by a non-decreasing polynomial in $|N|$ and $|w|$.*

Proof: By Claim 1, 2 and 3, it is straightforwardly implied. □

Lemma 8 *Let L be an SDL, w be a string in L, and $G = (N, \Sigma, P, S)$ be an SDG such that $L(G) = L$. For any $A \in N - \{S\}$ that appears in the derivation $S \Rightarrow^* w$, there exists a nonterminal $(r, s, t) \in N(w)$ such that $L(A) = M((r, s, t))$.*

Proof: Suppose that $S \Rightarrow^* rA\alpha \Rightarrow^* rs\alpha \Rightarrow^* rst = w$. Then, from the definition of $N(w)$, the triple (r, s, t) is in $N(w)$. (Since G is an SDG and $A \neq S$, neither r nor s is ε.) Since $L(S) = L(G) = L$, by Proposition 1, $L(\alpha) = \overline{rs}L(S) = \overline{rs}L$. By the definition of $\varphi(r, s, t)$, $\varphi(r, s, t) \in L(\alpha)$ and $Suf_j(\varphi(r, s, t)) \notin L(\alpha)$ for any j $(1 \le j \le |\varphi(r, s, t)| - 1)$. Hence, by Lemma 6 and the definition of $M((r, s, t))$, $L(A) = M((r, s, t))$. □

The above lemma ensures that if the learning algorithm is given a positive counterexample w, then it can make all nonterminals with appropriate models that are necessary for generating w. In the result, nonterminal membership queries used in [Ang87a] can be replaced by membership queries. For any $x \in \Sigma^*$ and $A \in N(w)$, the diagnosis routine can accomplish each inquiry as to whether $x \in M(A)$ or not by making $|x|$ membership queries.

3.3 Correctness and Complexity

In what follows, let L be the target language, $G_0 = (N_0, \Sigma, P_0, S)$ be an SDG such that $L(G_0) = L$ and with the minimum number of nonterminals, $G = (N, \Sigma, P, S)$ be the grammar in the algorithm and $M = \{M(S), M((r_1, s_1, t_1)), \ldots, M((r_{|N|-1}, s_{|N|-1}, t_{|N|-1}))\}$ be the model for G defined in the previous section.

Lemma 9 *Suppose that the diagnosis routine is given as input a parse-tree that generates a string x and has its root node labeled with $A \in N$ such that $x \notin M(A)$. Then it returns a production in P that is incorrect for M.*

Proof: The lemma can be proved along the same line of argument as the proof of Lemma 5 in [Ang87a]. □

Lemma 10 *The time required by the diagnosis routine on an input parse-tree for a negative counterexample w is bounded by a non-decreasing polynomial in $|w|$ and ℓ, the length of the longest counterexample.*

Proof: Since G is in 2-standard form, there are at most $|w|$ occurrences of nonterminals in the parse-tree. Thus, the number of inquiries made by the diagnosis routine is at most $|w|$. For each inquiry as to whether $x \in M(A)$ or not, if $A = S$, then only one membership query "$x \in L$?" is made. Otherwise, that is, if $A = (r, s, t)$, the routine makes at most $|x|$ membership queries "$rPre_i(x)\varphi(r, s, t) \in L$?" for $1 \leq i \leq |x|$. Since x is a sub-string of w, the total number of queries made in a diagnosing process is at most $|w|^2$. Since the main operations made in the diagnosis routine are making strings $rPre_i(x)\varphi(r, s, t)$ and making membership queries, it is clear that the claim of the lemma holds. □

Lemma 11 *The total number of given positive counterexamples is bounded by $|N_0|$.*

Proof: Let w_n be the nth given positive counterexample. We define $N_0(w_n)$ and $P_0(w_n)$ as follows:

$$N_0(w_n) = \{A \in N_0 \mid S \Rightarrow^*_{G_0} uA\alpha \Rightarrow^*_{G_0} w_n\},$$
$$P_0(w_n) = \{A \to a\alpha \in P_0 \mid a \in \Sigma, A\alpha \in (\bigcup_{i=1}^{n} N_0(w_i))^+\}.$$

When w_n is given, the learning algorithm computes $N(w_n)$ and sets N to $N \cup N(w_n)$. Then it computes all new candidate productions and adds them to P as described in Section 3.1.

By Lemma 8, for each nonterminal $A \in N_0(w_n)$, there exists a nonterminal $A' \in N(w_n)$ such that $L(A) = M(A')$. Under this correspondence of A and A', for every production in $P_0(w_n)$, a corresponding production is added to P at least once. By Proposition 4, these corresponding productions are correct for M. Since correct productions are never removed from P, whenever the $n+1$st positive counterexample is given, there exists at least one nonterminal $A \in N_0$ such that

$$A \in N_0(w_{n+1}) \quad \text{and} \quad A \notin \bigcup_{i=1}^{n} N_0(w_i).$$

Thus, the number of given positive counterexamples is at most $|N_0|$. □

Lemma 12 *The number of nonterminals introduced by the learning algorithm is bounded by $|N_0|\ell_p(\ell_p - 1)/2$, where ℓ_p is the length of the longest positive counterexample.*

Proof: For each positive counterexample w_i, $|N(w_i)|$ is at most $|w_i|(|w_i|-1)/2$ as stated in the previous section. By Lemma 11, the total number of nonterminals introduced by the algorithm is bounded by $|N_0|\ell_p(\ell_p-1)/2$. □

Proof of Theorem 5: From the way to introduce new productions and Lemma 12, the total number m of productions introduced into P is at most

$$m = \frac{|N_0|\ell_p(\ell_p-1)}{2} \times |\Sigma| \times \left(\frac{|N_0|\ell_p(\ell_p-1)}{2}+1\right)^2.$$

By Lemma 9, for each given negative counterexample, at least one incorrect production is found and it is removed from P. With Lemma 11, this implies that, after giving at most $|N_0|$ positive counterexamples and at most m negative ones, the learning algorithm outputs a grammar G such that $L(G) = L$.

Let ℓ be the length of the longest counterexample. By Lemma 12, at any stage of learning, the size of G is bounded by a non-decreasing polynomial in $|N_0|$ and ℓ. From the assumption on the parsing sub-procedure, the algorithm can determine whether a given counterexample is positive or negative in time polynomial in $|N_0|$ and ℓ. The total number of given counterexamples is at most $|N_0| + m$. With Lemma 7 and Lemma 10, this implies the claim for the complexity of the algorithm. □

4 Conclusion

We have discussed the problem of learning SDLs. The main idea presented in this paper was how to introduce necessary nonterminals with their appropriate models (or meanings). The problem introducing new sub-concepts that are useful for representing a target concept but not observed is one of the most important and difficult problems in machine learning. Recently, there have been several approaches to this problem, e.g., [Ban88, MB88]. It seems, however, that we have no sufficient result yet. Of course, our result is also for a class that is too restricted, so we need to try to find more general and useful approaches to the problem.

For the given algorithm, there is a problem of efficiency. As shown in the proof of Theorem 5, it is ensured that the algorithm runs in time polynomial in $|N_0|$ and ℓ. However, the polynomial has a rather high degree. If we can set each intermediate hypothetical grammar to an SDG, we will be able to decrease the degree. Of course, such a restriction on hypothetical grammars also results in the development of an algorithm that produces an SDG as its output using pure equivalence queries.

Yokomori [Yok88] gives another algorithm for learning SDLs in polynomial time. His algorithm exactly constructs an SDG. In his setting, however, a very powerful teacher is assumed. The teacher can answer the special two types of queries: prefix membership queries and derivatives equivalence queries. A prefix membership query is an extension of the membership query. A derivatives equivalence query proposes two pairs of strings $(u_1, w_1), (u_2, w_2)$ and asks whether $\overline{u_1}L\overline{w_1} = \overline{u_2}L\overline{w_2}$, where L is the target language.

It is clear that derivatives equivalence queries can be used, in our algorithm, to test whether two candidate nonterminals are identical. For example, for two nonterminals (u_1, v_1, w_1) and (u_2, v_2, w_2), if $\overline{u_1}L\overline{w_1} = \overline{u_2}L\overline{w_2}$, then they are identical. Thus, the number of nonterminals generated by our algorithm will be reduced. Unfortunately, we have not sufficiently discussed the problem of such relations between the ability of the teacher and the efficiency of the algorithm.

5 Acknowledgements

I wish to thank Dr. K. Furukawa and Dr. R. Hasegawa for their continuous support and advice. I am also deeply grateful to Dr. T. Yokomori for his many valuable comments. Mr. Y. Sakakibara pointed out my misunderstanding about equivalence queries in the draft. Discussions with the members of the Learning and Non-monotonic Reasoning Research Group at ICOT have also been very fruitful. Finally, I wish to thank Dr. K. Fuchi, the director of ICOT Research Center, for providing the opportunity to conduct this research in the Fifth Generation Computer Systems Project.

References

[Ang87a] Dana Angluin. Learning k-bounded context-free grammars. Research Report 557, Yale University Computer Science Dept., 1987.

[Ang87b] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Ban88] Ranan B. Banerji. Learning theories in a subset of a polyadic logic. In *Proc. Computational Learning Theory '88*, pp. 281–295, 1988.

[Har79] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1979.

[Ish89] Hiroki Ishizaka. Inductive inference of regular languages based on model inference. To appear in IJCM, 1989.

[MB88] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, pp. 339–352, 1988.

[Sha81] Ehud Y. Shapiro. Inductive inference of theories from facts. Technical Report 192, Yale University Computer Science Dept., 1981.

[Yok88] Takashi Yokomori. Learning simple languages in polynomial time. In *Proc. of SIG-FAI*, pp. 21–30. Japanese Society for Artificial Intelligence, June 1988.