Inferring Parenthesis Linear Grammars Based on Control Sets

YUJI TAKADA*

We provide a new grammatical inference method for a class of linear grammars, called parenthesis linear grammars. We first show that any linear language can be generated by a fixed linear grammar together with a regular control set. From this, it is shown that the inference problem for parenthesis linear grammars can be reduced to the problem for regular sets. This implies that to infer parenthesis linear grammars, we can use any inference algorithm for regular sets. An application of our results to the inference problem for generalized sequenctial machines is also discussed.

1. Introduction

Formal language theory is applied to the modeling and description of various concepts in many areas, for example, in artificial intelligence, in software engineering, and so on. The grammatical inference problem is a central issue in these areas, because the study leads us to basic methods for machine learning and automatic program synthesis. However, as far as efficient and practical methods are concerned, they have been provided only for the class of regular sets. In this paper, we consider the grammatical inference problem for linear grammars.

Biermann [4] and Tanatsugu [13] have presented inference methods for linear languages, which construct a linear grammar by finding self-embedding property of it from given strings. Both of them use some given parameters for bounding computation of finding the self-embedding substrings in a linear language. The greater values of parameters are, the less efficient procedures for finding self-embedding substrings are. Thus, the methods based on the self-embedding property of languages are not so efficient.

We provide a new grammatical inference method for a class of linear grammars, called parenthesis linear grammars. The class of parenthesis linear languages, in principle, contains the class of regular sets and is properly contained in the class of linear languages. We first show that there exists a fixed linear grammar over an alphabet by which any linear language over the alphabet is generated together with a regular control set (Theorem 3.2). In this case, self-embedding variables of a linear grammar correspond to the states on loops in the transition diagram of the finite state automaton which accepts the regular control set. From this, we next show that the inference problem for parenthesis linear grammars can be reduced to the problem for regular sets. This implies that, using any inference algorithm for regular sets, we can construct an inference algorithm for parenthesis linear grammars. Further, the correctness and time complexity of the algorithm are immediately obtained from the ones of an algorithm for regular sets. It is well known (e.g., Angluin [1, 2], Biermann [5]) that any regular set is identified by some effective algorithms.

Finally, as an application of our results, the inference problem for generalized sequential machines is discussed. It is shown that if structural information of the input-output pairs are available, then the problem is also reduced to the problem for regular sets.

2. Preliminaries

Let Σ denote an alphabet and Σ^* denote the set of all strings over Σ including the null string λ. $|u|$ denotes the length of a string u.

We denote a *nondeterministic finite automaton* (abbreviated *NFA*) M by a 5-tuple $(K, \Sigma, \delta, q_0, F)$, where K is a finite nonempty set of states, Σ is a finite input alphabet, δ is a transition function from $K \times \Sigma$ to 2^K, q_0 in K is the initial state, and $F \subseteq K$ is the set of final states. The transition function δ can be extended to a function from $2^K \times \Sigma^*$ to 2^K such that $\delta(q, \lambda) = \{q\}$ and for all strings u and v in Σ^*, $\delta(q, uv) = \{p |$ for some state r in $\delta(q, u)$, p is in $\delta(r, v)\}$ for all $q \in K$, and $\delta(Q, u) = \bigcup_{q \in Q} \delta(q, u)$ for $Q \subseteq K$. The set accepted by M, denoted $T(M)$, is the set $T(M) = \{u \in \Sigma^* | \delta(q_0, u) \cap F \neq \phi\}$.

A *deterministic finite automaton* (abbreviated *DFA*) M is a 5-tuple $(K, \Sigma, \delta', q_0, F)$, where K, Σ, q_0, and F have the same meaning as for an *NFA*, but δ' is a function from $K \times \Sigma$ to K.

It is well known that if a set R is accepted by an *NFA* then there exists a DFA which accepts R.

*International Institute for Advanced Study of Social Information Science (IIAS-SIS) FUJITSU LIMITED, 140, Miyamoto, Numazu, Shizuoka 410-03, Japan

A subset R of Σ^* is called *regular* if and only if R is accepted by an *NFA*. If R is regular, then there is a minimum state *DFA*, unique up to isomorphism, which accepts R. This is called the *canonical* finite automaton for R.

A context-free grammar G over Σ is a 4-tuple (N, Σ, Π, S). N is a finite nonempty set and called *variables*. We assume that N and Σ are disjoint, and denote $N \cup \Sigma$ by V. Π is a finite nonempty set of *productions*; each production is of the form $A \to x$ where A is a variable and x is a string in V^*. We distinguish each production $A \to x$ in Π by its label π_i: $A \to x$. S is a special variable called the *start symbol*.

A *linear grammar* G is a context-free grammar such that each production in Π is of the form

$$A \to uBv \text{ or } A \to u,$$

where $A, B \in N$ and $u, v \in \Sigma^*$.

Let $G=(N, \Sigma, \Pi, S)$ be a linear grammar. We define the relation $\overset{\pi_i}{\underset{G}{\Rightarrow}}$ between strings in V^*. For $x, y \in V^*$, $x \overset{\pi_i}{\underset{G}{\Rightarrow}} y$ if and only if $x=vAw$, $y=vuw$ and π_i: $A \to u$ is a production of Π for some v, $w \in \Sigma^*$.

Let $x_0, x_1, \ldots, x_k$ be strings in V^*, where $k \geq 1$. If

$$x_0 \overset{\pi_1}{\underset{G}{\Rightarrow}} x_1, x_1 \overset{\pi_2}{\underset{G}{\Rightarrow}} x_2, \ldots, x_{k-1} \overset{\pi_k}{\underset{G}{\Rightarrow}} x_k,$$

then we denote $x_0 \overset{\alpha}{\underset{G}{\Rightarrow}} x_k$, where $\alpha=\pi_1\pi_2 \ldots \pi_k$, which is called a *derivation* from x_0 to x_k with an *associate word* α in G.

Let $L(A, G)$ denote the set $\{w \in \Sigma^* \mid A \overset{\alpha}{\underset{G}{\Rightarrow}} w, \alpha \in \Pi^*\}$. The *language generated by* G, denoted $L(G)$, is the set $L(S, G)$, i.e.,

$$L(G)=\{w \in \Sigma^* \mid S \overset{\alpha}{\underset{G}{\Rightarrow}} w, \alpha \in \Pi^*\}.$$

A language L is said to be *linear* if and only if there exists a linear grammar G such that $L=L(G)$ holds.

A linear grammar $G=(N, \Sigma, \Pi, S)$ is called in *linear normal form* (abbreviated as *LNF*) if and only if each production is of the form

$$S \to \lambda,\ A \to a,\ A \to aB, \text{ or } A \to Ba, \qquad (*)$$

where $A, B \in N$, $a \in \Sigma$. We say an *LNF* grammar for a linear grammar in *LNF*.

Proposition 2.1 Any linear language is generated by an *LNF* grammar G.

Proof. Without loss of generality, we may assume that any linear language is generated by a linear grammar $G'=(N', \Sigma, \Pi', S)$ which has no production of the form $A \to A$ or $B \to \lambda$, where $A \in N'$, $B \in N'-\{S\}$.

We construct an *LNF* grammar $G=(N, \Sigma, \Pi, S)$ from G' as follows; Every variable of G' is in N and each production of the form $S \to \lambda$, $A \to aB$, $A \to Ba$ or $A \to a$ in Π' is in Π. If a production of the form $A \to a_1a_2 \ldots a_k$ $(k \geq 2)$ is in Π', then we introduce new variables $C_1, C_2, \ldots, C_{k-1}$ into N and new productions $A \to a_1C_1$, $C_1 \to a_2C_2, \ldots, C_{k-1} \to a_k$ into Π. If a production of the form $A \to a_1a_2 \ldots a_iBb_1b_2 \ldots b_j$ is in Π', where $i \geq 2$ and $j \geq 0$, or $i \geq 0$ and $j \geq 2$, then we introduce new variables $C_1, C_2, \ldots, C_{i+j-1}$ into N and new productions $A \to a_1C_1$, $C_1 \to a_2C_2, \ldots, C_{i-1} \to a_iC_i$, $C_i \to C_{i+1}b_1$, $C_{i+1} \to C_{i+2}b_2, \ldots, C_{i+j-1} \to Bb_j$ into Π.

It is easily seen that $S \overset{\alpha}{\underset{G'}{\Rightarrow}} w$ if and only if $S \overset{\alpha}{\underset{G}{\Rightarrow}} w$. Therefore, $L(G')=L(G)$. □

In what follows, we assume that for some $w \in L(G)$ every variable A appears in some derivation $S \overset{\alpha}{\underset{G}{\Rightarrow}} w$, i.e., there exists a derivation $S \overset{\beta}{\underset{G}{\Rightarrow}} yAv \overset{\gamma}{\underset{G}{\Rightarrow}} w$, where $\alpha=\beta\gamma$.

3. Representation Theorems for Linear Languages

In this section, we show that there exists a fixed *LNF* grammar G^0 over an alphabet Σ by which any linear language L over Σ is generated with a regular control set C.

Definition Let $\Sigma=\{a_1, a_2, \ldots, a_n\}$ be an alphabet. An *LNF* grammar $G^0=(\{S^0\}, \Sigma, \Pi^0, S^0)$ is said to be *universal* if and only if Π^0 consists of the following productions,

$$\Pi^0=\left\{\begin{array}{l} S^0 \to a_1S^0, S^0 \to a_2S^0, \ldots, S^0 \to a_nS^0, \\ S^0 \to S^0a_1, S^0 \to S^0a_2, \ldots, S^0 \to S^0a_n, \\ S^0 \to a_1, S^0 \to a_2, \ldots, S^0 \to a_n, \\ S^0 \to \lambda \end{array}\right\}.$$

Note that for a given alphabet Σ, the universal *LNF* grammar G^0 is uniquely determined and $L(G^0)=\Sigma^*$.

Definition Let $G=(N, \Sigma, \Pi, S)$ be a linear grammar. Then, a subset C of Π^* is called *control set* for G and

$$L_C(G)=\{w \in \Sigma^* \mid S \overset{\alpha}{\underset{G}{\Rightarrow}} w, \alpha \in C\}$$

is called *the language generated by G with control set C*.

Let $G=(N, \Sigma, \Pi, S)$ be an *LNF* grammar, and $G^0=(\{S^0\}, \Sigma, \Pi^0, S^0)$ be the universal *LNF* grammar. We define a homomorphism h from Π^* to Π^{0*} such that

$$h(\pi)=\left\{\begin{array}{lll} \pi_i^0 \text{ where } \pi_i^0\text{: } S^0 \to aS^0, & \text{if} & \pi\text{: } A \to aB, \\ \pi_j^0 \text{ where } \pi_j^0\text{: } S^0 \to S^0a, & \text{if} & \pi\text{: } A \to Ba, \\ \pi_k^0 \text{ where } \pi_k^0\text{: } S^0 \to a, & \text{if} & \pi\text{: } A \to a, \\ \pi^0 \text{ where } \pi^0\text{: } S^0 \to \lambda, & \text{if} & \pi\text{: } S \to \lambda. \end{array}\right.$$

We construct the *corresponding NFA* $M=(K, \Pi^0, \delta, S, F)$ to G, where K, δ, F are defined as follows;

---

1. $K=N \cup \{q_F\}$ where $q_F \notin N$.
2.

$$\delta(S, \pi^0)=\begin{cases}\{q_F\} & \text{if } \pi: S\to\lambda,\ \pi^0: S^0\to\lambda, \text{ and } \pi\in h^{-1}(\pi^0),\\ \phi & \text{if } \pi^0: S^0\to\lambda \text{ and } h^{-1}(\pi^0)=\phi.\end{cases}$$

$$\delta(A, \pi_j^0)=\begin{cases}\{B\mid \pi_i: A\to aB\in\Pi,\ \pi_i\in h^{-1}(\pi_j^0)\} & \text{if } \pi_j^0: S^0\to aS^0,\\ \{B\mid \pi_i: A\to Ba\in\Pi,\ \pi_i\in h^{-1}(\pi_j^0)\} & \text{if } \pi_j^0: S^0\to S^0a.\end{cases}$$

$$\delta(A, \pi_l^0)=\begin{cases}\{q_F\} & \text{if } \pi_k: A\to a\in\Pi,\ \pi_l^0: S^0\to a, \text{ and } \pi_k\in h^{-1}(\pi_l^0),\\ \phi & \text{if } \pi_l^0: S^0\to a \text{ and } h^{-1}(\pi_l^0)=\phi.\end{cases}$$

3. $F=\{q_F\}$.

Now we have the following lemma.

Lemma 3.1 For any $w\in\Sigma^*$, $A\in N$ and $\alpha\in\Pi^*$, $A\overset{\alpha}{\underset{G}{\Rightarrow}} w$ if and only if $S^0\overset{h(\alpha)}{\underset{G^0}{\Rightarrow}} w$ and $\delta(A, h(\alpha))\ni q_F$.

Proof. If $w=\lambda$, then clearly $S\overset{\pi}{\underset{G}{\Rightarrow}}\lambda$ if and only if $S^0\overset{\pi^0}{\underset{G^0}{\Rightarrow}}\lambda$ and $\delta(S, \pi^0)\ni q_F$, where $\pi^0=h(\pi)$, $\pi: S\to\lambda$ and $\pi^0: S^0\to\lambda$.

Suppose $|w|>0$. The argument is an induction on the length of associate words α and α^0. If $A\overset{\pi}{\underset{G}{\Rightarrow}} a$, where $a\in\Sigma$, then $\pi: A\to a$ is in Π, so $S^0\overset{h(\pi)}{\underset{G^0}{\Rightarrow}} a$. By definition of δ, $\delta(A, h(\pi))\ni q_F$. Conversely, if $S^0\overset{\pi^0}{\underset{G^0}{\Rightarrow}} a$ and $\delta(A, \pi^0)\ni q_F$, then by definition of δ, there exists a $\pi\in h^{-1}(\pi^0)$: $A\to a$ and $A\overset{\pi}{\underset{G}{\Rightarrow}} a$.

Inductively suppose that for any α in Π^* and any α^0 in Π^{0*} such that $|\alpha|\le n$ and $|\alpha^0|\le n$ the assertion holds. If $A\overset{\pi}{\underset{G}{\Rightarrow}} aB\overset{\alpha}{\underset{G}{\Rightarrow}} aw$, then $S^0\overset{h(\alpha)}{\underset{G^0}{\Rightarrow}} w$ and $\delta(B, h(\alpha))\ni q_F$ by the inductive hypothesis. Since $\pi: A\to aB$ is in Π, we have $\delta(A, h(\pi))\ni B$ by definition of δ. Therefore, $S^0\overset{h(\pi)}{\underset{G^0}{\Rightarrow}} aS^0\overset{h(\alpha)}{\underset{G^0}{\Rightarrow}} aw$ and

$$\delta(\delta(A, h(\pi)), h(\alpha))=\delta(A, h(\pi)h(\alpha))=\delta(A, h(\pi\alpha))\ni q_F.$$

Conversely, suppose that $S^0\overset{\pi^0}{\underset{G^0}{\Rightarrow}} aS^0\overset{\alpha^0}{\underset{G^0}{\Rightarrow}} aw$, $\delta(A, \pi^0)\ni B$, and $\delta(B, \alpha^0)\ni q_F$. Then $B\overset{\alpha}{\underset{G}{\Rightarrow}} w$ where $\alpha\in h^{-1}(\alpha^0)$ by the inductive hypothesis. By definition of δ, for π^0: $S^0\to aS^0$ in Π^0 there exists $\pi\in h^{-1}(\pi^0)$: $A\to aB$, therefore, $A\overset{\pi\alpha}{\underset{G}{\Rightarrow}} aw$. □

The next theorem follows immediately from this lemma by putting $C=T(M)$.

Theorem 3.2 Let G^0 be the universal *LNF* grammar over Σ. Then, for any linear language L over Σ, there exists a regular control set C such that $L=L_C(G^0)$ holds.

We can also prove the converse case.

Theorem 3.3 Let G^0 be the universal *LNF* grammar over Σ and C be a regular control set for G^0. Then, $L=L_C(G^0)$ is a linear language.

Proof. Let $M=(K, \Pi^0, \delta, S, F)$ be a *DFA* such that $C=T(M)$ holds. Without loss of generality, we assume that if $\lambda\in L_C(G^0)$, the transition for the input $\pi^0: S^0\to\lambda$ is defined for the initial state S. We define an *LNF* grammar $G=(K, \Sigma, \Pi, S)$, to which M is corresponding, and a homomorphism h from Π^* to Π^{0*} as follows:

1. If $\delta(A, \pi_i^0)=B$ and π_i^0 is $S^0\to aS^0$, then $\pi_i: A\to aB$ is in Π and $h(\pi_i)=\pi_i^0$.
2. If $\delta(A, \pi_j^0)=B$ and π_j^0 is $S^0\to S^0a$, then $\pi_j: A\to Ba$ is in Π and $h(\pi_j)=\pi_j^0$.
3. If $\delta(A, \pi_k^0)\in F$ and π_k^0 is $S^0\to a$, then $\pi_k: A\to a$ is in Π and $h(\pi_k)=\pi_k^0$.
4. If $\delta(S, \pi^0)\in F$ and π^0 is $S^0\to\lambda$, then $\pi: S\to\lambda$ is in Π and $h(\pi)=\pi^0$.

By Lemma 3.1, for any $w\in\Sigma^*$, $S^0\overset{\alpha^0}{\underset{G^0}{\Rightarrow}} w$ and $\delta(S, \alpha^0)\in F$ if and only if $S\overset{\alpha}{\underset{G}{\Rightarrow}} w$, where $\alpha\in h^{-1}(\alpha^0)$. □

Thus, to infer a linear language, an inference algorithm has only to identify a regular control set for a universal *LNF* grammar. However, we have further difficulties in identifying a regular control set from examples of $L(G)$ because of the universal property of G^0. That is, in general, even if w is in $L(G)$, an associate word α^0 such that $S^0\overset{\alpha^0}{\underset{G^0}{\Rightarrow}} w$ is not always in C.

Example 1 Consider a linear language $L=\{a^nba^n\mid n\ge 0\}$. Let $G=(N, \Sigma, \Pi, S)$ be an *LNF* grammar, where $N=\{S, A\}$, $\Sigma=\{a, b\}$, and $\Pi=\{\pi_1: S\to aA,\ \pi_2: S\to b,\ \pi_3: A\to Sa\}$. Then, $L=L(G)$. The universal *LNF* grammar over Σ is $G^0=(\{S^0\}, \Sigma, \Pi^0, S^0)$, where $\Pi^0=\{\pi_1^0: S^0\to aS^0,\ \pi_2^0: S^0\to bS^0,\ \pi_3^0: S^0\to S^0a,\ \pi_4^0: S^0\to S^0b,\ \pi_5^0: S^0\to a,\ \pi_6^0: S^0\to b,\ \pi_7^0: S^0\to\lambda\}$. We define a homomorphism h from Π^* to Π^{0*} such that $h(\pi_1)=\pi_1^0$, $h(\pi_2)=\pi_6^0$, $h(\pi_3)=\pi_3^0$, and also define the corresponding *DFA* $M=(K, \Pi^0, \delta, S, F)$ to G such that $K=\{S, A, q_F\}$, $F=\{q_F\}$, and the transition function δ is defined as $\delta(S, \pi_1^0)=A$, $\delta(S, \pi_6^0)=q_F$, $\delta(A, \pi_3^0)=S$. Then, $L=L(G)=L_C(G^0)$ holds. But, even if the string *aba* is in $L(G)$, the associate word $\alpha^0=\pi_1^0\pi_2^0\pi_5^0$ such that $S^0\overset{\alpha^0}{\underset{G^0}{\Rightarrow}} aba$ is not in C.

From these observations, if we get a unique associate word in a universal *LNF* grammar and therefore, define a unique regular control set, then we can reduce the inference problem for linear languages to the problem for regular sets.

4. Parenthesis Linear Grammars

In this section, we show that for any parenthesis grammar $[G]$ of *LNF* grammar G, we can get a *unique* regular control set C such that for the parenthesis grammar $[G^0]$ of a universal *LNF* grammar G^0, $L([G])=L_C([G^0])$ holds. A parenthesis grammar of G displaies derivations in G in corresponding strings from $L(G)$.

Let $G=(N, \Sigma, \Pi, S)$ be an *LNF* grammar and $G^0=(\{S^0\}, \Sigma, \Pi^0, S^0)$ be the universal *LNF* grammar.

Definition Let $G=(N, \Sigma, \Pi, S)$ be a linear grammar. The parenthesis grammar of G is denoted by $[G]=(N, \Sigma\cup\{[\ ,\ ]\}, \Pi', S)$, where "[" and "]" are special symbols not in Σ and Π' is obtained from Π by replacing every production $A\to u$ by $A\to[u]$.

In what follows, we say the "parenthesis *LNF* grammar" for the parenthesis grammar of an *LNF* grammar, and the "universal parenthesis *LNF* grammar" for the parenthesis grammar of a universal *LNF* grammar. Let Σ_p denote the augmented alphabet $\Sigma\cup\{[\ ,\ ]\}$.

A parenthesis grammar $[G]$ is *backwards-deterministic* if and only if no two productions in $[G]$ have the same right side. Note that a backwards-deterministic parenthesis grammar $[G]$ is unambiguous, i.e., any string in $L([G])$ has the unique derivation. Clearly, any universal parenthesis *LNF* grammar $[G^0]$ is backwards-deterministic. Therefore, the next lemma follows from Lemma 3.1.

Lemma 4.1 Let $[G]$ be a parenthesis *LNF* grammar over Σ_p and $[G^0]$ be the universal parenthesis *LNF* grammar over Σ_p. Let C be a regular control set which the corresponding *NFA* to $[G]$ accepts. Then, for any $w\in\Sigma_p^*$, $w\in L([G])$ if and only if for the associate word α^0 such that $S^0 \overset{\alpha^0}{\underset{|G^0|}{\Longrightarrow}} w$, $\alpha\in C$.

Lemma 4.1 ensures that for any parenthesis *LNF* grammar $[G]$ the regular control set $C=\{\alpha^0 \mid S^0 \overset{\alpha^0}{\underset{|G^0|}{\Longrightarrow}} w, w\in L([G])\}$ is unique. Therefore, to infer a parenthesis *LNF* grammar $[G]$, we have only to construct a canonical finite automaton which accepts C. In fact, given a parenthesis *LNF* grammar $[G]$, we can effectively get a parenthesis *LNF* grammar $[G']$ such that $L([G])=L([G'])$ holds and the corresponding *NFA* to $[G']$ is canonical.

Definition A parenthesis *LNF* grammar $[G]=(N, \Sigma_p, \Pi, S)$ is *canonical* if and only if it satisfies the following conditions:

1. There is no pair of productions $A\to[aB]$, $A\to[aC]$ or $A\to[Ba]$, $A\to[Ca]$, where $B\neq C$.
2. For any distinct variables A, $B\in N$, $L(A, [G])\neq L(B, [G])$.

Note that the equivalence problem for the class of parenthesis grammars is solvable.

Lemma 4.2 Given an *LNF* grammar G, one can effectively get an *LNF grammar* G' such that $L(G')=L(G)$ and G' has no pair of productions $A\to aB$, $A\to aC$ or $A\to Ba$, $A\to Ca$, where $B\neq C$.

Proof. Given an *LNF* grammar G, by Theorem 3.2, we can get an *NFA* M corresponding to G. Let M' be the *DFA* such that $T(M)=T(M')$ holds. By Theorem 3.3, we can get a linear grammar G' from M'. Then, clearly, $L(G)=L_C(G^0)=L(G')$ holds. The construction of G' ensures that G' has no pair of productions $A\to aB$, $A\to aC$ or $A\to Ba$, $A\to Ca$. □

Lemma 4.3 Given a parenthesis *LNF grammar* $[G]$, one can effectively get a parenthesis *LNF* grammar $[G']$ such that $L([G'])=L([G])$ and for any distinct variables A and B of $[G']$, $L(A, [G'])\neq L(B, [G'])$.

Proof. Let $[G]=(N, \Sigma_p, \Pi, S)$ be a parenthesis *LNF* grammar. For any variables A, $B\in N$, if $L(A, [G])=L(B, [G])$, then we remove B from N and replace all occurrences of B in each production of Π by A. Let N' be the new set of variables and Π' be the new set of productions. Then $[G']=(N', \Sigma_p, \Pi', S)$ is a parenthesis *LNF* grammar. Clearly, $S \overset{\alpha}{\underset{|G|}{\Longrightarrow}} w$ if and only if $S \overset{\alpha}{\underset{|G'|}{\Longrightarrow}} w$. Therefore, $L([G])=L([G'])$. By repeating this procedure the proof is completed. □

From Lemmas 4.2 and 4.3, we get the following result.

Proposition 4.4 Given a parenthesis *LNF* grammar $[G]$, one can effectively get a canonical parenthesis *LNF* grammar $[G']$ such that $L([G'])=L([G])$ holds.

Proposition 4.5 Let $[G]$ be a parenthesis *LNF* grammar and M be an *NFA* corresponding to $[G]$. Then, $[G]$ is canonical if and only if M is canonical.

Proof. By Lemma 4.2, $[G]$ satisfies the condition 1 in the definition of canonical parenthesis *LNF* grammar if and only if M is a *DFA*. Since $[G^0]$ is unambiguous, for any distinct variables A and B of $[G]$, there exists a string w in Σ_p^* such that $w\in L(A, [G])$ but $w\notin L(B, [G])$ if and only if for any distinct states p and q of M, there exists an associate word α^0 such that $\delta(p, \alpha^0)\in F$ and $\delta(q, \alpha^0)\notin F$, or vice versa. □

5. Inference for Parenthesis Linear Grammars

In this section, we consider the inference problem for parenthesis *LNF* grammars.

As mentioned in the previous section, for any given parenthesis *LNF* grammar $[G]$, we can effectively get a canonical finite automaton M which accepts a unique regular control set C for a universal parenthesis *LNF* grammar $[G^0]$ such that $L([G])=L_C([G^0])$ holds. Therefore, we have the following main theorem.

Theorem 5.1 The problem of inferring an unknown parenthesis *LNF* grammar is reduced to the problem of identifying an unknown regular set.

This theorem implies that to infer parenthesis *LNF* grammars, we can use any algorithm which identifies regular sets. To constructs an inference algorithm for parenthesis *LNF* grammars, we have only to add the following auxiliary processes to an algorithm for regular sets;

1. conversions from input strings to associate words by parsing in $[G^0]$.
2. conversions from associate words to output strings by generating in $[G^0]$.
3. conversions from canonical finite automata to parenthesis *LNF* grammars.

Note that the time complexity of the conversion be-

tween a string and an associate word is $O(n^2)$, where n is the length of an input string (parsing an input string in $[G^0]$ takes n^2 steps by Earley's algorithm [6] because $[G^0]$ is unambiguous). Also note that the time complexity of the conversion from a canonial finite automaton to a canonical parenthesis *LNF* grammar is $O(m^2)$, where m is the number of states of the automaton.

Many inference althorithms for regular sets have been presented. Angluin [1] has presented an effective algorithm of identifying a regular set from a "representative sample" and membership queries. We can construct an inference algorithm for parenthesis *LNF* grammars with her algorithm. Then, a representative sample for a parenthesis *LNF* grammar $[G]$ is defined as follows;

Definition The *representative sample* for $[G]$ is a finite subset R_a of $L([G])$ such that for every production π of $[G]$ there exists a string w in R_a such that π appears in the derivation $S \overset{\alpha}{\underset{[G]}{\Rightarrow}} w$, i.e., $\alpha=\beta\pi\gamma$ for some $\beta, \gamma \in \Pi^*$.

A finite subset R_c of Π^{0*} is said to be the *associate representative sample* for C with respect to R_a if and only if it is satisfies

$$R_c=\{\alpha^0 \mid S^0 \overset{\alpha^0}{\underset{[G^0]}{\Rightarrow}} w,\ w\in R_a\}.$$

An associate representative sample corresponds to a representative sample of a regular set. We note that for every variable A of $[G]$, A appears in a derivations $S \overset{\alpha}{\underset{[G]}{\Rightarrow}} w$ for some $w\in R_a$.

The time which Angluin's algorithm takes is bounded by a polynomial of the size of an alphabet and the size of a given representative sample. Therefore, using her algorithm, any parenthesis *LNF* grammar can be inferred in a polynomial time of the size of an alphabet and the size of a representative sample.

Example 2 We illustrate a process of the inference algorithm for parenthesis *LNF* grammars with Angluin's algorithm.

Consider a parenthesis *LNF* grammar $[G]=(N, \Sigma_p, \Pi, S)$, where $N=\{S, A\}$, $\Sigma=\{a, b\}$, $\Pi=\{S\to[aA], S\to[b], A\to[Sa]\}$. Clearly, $[G]$ is canonical. The universal parenthesis *LNF* grammar over Σ_p is $[G^0]=(\{S^0\}, \Sigma_p, \Pi^0, S^0)$, where $\Pi^0=\{\pi_1^0\colon S^0\to[aS^0], \pi_2^0\colon S^0\to[bS^0], \pi_3^0\colon S^0\to[S^0a], \pi_4^0\colon S^0\to[S^0b], \pi_5^0\colon S^0\to[a], \pi_6^0\colon S^0\to[b], \pi_7^0\colon S^0\to[\ ]\}$. A representative sample R_a for $[G]$ is $\{[a[[b]a]]\}$. Then, the associate representative sample R_c with respect to R_a is $\{\pi_1^0\pi_3^0\pi_6^0\}$.

First, the inference algorithm constructs R_c from a given R_a, then using Angluin's algorithm, constructs a canonical finite automaton. In the process, her algorithm outputs associate words to ask whether they are in the unknown regular control set or not. Then, the inference algorithm converts them to strings and ask whether the strings are generated by the unknown parenthesis *LNF* grammar or not.

Angluin's algorithm outputs the canonical finite automaton $M'=(K, \Pi^0, \delta, q_0, F)$ as follows;

1. $K=\{q_0, q_1, q_2, q_3\}$, where
   $q_0=\{\lambda, \pi_1^0\pi_3^0\}$,
   $q_1=\{\pi_1^0, \pi_1^0\pi_3^0\pi_1^0\}$,
   $q_2=\{\pi_6^0, \pi_1^0\pi_3^0\pi_6^0\}$,
   $q_3=\{d_0, \pi_2^0, \pi_3^0, \pi_4^0, \pi_5^0, \pi_1^0\pi_1^0, \pi_1^0\pi_2^0, \pi_1^0\pi_4^0, \pi_1^0\pi_5^0, \pi_1^0\pi_6^0, \pi_1^0\pi_3^0\pi_2^0, \pi_1^0\pi_3^0\pi_3^0, \pi_1^0\pi_3^0\pi_4^0, \pi_1^0\pi_3^0\pi_5^0\}$.
2. $F=\{q_2\}$,
3. the transition function δ: $K\times\Pi^0\to K$ is defined as follows

$\delta(q_0, \pi_1^0)=q_1$,
$\delta(q_0, \pi_6^0)=q_2$,
$\delta(q_1, \pi_3^0)=q_0$,
$\delta(q, \pi^0)=q_3$ for all other $\pi^0\in\Pi^0$ and $q\in K$.

Regarding states of M' as variables, the inference algorithm get a grammar $[G']=(N', \Sigma_p, \Pi', S')$, where $N'=\{S', A'\}$ and $S'=q_0$, $A'=q_1$, and $\Pi'=\{S'\to[aA'], S'\to[b], A'\to[S'a]\}$. Clearly, $L([G])=L([G'])$.

Angluin [2] has also presented another inference algorithm for regular sets. It is related to the method used by Gold [8] to show that the problem of finding a canonical finite automaton compatible with a given finite set of positive and negative examples is NP-hard. The time which her second algorithm takes is bounded by a polynomial of the number of states of the canonical finite automaton and the maximum length of any counter-example. If we construct an inference algorithm for parenthesis *LNF* grammars with her second algorithm, then the time which our algorithm takes is also bounded by a polynomial of the number of variables of the canonical parenthesis *LNF* grammar and the maximum length of any counter-example.

6. Inference for Generalized Sequential Machines

In this section, we consider an inference method for generalized sequential machines. It will be used to infer a transducer from tokens in one programming language to tokens in another programming language.

A *generalized sequential machine* (*GSM*) is a 6-tuple $S_g=(K, \Sigma, \Delta, \delta, q_0, F)$, where K, Σ, and Δ are the states, input alphabet, and output alphabet, respectively, δ is a mapping from $K\times\Sigma$ to finite subsets of $K\times\Delta^*$, q_0 is the initial state, and F is the set of final states.

We extend δ to a function from $K\times\Sigma^*$ to $K\times\Delta^*$ such that for all $q\in K$, $\delta(q, \lambda)=\{(q, \lambda)\}$ and $\delta(q, ua)=\{(p, w) \mid w=w_1w_2$ and for some p', (p', w_1) is in $\delta(q, u)$ and (p, w_2) is in $\delta(p', a)\}$ for all string $u\in\Sigma^*$ and all $a\in\Sigma$.

Any string in the set $\{u\in\Sigma^* \mid (p, v)\in\delta(q_0, u)$ and $p\in F\}$ is called an *input* of *GSM* S_g, and any string in the set $\{v\in\Delta^* \mid (p, v)\in\delta(q_0, u)$ and $p\in F\}$ called an *output* of S_g.

We first show a relation between a *GSM* and an *LNF* grammar. If u is a string, then we denote by u^T the mirror image of u. For example, $abcd^T$ is $dcba$.

Proposition 6.1 For any strings $w_1 \in \Sigma^*$ and $w_2 \in \Delta^*$, there exists a *GSM* $S_g=(K, \Sigma, \Delta, \delta, q_0, F)$ such that $(p, w_2) \in \delta(q_0, w_1)$ and $p \in F$ if and only if there exists an *LNF* grammar G over $\Sigma \cup \Delta \cup \{\#\}$ such that $w_1 \# w_2^T \in L(G)$, where "#" is a new symbol not in $\Sigma \cup \Delta$.

Proof. Let $S_g=(K, \Sigma, \Delta, \delta, S, F)$ be a *GSM*. We construct an *LNF* grammar $G=(N, \Sigma \cup \Delta \cup \{\#\}, \Pi, S)$ as follows; All states in S_g are variables in G. If $\delta(A, a) \ni (B, \lambda)$, then we introduce the production $A \to aB$ into Π. If $\delta(A, a) \ni (B, w)$ and $w=b_1 b_2 \ldots b_n$ $(n>0)$ where $b_1, b_2, \ldots, b_n \in \Delta$, then we introduce new variables $A', B_1, B_2, \ldots, B_n$ into N, and productions $A \to aA'$, $A' \to B_1 b_1$, $B_1 \to B_2 b_2, \ldots, B_n \to B b_n$ into Π. For any state A is in F, we introduce the production $A \to \#$ into Π.

Conversely, given an *LNF* grammar $G=(N, \Sigma \cup \Delta \cup \{\#\}, \Pi, S)$, then we construct a *GSM* $S=(K, \Sigma, \Delta, \delta, S, F)$ as follows; For any production of the form $A \to aB$ in Π, if there exists a derivation $B \overset{\alpha}{\underset{G}{\Rightarrow}} Cw$ such that $w \in \Delta^* - \{\lambda\}$, $\alpha \in \Pi^*$ and there is no production of the form $C \to Db$, and C does not appear in the derivation, then we introduce A and C into K and define $\delta(A, a) \ni (C, w^T)$. If such a derivation does not exist, then we introduce A and B into K and define $\delta(A, a) \ni (B, \lambda)$. If a production of the form $A \to \#$ is in Π, then A is in F.

In both case, by the obvious induction, it is easy to verify that for any $w_1 \in \Sigma^*$ and $w_2 \in \Delta^*$, $\delta(S, w_1) \ni (A, w_2)$ and $A \in F$ if and only if $S \overset{\alpha}{\underset{G}{\Rightarrow}} w_1 \# w_2^T$. □

A *GSM* S_g is said to be a *parenthesis GSM* if and only if an *LNF* grammar constructed from S_g by Proposition 6.1 is a parenthesis *LNF* grammar.

About the inference problem for parenthesis *GSMs*, we have the following theorem by Theorem 5.1 and Proposition 6.1.

Theorem 6.2 The inference problem for parenthesis *GSMs* is reduced to the problem for regular sets.

We note that constructing a *GSM* S_g from an *LNF* grammar G in the obvious way takes time polynomial in the size of productions of G.

Example 3 We consider the following *GSM* S_g; Suppose that $\Sigma=\{a, b, c, \&\}$ and $\Delta=\{0, 1\}$. A *GSM* S_g receives as input any string from Σ^* and converts symbols a, b, c to 00, 01, 11, respectively, from left to right as long as & does not appear. If & appears, then the rest of input is deleted from the output. For example, if S_g receives as input $abcb\&abc$, then it outputs 00011101.

Such a *GSM* may be needed when a first legal part may be taken as a token in a programming language from any string in Σ^*. In the above example, if we consider & as an illegal character to construct tokens, then $abcb$ is the legal part of $abcb\&abc$ as a token.

GSM $S_g=(K, \Sigma, \Delta, \delta, q_0, F)$ can be defined as follows;

1. $K=\{q_0, q_1\}$,
2. $\delta(q_0, a)=\{(q_0, 00)\}$, $\delta(q_0, b)=\{(q_0, 01)\}$, $\delta(q_0, c)=\{(q_0, 11)\}$, $\delta(q_0, \&)=\{(q_1, \lambda)\}$, $\delta(q_1, x)=\{(q_1, \lambda)\}$ for all $x \in \Sigma$,
3. $F=K$.

From *GSM* S_g, we can construct an *LNF* grammar $G=(N, \Sigma \cup \Delta \cup \{\#\}, \Pi, S)$, where $N=\{S, A, A', B, B', C, C', D\}$ and

$$\Pi=\{S \to aA, S \to bB, S \to cC, S \to \&D, S \to \#,$$
$$A \to A'0, A' \to S0, B \to B'0, B' \to S1, C \to C'1, C' \to S1,$$
$$D \to aD, D \to bD, D \to cD, D \to \&D, D \to \#\}.$$

To infer *GSM* S_g, we first infer the parenthesis *LNF* grammar of G. For example, from a representative sample $\{[a[[[\#]0]0]], [b[[[\#]1]0]], [c[[[\#]1]1]], [\&[a[b[c[\#]]]]]\}$ and membership queries, we can get the canonical parenthesis *LNF* grammar $[G]$. Then, parenthesis *GSM* $[S_g]$ and therefore, *GSM* S_g can be easily constructed from $[G]$.

7. Concluding Remarks

Angluin [3] has presented an inference algorithm for k-bounded context-free grammars, where a context-free grammar is k-bounded if no production has more than k occurrences of variables in its right-hand side (Note that a linear grammar is a 1-bounded context-free grammar). Her algorithm assumes that the set of variables of an unknown grammar are known, and identifies the productions of the grammar. On the other hand, our method only requires structural informations (parentheses) of an unknown grammar, then identifies variables of the grammar (Our method assumes that the grammar is in *LNF*, but this is not critical because for any linear grammar, one can effectively get an *LNF* grammar).

In general, the inference problem for linear languages can not be reduced to the problem for regular control sets for universal *LNF* grammars. Otherwise, the equivalence problem for linear languages could be solvable. However, representation theorems for linear languages we have shown suggests that, to infer linear languages, inference methods similar to the ones for regular sets are promising. This will be discussed elsewhere.

Linear grammars are closely related to two-tape automata and a-transducers, as well as generalized sequential machines. Therefore, our method can be the first step towards the study of inductive inference methods for more general transducers.

Acknowledgements

The author is indebted to Dr. Toshio Kitagawa, the president of IIAS-SIS, Dr. Hajime Enomoto, the director of IIAS-SIS, for their useful suggestion and warm encouragement. He is also grateful to his colleagues, Dr. T. Yokomori, Y. Sakakibara, H. Ishizaka and Dr. T. Nishida who worked through an earlier draft of the paper and gave many suggestions.

This is a part of the work in the major R&D of the Fifth Generation Computer Project, conducted under a program set up by MITI.

References

1. ANGLUIN, D. A Note on the Number of Queries Needed to Identify Regular Languages, *Information and Control* **51** (1981), 76–87.
2. ANGLUIN, D. Learning Regular Sets from Queries and Counterexample, *Information and Computation* **75** (1987), 87–106.
3. ANGLUIN, D. Learning k-bounded context-free grammars, *Technical Report, Yale University Computer Science Dept.*, RR-557 (1987).
4. BIERMANN, A. W. A Grammatical Inference Program for Linear Languages, *Proc. of Forth Hawaii International Conference on System Sciences* (1971), 12–14.
5. BIERMANN, A. W. An Interactive Finite-state Language Learner, *Proc. of First USA-JAPAN Computer Conference (1972), 13–20.*
6. EARLEY, J. An Efficient Context-Free Parsing Algorithm, *Comm. of the ACM* **13** (1970), 94–102.
7. GINSBURG, S. and SPANIER, E. H. Control Sets on Grammars, *Math. Systems Theory* **2** (1968), 159–177.
8. GOLD, E. M. Complexity of Automaton Identification from Given Data, *Information and Control* **37** (1978), 302–320.
9. HARRISON, M. A. Introduction to Formal Language Theory, *Addison-Wesley, Reading, Mass.* (1978).
10. HOPCROFT, J. E. and ULLMAN, J. D. Introduction to Automata Theory, Languages and Computation, *Addison-Wesley, Reading, Mass.* (1979).
11. MCNAUGHTON, R. Parenthesis Grammars, *J. of the ACM* **14**, (1967), 490–500.
12. TAKADA, Y. A Constructive Method for Grammatical Inference of Linear Languages Based on Control Sets, IIAS-SIS *Research Report* **78**, *FUJITSU LIMITED* (1987).
13. TANATSUGU, K. A Grammatical Inference for Context-free Languages based on Self-embedding, *Bulletin of Informatics and Cybernetics* **22** (1987), 149–163.

(Received November 27, 1987; revised June 20, 1988)