ICOT Technical Report: TR-359

TR-359

Learning Simple Languages in Polynomial Time

by
T. Yokomori

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 Languages in Polynomial Time

by

Takashi YOKOMORI

March 1988

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

Contents

Abstract

A grammatical inference problem for simple deterministic context-free grammars is investigated. The inference algorithm, based on the constructive method, is given in which from two oracles for a simple language L (one is called *prefix-membership oracle for L*, the other *derivative oracle for L*), the algorithm learns L in time polynomial in the size of a minimum grammar for L.

1. Introduction

An inductive inference is, in general, recognized as a process of finding a finite set of rules which explains given many examples. The mechanism underlying is one of the most significant functions for supporting knowledge acquisition process in the various phases of problem solving we encounter, and it is also one of the primary subjects in the research on machine learning.

In the context of grammatical inference, quite a few efforts have been devoted to developing efficient inference algorithms for the class of regular grammars or finite-state acceptors ([An78],[Bi72],[ET72],[Is88]) and the class of context-free grammars([KK77],[Ta82],[Ta87],[Tak87],[Sak87],[Wh77],[Yo88a]). Recently, Angluin gives a constructive induction algorithm for regular languages which runs in polynomial time[An87]. The problem setting she deals with in the paper assumes two types of oracles: One is for testing a conjecture and indicating if it is equal to the unknown set or not, and the other is for providing a counter-example if not. Using these abilities successfully, the algorithm learns the canonical acceptor for the unknown regular set in polynomial time. It seems, however, that the latter type of an oracle is harder for users(humans) to carry out as the size of a conjecture becomes larger. This becomes much more critical when the class of more complicated grammars is targeted.

Talking of the grammatical inference of context-free grammars, [Cr72],[Fa83] and [Sak87] deal with the problem setting in which the unknown grammar is inferred from its structural examples. The user is expected to perform an oracle which provides the algorithm with structural examples of the unknown grammar. In practical use of the inductive inference, however, it is not so easy for users to behave as a perfect oracle for providing structural examples, unless the user knows the unknown grammars.

In this paper, we present an algorithm for inferring language class called simple (deterministic context-free) languages. The class of simple languages is,

in principle, larger than that of regular languages, and takes an important position to develop the fundamental tools for designing and constructing parsers or compilers.

The algorithm given in this paper, belonging to the category of constructive induction method, requires two kinds of oracles for the target language L: the *prefix-membership oracle*(an extension of the membership oracle) and the *derivative oracle*, and it learns L in polynomial time.

This paper is organized as follows: In Section 2, formal definitions needed for discussing the problem of inductive inference is given. Section 3.1 introduces simple (deterministic context-free) grammars and their languages. In Section 3.2 the notions of cover graph and characteristic cover graph of a simple grammar are introduced, and a method for reconstructing a simple grammar from its characteristic cover graph is presented in Section 3.3. Section 4 deals with the inductive inference problem for simple languages and gives an inference algorithm for the problem together with the complexity result. Example runs are also provided in Section 4.3. Finally, Section 5 concludes this paper by briefly mentioning an application to compiler construction and a possible variant of the algorithm.

2. Preliminaries

We shall give some basic notions and notations needed through this paper. (The reader is assumed to be familiar with the rudiments in the formal language theory. See, e.g., [Sa73] or [Ha78] for definitions not mentioned here.)

Definition

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 ε.) A *language* L *over* Σ is a subset of Σ^*. For a string(or *word*) x in Σ^* and a language L over Σ, let $x\backslash L = \{w \mid xw \in L\}$ ($L/x = \{w \mid wx \in L\}$). The set $x\backslash L(L/x)$ is called the *left(right)*-

derivative of L with respect to x. For any subset S of Σ^*, *Prefix*(*S*) denotes the set of all prefixes of strings in S, while by *Suffix*(*S*) we denote the set of all suffixes of strings in S. That is, Prefix(S) = {x| for some $z \in \Sigma^*$, $xz \in S$ } and Suffix(S) = {x|for some $z \in \Sigma^*$, $zx \in S$ }. For $w \in \Sigma^*$, we simply write Prefix(w) rather than Prefix({w}). Let L_1 and L_2 be languages, then the *product* of L_1 and L_2 is a language $L_1L_2 = \{xy | x \in L_1, y \in L_2\}$. In particular, when L_1 is a singleton, i.e., $L_1 = \{x\}$, we wirte xL_2 instead of $\{x\}L_2$.

A *context-free grammar* is a 4-tuple $G = (N,\Sigma,P,S)$, where N is a finite alphabet of nonterminals, Σ is a finite alphabet of terminals such that $N \cap \Sigma = \Phi$, S is a distinguished element of N called the initial symbol, and P is a finite set of production rules of the form $A \rightarrow w$ ($A \in N$, $w \in (N \cup \Sigma)^*$). For x, $y \in (N \cup \Sigma)^*$, a binary relation $\Rightarrow$ is defined as follows: $x \Rightarrow y$ iff there exist u, $v \in (N \cup \Sigma)^*$, $A \rightarrow w \in P$ such that $x = uAv$ and $y = uwv$. Let $\Rightarrow^*$ be the reflexive, transitive closure of $\Rightarrow$. For A in N, A is *recursive* if there exists a derivation : $A \Rightarrow^* xAy$, for $\exists x,y \in (N \cup \Sigma)^*$.

For α in N^+, we define $L(\alpha) = \{x \mid \alpha \Rightarrow^* x \text{ and } x \in \Sigma^*\}$. In particular, for the initial symbol S, a set L(S) is denoted by L(G) and is called the language generated by G. A language is called *context-free* if there exists a context-free grammar G such that L = L(G) holds.

3. Characterizations of Simple Languages

3.1 Simple Grammars and their Languages

Definition(Simple deterministic grammar/Simple grammar)

A context-free grammar in Greibach normal form is *simple* (deterministic) if for A in N, a in Σ, α, β in N^*, $A \rightarrow a\alpha$ and $A \rightarrow a\beta$ in P implies $\alpha = \beta$.

Note that the definition does not imply that simple grammars generate only ε-free languages. In this paper, however, our attention focuses on only ε-free simple grammars.

Example 3.1

Consider a simple grammar $G=(\{S,A,B,C\},\{a,b\},P,S)$, where P is defined as follows:

$S \rightarrow aAC, A \rightarrow a, A \rightarrow bAB, B \rightarrow b, C \rightarrow a.$

Then, we have $L(G)=\{ab^n ab^n a \mid n \geq 0\}$, which is not regular.□

The following proposition provides preferable features of simple grammars and languages for our purpose.

Proposition(e.g.,[Ha78])

Given any simple grammar G, there effectively exists an equivalent simple grammar G' which is reduced and in 2-standard form, i.e., there effectively exists a simple grammar $G'=(N,\Sigma,P,S)$ *such that*

(1) $L(G) = L(G')$ *holds,*

(2) *for* A, B *in* N *such that* $A \neq B$, $L(A) \neq L(B)$ *holds, and*

for A *in* N, *there uniquely exist left-most derivations :* $S \Rightarrow^* xAy$ *and* $A \Rightarrow^* w$

(*where* y *in* N^*, x, w *in* T^*) *in* G',

(3) *each rule of* G' *is of one of the following forms:* $A \rightarrow aBC, A \rightarrow aB, A \rightarrow a$,

where A,B,C *in* N, a *in* Σ.

[Convention]

(1) In what follows, we consider only *ε-free reduced simple grammar in 2-standard form.*

(2) Otherwise stated, the derivation $\Rightarrow^*$ always means the *left-most* one.

Simple languages can be viewed as generalizations of regular languages in the following sense.

Lemma 3.1

For any regular language R, $R\not\in$ is a simple language, where $\not\in$ is a specific symbol not in the terminal alphabet of R.

Proof. Let $A=(Q,\Sigma,\delta,p_0,F_0)$ be a deterministic finite-state automaton such that $T(A)=R$. Construct a simple grammar $G=(N,\Sigma',P,p_0)$ as follows: $N=Q$, $\Sigma'=\Sigma\cup\{¢\}$, $P=\{p\to aq|\delta(p,a)=q\}\cup\{q\to ¢|q\in F_0\}$. It is obvious that G is simple and $L(G)=R¢$ holds.□

Definition(Transition graph of G: T_G)

Given a simple grammar $G=(N,\Sigma,P,S)$, we associate with the (possibly infinite) *transition graph* $T_G=(N_G, E_G, \Sigma)$ of G defined as follows :

(1) $N_G=(\{\alpha|S\Rightarrow^*x\alpha$, for some $x\in\Sigma^*$, and $\alpha=\varepsilon\}/\equiv)\cup\{F\}$, where $\alpha\equiv\alpha'$ iff $L(\alpha)=L(\alpha')$,

(2) For A in N, α, β in N^*, and a in Σ,

E_G contains $A\alpha\to^a\beta\alpha$ iff there is a derivation :

$$S\Rightarrow^*xA\alpha\Rightarrow xa\beta\alpha, \text{ for some x in } \Sigma^*, \text{ and}$$

E_G contains $A\to^a F$ iff there is a derivation :

$$S\Rightarrow^*xA\Rightarrow xa, \text{ for some x in } \Sigma^*,$$

where F is the special symbol not used elsewhere.

The node S is called the initial node, while F the final node. When $\alpha_1\to^{a_1}\alpha_2\to^{a_2}\cdots\to^{a_n}\alpha_n$ $(n\geq 1)$holds in T_G, we denote it by $\alpha_1\to^{a_1\cdots a_n}\alpha_n$.

Example 3.2

Consider a simple grammar $G=(N,\Sigma,P,S)$ given in Example 3.1. Then, the transition graph T_G is as in Figure 3.1.□

Definition (Graph isomorphism)

Let $T_i=(N_i,E_i,\Sigma)(i=1,2)$ be a labeled digraph. Then, T_1 is *isomorphic* to T_2, denoted by $T_1\simeq T_2$, if there is a bijection b from N_1 onto N_2 such that for any α, β in N_1 and a in Σ, $\alpha\to^a\beta$ in E_1 iff $b(\alpha)\to^a b(\beta)$ in E_2.

Definition (Structure graph of L : T_L)

Let L be a simple language over Σ. Consider a labeled digraph $T_L=(N_L, E_L, \Sigma)$, where $N_L=\cup_{j\geq 0}N_L(j)$ and E_L are constructed by the following procedure :

[Procedure]

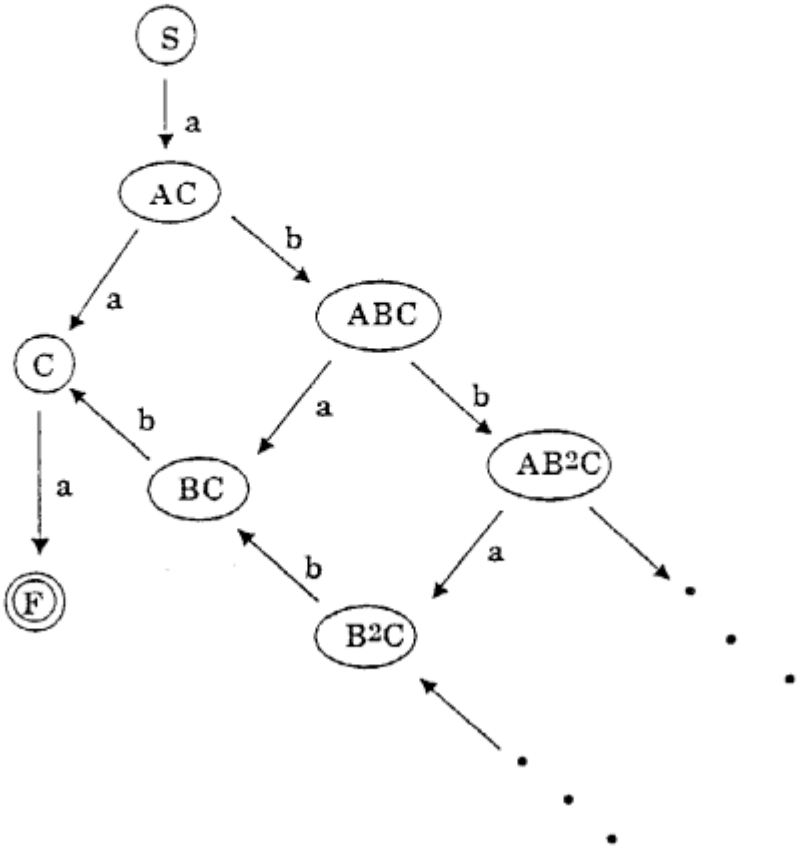


Figure 3.1 Transition graph of G

step 0: let $N_L(0) = New_L(0) = \{\varepsilon\backslash L (= L)\}$ and initialize $E_L \leftarrow \varnothing$

set $i = 1$

step i: initialize $New_L(i) \leftarrow \varnothing$

for all $x\backslash L$ in $New_L(i-1)$

do let $\Gamma_x = \{a \in \Sigma | xa\backslash L \neq \varnothing\}$*)

for each $a \in \Gamma_x$

do if $xa\backslash L \notin \cup_{j=0}^{i-1} N_L(j)$**)

then $E_L \leftarrow E_L \cup \{x\backslash L \rightarrow^a xa\backslash L\}$ and $New_L(i) \leftarrow New_L(i) \cup \{xa\backslash L\}$

else find $u\backslash L \in \cup_{j=0}^{i-1} N_L(j)$ such that $u\backslash L = xa\backslash L$

if $x\backslash L \rightarrow^a u\backslash L \notin E_L$, then $E_L \leftarrow E_L \cup \{x\backslash L \rightarrow^a u\backslash L\}$

let $N_L(i) = \cup_{j=0}^{i} New_L(j)$

if $N_L(i) = N_L(i-1)$, then halt

else set $i \leftarrow i+1$ and go to step i

T_L is called the *structure graph of L*. When $x\backslash L \rightarrow^{a_1} xa_1\backslash L \rightarrow^{a_2} \cdots \rightarrow^{a_n} xa_1\cdots a_n\backslash L (n \geq 1)$ holds, we denote by $x\backslash L \rightarrow^{u} xu\backslash L$, where $u = a_1\cdots a_n$. For any $x\backslash L$ in N_L, define $L(x\backslash L) = \{w \mid x\backslash L \rightarrow^{w} \{\varepsilon\}\}$. Note that $L(\varepsilon\backslash L) = L(L) = L$, $L(\{\varepsilon\}) = \varnothing$, and $u\backslash L = L(u\backslash L)$ (for $\forall u$) hold.

Notes

(i) $z\backslash L (= \{\varepsilon\})$, where z is one of the shortest words in L, provides the final node unique in T_L.

(ii) T_L is , in general, an infinite graph.

*) Note that if L is simple, then a left-derivative $xa\backslash L$ is also simple, and the emptiness problem for simple languages is decidable.

**)Also note that the equivalence problem for simple languages is decidable.[KH66]

Example 3.3

Consider a simple language $L = \{ab^n ab^n a \mid n \geq 0\}$. Then, the structure graph T_L is as in Figure 3.2. Note that $aaa\backslash L = \{\varepsilon\}$, and for all $n \geq 0$ $ab^n a\backslash L = ab^{n+1}ab\backslash L = \{b^n a\}$.□

Lemma 3.2

Let $G = (N, \Sigma, P, S)$ be a simple grammar and let $L = L(G)$. Then, $T_L \simeq T_G$ holds.

Proof. Define a mapping b as follows:

$b(L) = S$, $b(\{\varepsilon\}) = F$,

and for any $u\backslash L$ in $N_L - \{L, \{\varepsilon\}\}$ $b(u\backslash L) = \alpha$, where $S \rightarrow^{u} \alpha$ in T_G.

The mapping b is well-defined, i.e., it holds that $u\backslash L = \varnothing$ implies the existence of v such that uv is in L, hence there exists α such that $S \rightarrow^{u} \alpha$ and $\alpha \rightarrow^{v} F$. Further, suppose $S \rightarrow^{u} \alpha$ and $S \rightarrow^{u'} \alpha'$, then by the definition of the structure graph T_G, $u\backslash L = u'\backslash L$ implies $\alpha = \alpha'$.

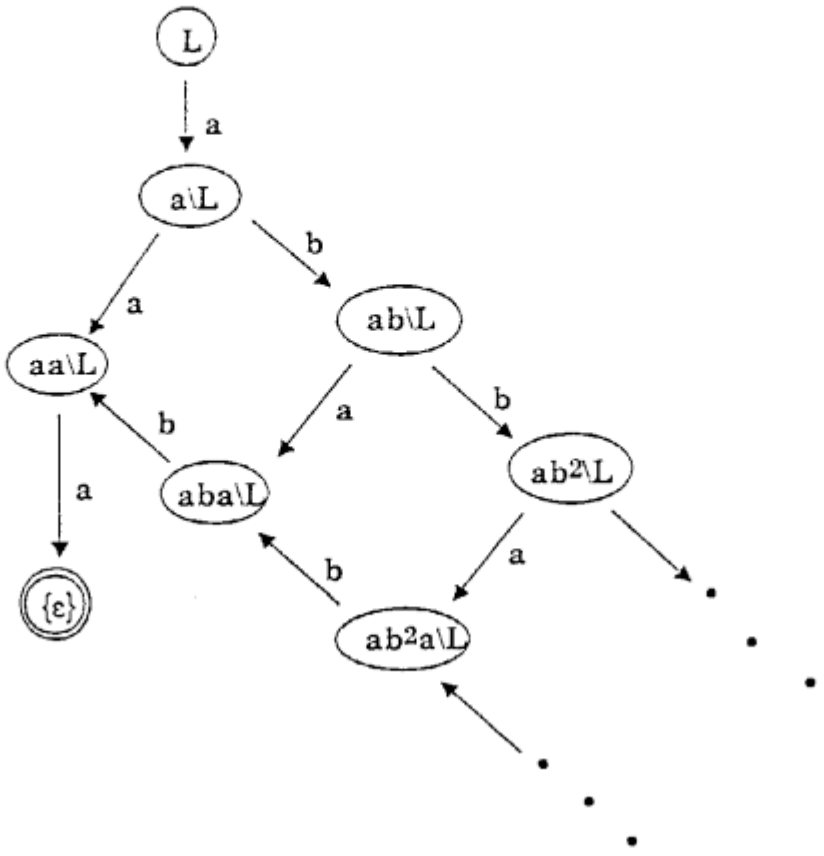


Figure 3.2 Structure graph of L

Now, we shall show the mapping b is bijective. (Note that $L(\alpha)=u\backslash L$ if $S \rightarrow^{u} \alpha$.) Suppose $b(u\backslash L)=b(u'\backslash L)$, i.e., $\alpha=\alpha'$, where $S \rightarrow^{u} \alpha$ and $S \rightarrow^{u'} \alpha'$. Then, $L(\alpha)=L(\alpha')$, i.e., $u\backslash L=u'\backslash L$ is obtained. Further, let α be in N_G, i.e., $S \rightarrow^{u} \alpha$ for some u in Σ^*. Then, by the definition of G, $L(\alpha)\neq\varnothing$, i.e., $u\backslash L\neq\varnothing$, hence $u\backslash L$ is in N_L.

Further, let $u\backslash L \rightarrow^{a} ua\backslash L$ be in T_L and let $b(u\backslash L)=\alpha=A\beta(A\in N, \beta\in N^*)$. Then, since $ua\backslash L\neq\varnothing$, there exists v such that uav is in L. Hence, there exists $A\rightarrow a\gamma$ in $\dot{P}$ such that $\alpha \rightarrow^{a} \gamma\beta$ and $\gamma\beta \rightarrow^{v} F$, that is, $S \rightarrow^{ua} \alpha'$ and $\alpha'=\gamma\beta$, where $b(ua\backslash L)=\alpha'$. Thus, $\alpha \rightarrow^{a} \alpha'$, i.e., $b(u\backslash L) \rightarrow^{a} b(ua\backslash L)$ holds in T_G.

Conversely, suppose $b(u\backslash L) \xrightarrow{a} b(ua\backslash L)$ holds in T_G, and let $\alpha = b(u\backslash L)$ and $\alpha' = b(ua\backslash L)$. By definition, $S \xrightarrow{u} \alpha \xrightarrow{a} \alpha'$ and $L(\alpha) = \varnothing$, $L(\alpha') = \varnothing$. Hence, $u\backslash L = \varnothing$, $ua\backslash L = \varnothing$ and it follows that $u\backslash L \xrightarrow{a} ua\backslash L$ holds in T_L.□

The following is immediately obtained from Lemma 3.2.

Theorem 3.1

Let $G_i (i = 1,2)$ be simple grammars. Then, $L(G_1) = L(G_2)$ holds if and only if T_{G_1} is isomorphic to T_{G_2}.

Proof. By Lemma 3.2, if $L(G_1) = L(G_2) (= L)$ holds, then $T_{G_1} \simeq T_L \simeq T_{G_2}$. Conversely, it is obvious that $T_{G_1} \simeq T_{G_2}$ implies $L(G_1) = L(G_2)$.□

Thus, given a simple grammar G such that $L(G) = L$, there exists the structure graph T_L unique up to isomorphism.

3.2 Cover Graph and Characteristic Cover Graph

Now, we introduce the notion of a cover graph of a simple grammar which is an extension of the cover tree for a finite-state automaton[Bi72], and is basically due to [EDM71].

Definition (Cover graph of G: C_G)

Let $G = (N,\Sigma,P,S)$ be a simple grammar and T_G be its transition graph, where a total order(e.g., an alphabetical order) on Σ is assumed. For each A in N, let $\Sigma_A = \{a \in \Sigma \mid A \Rightarrow a\alpha_1 \Rightarrow x_2 \cdots \Rightarrow x_k \in \Sigma^* : \text{shortest derivation}\}$. Then, we call $A \rightarrow a\alpha_1$ the *SH-rule* of A if a is the first element of Σ_A in the assumed order, and a path $A \xrightarrow{a} \alpha_1 \text{---} \cdots \text{---} F$ is called the *key path* for A.

Construct a finite subgraph C_G of T_G as follows:

[step 1] For each a in Σ such that $S \rightarrow a\alpha \in P$, extend the initial node S in the assumed order by making son node α.(where if $S \rightarrow a \in P$, then $\alpha = F$ and F is never extended furthermore.)

[step i] Let $\alpha = Au(A \in N, u \in N^*)$ be a node created at step (i-1). Then, take each α in the order created and apply the following procedure: If A does not yet appear as the left-most nonterminal of any node created in the previous steps(in each step less than (i-1)), or in the previous procedure at step(i-1), then for each a in Σ such that $A \rightarrow a\beta \in P$, extend α in the assumed order by making a son node βu with the edge labeled a. Otherwise, extend it by making a son node for only the SH-rule of A, and at the same time, make an edge, represented by a dotted arrow labeled L_A, from node $\alpha(=Au)$ to u. In case of $\beta u = \varepsilon$, make an edge from A to F with the label a. Let i be i + 1(i.e., increase the value of i by one) and repeat step i until the above procedure cannot be applied to any node, or the extension of any node has been completed.

It is easy to see that this procedure terminates in finite time. The graph $C_G = (N_G, E_G, \Sigma)$ is called the *cover graph* of G.

Let $\alpha = A\beta(A \in N, \beta \in N^*)$ be a node of C_G which is neither initial nor final. Then, if there is a dotted arrow labeled L_A going out of α and degree(A) $\geqq$ 2, then such a node α is called *incomplete,* while if degree(A) = 1, then it is called *redundant.* A node in $N_G - (\{\alpha | \alpha$:incomplete or redundant$\} \cup \{F\})$ is called *complete.*

Definition(Characteristic cover graph of G: CC_G)

Let C_G be the cover graph of a simple grammar $G = (N, \Sigma, P, S)$. Then, consider a graph which is obtained from $C_G = (N_G, E_G, \Sigma)$ by relabeling nodes as follows: Let $A\beta \in N_G - \{\alpha | \alpha$:incomplete or redundant$\}$ be a complete node$(A \in N, \beta \in N^*)$. Then, relabel it with a new symbol X_A. For an incomplete node or a redundant node $\alpha = A\beta$, relabel it with a new arrow's label L_{X_A}. Further, for F, we don't change it, i.e., relabel it with F itself. Let $Relabel_G = \{(\alpha, \alpha') | \alpha \in N_G, \alpha'$:new label for $\alpha\}$, $N'_G = \{\alpha' | (\alpha, \alpha') \in Relabel_G\}$ and E'_G be the set of edges obtained from E_G by relabeling. The resulting graph (N'_G, E'_G, Σ) is called the *characteristic cover graph of G,* and is denoted by CC_G. *$Relabel_G$* is called *node relabeling information*

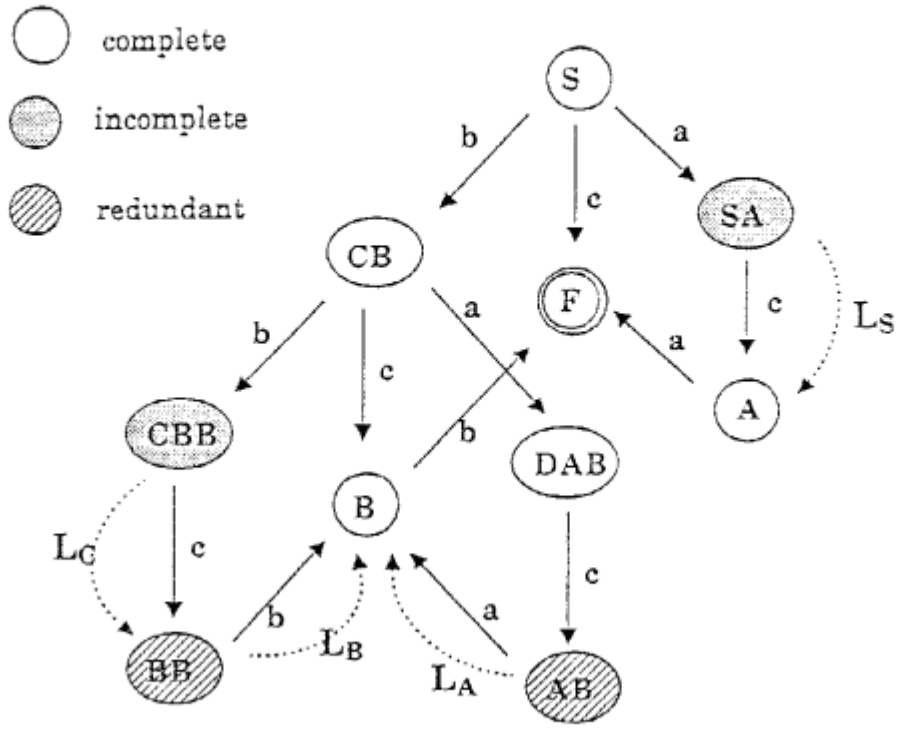


Figure 3.3 Cover graph of G: C_G

for CC_G. A node α' in N'_G is *complete*(*incomplete*) if so is α in N_G, where $(\alpha,\alpha') \in Relable_G$.

Example 3.4

Consider a simple grammar $G=(\{S,A,B,C,D\},\{a,b,c\},P,S)$, where $P=$ $\{S \rightarrow aSA|bCB|c,\ A \rightarrow a,\ B \rightarrow b,\ C \rightarrow aDA|bCB|c,\ D \rightarrow c\}$. Figures 3.3 and 3.4 illustrate the cover graph C_G and its characteristic cover graph CC_G, respectively. Here, we have $Relabel_G = \{(S,X_S), (SA,L_{X_S}), (CB,X_C), (A,X_A), (CBB,L_C), (DAB,X_D), (AB,L_{X_A}), (B,X_B), (BB,L_{X_B}), (F,F)\}$.□

Definition (Minimal complete subgraph : g_X)

Let $CC_G=(N'_G,E'_G,\Sigma)$ be the characteristic cover graph of $G=(N,\Sigma,P,S)$. A path is called *α-path* if it begins with an edge labeled α. For each complete node X in $N'_G-\{F\}$, construct a graph g_X from CC_G in the following way:

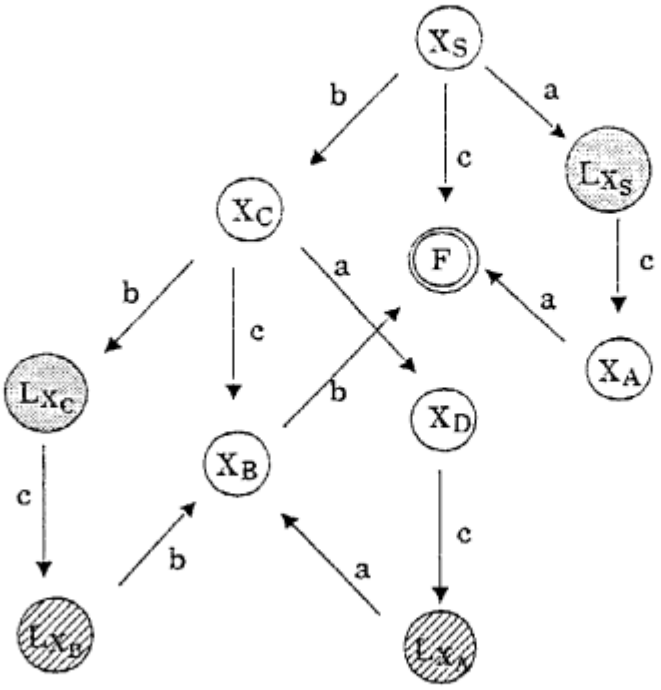


Figure 3.4 Characteristic cover graph of G: CC_G

(1) find a node Y with the properties that (i) for all $a \in \Sigma$, each a-path (if exists) from a complete node X ends up at Y(Y is called the *subfinal node* of g_X, and the shortest path among all a-paths from X to Y is called the *key path* for X), and (ii) for all $a \in \Sigma$, each a-path from X to Y(if exists) is the shortest one,

(2) if there is any complete node Z with degree(Z)$\geq$2 between X and Y on each a-path , then relabel it with L_Z (to make it an incomplete node) and link it to the nearest node Z' by a dotted arrow, provided that a path from Z to Z' is the key path for Z on the a-path,

(3) otherwise, if there is an incomplete node L_Z between the two, then link it to the node U by a dotted arrow, where a path from L_Z to U is isomorphic to the key path for Z.

Such a subgraph g_X is called *minimal complete subgraph for X*.

Example 3.5

Given a simple grammar G considered in Example 3.4, the minimal complete subgraphs for X_S and X_C are given in Figure 3.5.

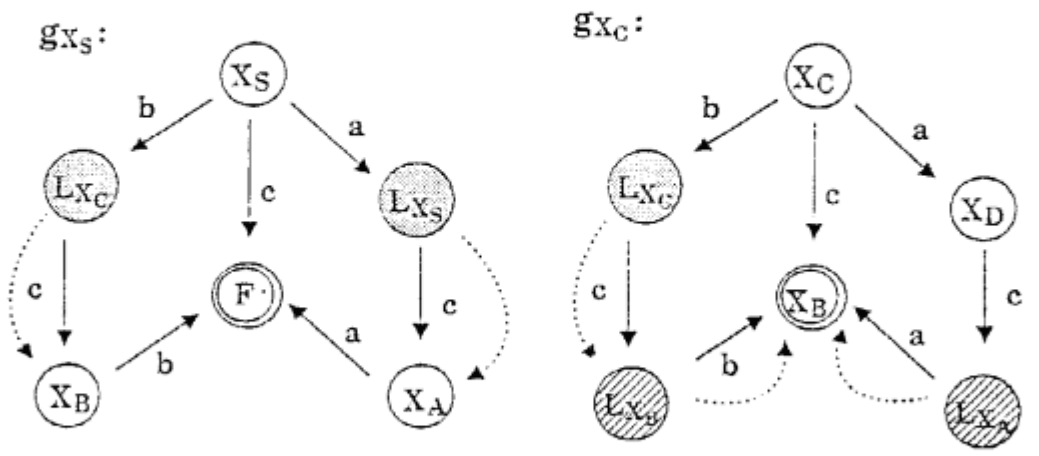


Figure 3.5 Minimal complete subgraphs

3.3 Reconstructing Simple Grammar from Characteristic Cover Graph

We observe that the C_G and CC_G have the following properties which assures that the CC_G together with the node relabeling information preserves the complete information on the grammar G.

[Observation 1]

For a given simple grammar $G=(N,\Sigma,P,S)$, let $C_G=(N_G,E_G,\Sigma)$ and $CC_G=(N'_G,E'_G,\Sigma)$. Then, it holds that

(1)for any rule $A\rightarrow\alpha$ in P, there is a path from the initial node to the final node of C_G on which the rule $A\rightarrow\alpha$ is used at least once,

(2) for any A in N, there is exactly one subgraph h_A up to isomorphism embedded in C_G which corresponds to the minimal complete subgraph g_{X_A} ($X_A\in N_G'$) in CC_G, where $(A,X_A)\in Relabel_G$,

(3)in each h_A, an incomplete node $X\alpha$(if any) is associated with the dotted arrow labeled L_X linking $X\alpha$ to α, which means that any word in L(X) could be generated by a path extended between the two nodes,

(4) the existence of an incomplete node with the label L_X in CC_G implies that of a recursive nonterminal X in N, and vice verse,

(5) for G there uniquely exist C_G and CC_G.

[Reconstructing simple grammar from CC_G]

It is clear that CC_G together with $Relabel_G$ is sufficient for reconstructing the original grammar G.

Now, we shall show that from only the characteristic cover graph CC_G of a simple grammar G one can construct a simple grammar G'(not necessarily G but) equivalent to the original grammar G.

Suppose that the $CC_G=(N'_G,E'_G,\Sigma)$ is given, and consider the set $N'=\{X'\in N'_G | X'$: complete node $\}-\{F\}$. We shall construct a context-free grammar $G'=(N', \Sigma,P',X_S)$ as follows.

Given an X' in N' such that degree(X')$\geq$2, let $g_{X'}$ be the minimal complete subgraph for X'. For each a in Σ, let p_a be the a-path beginning with X' and ending with the subfinal node Y of $g_{X'}$(if any). Further, for each complete node X between X' and Y, consider a subpath t_a of p_a beginning with X and ending with Y.(Note that it is possible that X=X', which then implies that $p_a=t_a$.) We have three cases:

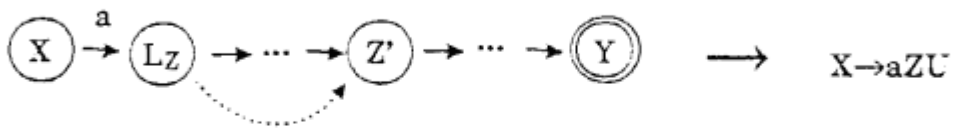
[Case I] The edge labeled a from X goes to an incomplete node labeled L_Z and the dotted arrow from L_Z links to a node Z' *before* the subfinal node Y of p_a (see Figure 3.6 (a)); Then, construct a rule :

$$X\rightarrow aZU,$$

where U comes from the label for the subfinal node Z' of g_Z. (Note that X,Z and U are not necessarily different.)

[Case II] Either the edge labeled a from X goes to a complete node labeled Z before the subfinal node Y, or the edge labeled a from X goes to an incomplete node labeled L_Z and the dotted arrow from L_Z links to the subfinal node Y of p_a (see Figure 3.6 (b)); Then, construct a rule :

(a) t_a:

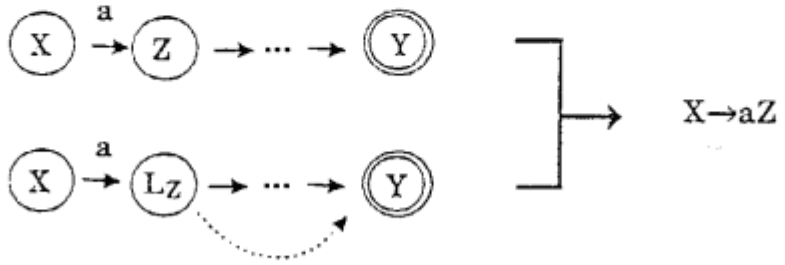


where $Z' = U$ or $= L_U$

(b) t_a:

(b)-1

(b)-2



(c) t_a:

Figure 3.6 Rule reconstruction

$$X \rightarrow aZ.$$

(Note, again, that X and Z are not necessarily different.)

[Case Ⅲ] The path consists of the node X and the subfinal node Y.(see Figure 3.6 (c)); Then, just construct a rule:

$$X \rightarrow a.$$

(Note that in the above construction, if degree(S) = 1, then we take g_S as t_a and apply the construction rules according to cases Ⅰ, Ⅱ and Ⅲ.)

Now, we shall show the following:

Lemma 3.3

Given the CC_G of a simple grammar $G=(N,\Sigma,P,S)$, let $G'=(N',\Sigma,P',X_S)$ be a grammar constructed from CC_G in the manner described above. Then, it holds that $L(G)=L(G')$.

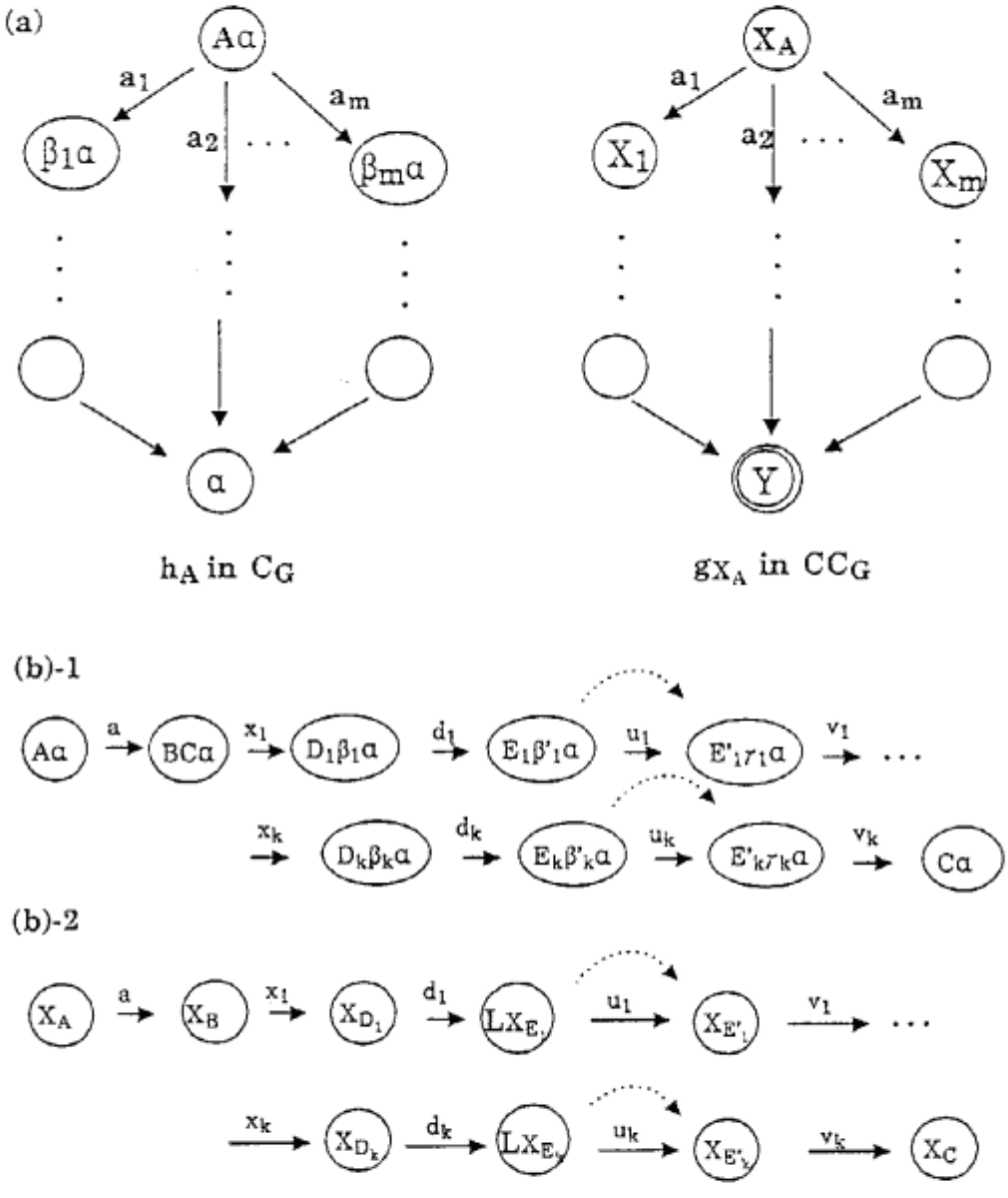


Figure 3.7 h_A and its corresponding g_{X_A}

Proof. For a complete node $A\alpha(A\in N,\alpha\in N^*)$ in C_G, consider a subgraph h_A of C_G which corresponds to the minimal complete subgraph g_{X_A} in CC_G. (See Figure

3.7.) We show by the induction on n(= the length of a word w) the following claim that for any A in N and w in Σ^*, w is in L(A) iff w is in $L(X_A)$, leading immediately to that L(G) = L(G'). [Base step:n = 1] By definition, a rule A—a is in P iff there is a path :$X_A \rightarrow^a Y$ in g_{X_A}, i.e., X_A—a is in P'. Hence, w is in L(A) iff w is in $L(X_A)$. [Induction step] Suppose that the claim holds for each j less than n. (Case 1) Suppose that A—aB is in P, $B \Rightarrow^* w'$, and $w = aw' \in \Sigma^*$. Then, there is an a-path in h_A: $A\alpha \rightarrow^a B\alpha \rightarrow \cdots \rightarrow \alpha$. Let $X_A \rightarrow^a X_1 \rightarrow \cdots \rightarrow Y$ be the corresponding a-path in g_{X_A}, where $X_1 = L_{X_B}$ or X_B and $(\alpha, Y) \in Relabel_G$. In either case, X_A—aX_B is uniquely in P'. By the induction hypothesis, since $w' \in L(B)$ iff $w' \in L(X_B)$, we have $w \in L(A)$ iff $w \in L(X_A)$. (Case 2) Suppose that A—aBC is in P, $B \Rightarrow^* w_1$, $C \Rightarrow^* w_2$ and $w = w_1 w_2 \in \Sigma^*$. Then, there is an a-path in h_A: $A\alpha \rightarrow^a BC\alpha \rightarrow^b \cdots \rightarrow C\alpha \rightarrow^c \cdots \rightarrow \alpha$. Let $X_A \rightarrow^a X_1 \rightarrow^b \cdots \rightarrow X_2 \rightarrow^c \cdots \rightarrow Y$ be the corresponding a-path in g_{X_A}, where $X_1 = L_{X_B}$ or X_B, $X_2 = L_{X_C}$ or X_C and $(\alpha, Y) \in Relabel_G$ Suppose that degree(B) ≧ 2 or BCα is redundant. Then, from the way of constructing P', $X_1 = L_{X_B}$ and a dotted arrow links X_1 to X_2. Hence, X_A—aX_BX_C is in P'. By the induction hypothesis, since $w_1 \in L(B)$ iff $w_1 \in L(X_B)$, and $w_2 \in L(C)$ iff $w_2 \in L(X_C)$, we have $w \in L(A)$ iff $w \in L(X_A)$. Suppose that degree(B) = 1 and BCα is complete. Further, suppose that a subpath from Aα to Cα contains k occurrences of nodes with dotted arrows for some k ≧ 0.(See (b)-1 of Figure 3.7.) The corresponding path in g_{X_A} is shown in (b)-2 of Figure 3.7. (It should be noted that each a-path from Aα to α contains either complete nodes with degree 1 or nodes with dotted arrows.) In the sequence of nodes in (b)-1 B, $D_i (1 \leqq i \leqq k)$ are complete nodes with degree 1, and each $E_i (1 \leqq i \leqq k)$ is incomplete or redundant, and v_i may be equal to $x_{i+1} (x_i, v_i \in \Sigma^*)$. Since $B \Rightarrow^* w_1$, there exist z_i in $L(E_i)$ such that $w_1 = x_1 d_1 z_1 \cdots x_k d_k z_k v_k (d_i \in \Sigma, z_i \in \Sigma^*)$. Now, from the way of constructing P', $X_{D_i} \rightarrow d_i X_{E_i} X_{E'_i} \in P' (1 \leqq i \leqq k)$ and the derivations $X_B \Rightarrow^* x_1 X_{D_1}$, $X_{E'_i} \Rightarrow^* v_i X_{D_{i+1}} (1 \leqq i \leqq k\text{-}1)$ and $X_{E'_k} \Rightarrow^* v_k X_C$ are uniquely realized in G'. Further, by the induction hypothesis, $z_i \in L(E_i)$ iff $z_i \in L(X_{E_i})$. Hence, we have : $X_B \Rightarrow^* x_1 d_1 z_1 \cdots x_k d_k z_k v_k X_C$, i.e, $X_B \Rightarrow^* w_1 X_C$. Since, by the induction hypothesis $w_2 \in L(C)$ iff $w_2 \in L(X_C)$, we eventually have : $w \in L(A)$ iff $w \in L(X_A)$. □

Note. The grammar G' reconstructed from CC_G is equivalent to the original G, but not necessarily isomorphic to G.(See, i.e., Example 3.6 below.)

Example 3.6

Taking, again, the characteristic cover graph considered in Example 3.4 and applying the procedure mentioned above, the grammar G' is obtained: G'= ($\{X_S,X_A,X_B,X_C,X_D\}$, {a,b,c}, $\{X_S \to aX_SX_A|bX_CX_B|c$, $X_C \to aX_D|bX_CX_B|c$, $X_A \to a$, $X_D \to cX_A$, $X_B \to b\}$, X_S). In fact, it is easy to see that L(G)=L(G') holds, but G' is not isomorphic to G. □

[Important Notes]

(1) Returning to the definition of the cover graph C_G, we remember that if a nonterminal A had already appeared as the left-most one of some node, then the node Au was extended by making a son node for "*only the SH-rule of A*". From the ways of constructing CC_G and of reconstructing an equivalent grammar described above, however, we note that the italic part above can be replaced with the statement that "*any rule* $A \to a\alpha$, *where* a *is in* Σ_A", because for a recursive nonterminal A we can recover all the information on A from the minimal complete subgraph g_{X_A}.

(2) In the manner of reconstruction, we actually needed minimal complete subgraphs g_{X_A} for only complete nodes X_A such that degree(A) is greater than 1. This is due to the following. Suppose that degree(A) =1 ($A \to a\alpha \in P$) and A is recursive. Then, since by the assumption of G $L(A) \neq \emptyset$, there must exist a nontermnal B such that $A \Rightarrow^* xB$ ($\exists x \in \Sigma^*$), $L(B) \neq \emptyset$, and degree(B)$\geq$2. Further, since A is recursive, B is also recursive. Hence, the minimal complete subgraph g_{X_B} contains all the information on $A \to a\alpha$. Suppose that degree(A)=1 and A is not recursive. Let z be one of the shortest words in $L(A)(\neq \emptyset)$. Consider a derivation $S \Rightarrow^* x C\alpha \Rightarrow^* xyA\alpha' \Rightarrow^* xyz\alpha'$ (where x, y:shortest). Then, if for all C appearing between S and A, degree(C)=1, then the information on $A \to a\alpha$ is taken into the reconstruction procedure, because by definition A appears somewhere in the subpath of $t_a = g_{X_S}$. Otherwise, there exists C such that

degree(C) ≥ 2 and it is assured that the information on $A \rightarrow aa$ is also contained in gx_C. □

Thus, we have a good reason for asserting the usefulness of the notion of CC_G. Actually, the inference algorithm we shall describe in the next section works such that it may infer the CC_G for some simple grammar G generating the unknown L.

4. Inductive Inference of Simple Languages

4.1 Problem Setting

In the problem setting we deal with in this paper, the Teacher is expected to have the following abilities on the target language L which include a special type of membership oracle stronger than the membership oracle in the usual sense and an oracle for equivalence checking.

Definitions(Prefix-membership oracle/Derivative oracle)

Given a (target) simple language L, *Teacher* is assumed to have the following abilities:(1) the prefix-membership oracle, (2) the derivatives oracle. *The prefix-membership oracle(PMO) for L* takes as an input a query of the form "w?" and produces as an output a string "wx" if there is such an x that is one of the shortest strings among all with the property that wx is in L, or "*No*" otherwise. *The derivative oracle (DEO) for L* takes as an input two pairs of strings (u_1,v_1), (u_2,v_2) and produces as an output "*Yes*" if $u_1\backslash L/v_1 = u_2\backslash L/v_2$ or "*No*" otherwise.

4.2 Inference Algorithm

We shall show that using the prefix-membership oracle for an unknown language L as well as the derivative oracle for L, one can eventually construct the characteristic cover graph CC_G for some simple grammar G generating L. To do so, we need to introduce the notion of a node characterization table which is similar to the observation table used in [An87].

Definition (Node Characterization Table:NCT)

Let P_S be a positive sample set of L. A *node characterization table of L* (NCT_L) consists of three components Prefix(P_S), p-Suffix(P_S)($=$Suffix(P_S)$-P_S$, i.e., a finite set of all suffixes of strings in P_S except for elements of P_S), and a finite function T_L mapping from Prefix(P_S)·p-Suffix(P_S) to {0,1} whose interpretation is that $T_L(u)=1$ iff u is a member of the language L. An NCT_L is sometimes denoted by $[P_S,T_L]$. When L is clear from the context, we simply write NCT and $[P_S,T]$, respectively.

An NCT is visualized as a two-dimensional array with rows labeled by elements of Prefix(P_S) and columns labeled by elements of p-Suffix(P_S), with the entry for row s and column e equal to $T_L(s\cdot e)$.

The NCTs are eventually used to construct the characteristic cover graph of some simple grammar G such that $L=L(G)$. Rows labeled by elements of Prefix(P_S) are the candidates for nodes of the graph being constructed, and columns labeled by elements of p-Suffix(P_S) correspond to distinguishing experiments for these nodes.

Example 4.1

Let L be a simple language generated by a simple grammar G given in Example 3.4. Suppose that a positive sample set P_S is given as {c, bcb, bacab, bbcbb}. Then, a node characterization table $[P_S,T_L]$ is given in Table 4.1. □

Definitions(Base graph/Complete node/Minimal complete subgraph)

Let $[P_S,T]$ be an NCT of a simple language L over Σ. With the NCT, we associate a finite graph $g=(Node,Edge,\Sigma)$ as follows: Node $=$ Prefix(P_S)/$\equiv$, where $u\equiv v$ iff row(u) $=$ row(v). An equivalence class including u is denoted by [u]. The equivalence class S($=[\varepsilon]$) provides the initial node of g, while the equivalence class F($=[w]$, for $w\in P_S$) the final node of g. Let [u] and [v] be in Node, then $[u]\xrightarrow{a}[v]\in$Edge iff $\exists u'\in[u]$, $v'\in[v]$ s.t. $v'=au'$. A graph g is called a *base graph* obtained from $[P_S,T]$.

N\E	ε	ab	cab	b	cb	bb	cbb	bcbb	acab
ε	0	0	0	0	0	0	0	0	0
b	0	0	0	0	1	0	0	1	1
c	1	0	0	0	0	0	0	0	0
ba	0	0	1	0	0	0	0	0	0
bb	0	0	0	0	0	0	1	0	0
bc	0	0	0	1	0	0	0	0	0
bac	0	1	0	0	0	0	0	0	0
bbc	0	0	0	0	0	1	0	0	0
bcb	1	0	0	0	0	0	0	0	0
baca	0	0	0	1	0	0	0	0	0
bbcb	0	0	0	1	0	0	0	0	0
bbcbb	1	0	0	0	0	0	0	0	0

Table 4.1 Node characterization table [P_S,T_L]

In a base graph g a node X=[u] is *complete* if for any "a" in Σ whenever the answer of PMO to a query "ua?" is "Yes", the a-path from X is always contained in g, otherwise a node X is called *incomplete.*

Let X be a complete node in a base graph g. Then, consider a node Y in common on which for all a in Σ each a-path from X in g meets. A graph which consists of X, Y and every other intermediate node in each a-path of the interval is called the *minimal complete subgraph for X*, and is denoted by g_X. The *shortest* (*X*) denotes the shortest path from X to Y in g_X (see Figure 4.1). Thus, it should be noted that for any complete node X there uniquely exist g_X and shortest(X) for some Y in g.

Definition(Lexicographic order ⋖ /First occurrence)

We define the *lexicographic order* ⋖ as follows: for α,β in a base graph such that $S \rightarrow^{u} \alpha$, $S \rightarrow^{v} \beta$, $\alpha \lessdot \beta$ iff lg(u)<lg(v) or [lg(u)=lg(v)&u<v in the usual alphabetical order], where lg(x) denotes the length of x.

A complete node is said to be *of the first occurrence* iff among all nodes whose minimal complete subgraphs are isomorphic, it appears first in the lexicographic order. An important note is that a complete node which is not of the first

Node	N\E	ε	b	ab	bb	cb	cab	cbb	bcbb	acab
S=[ε]	ε	0	0	0	0	0	0	0	0	0
X_1=[b]	b	0	0	0	0	1	0	0	1	1
X_2=[ba]	ba	0	0	0	0	0	1	0	0	0
X_3=[bb]	bb	0	0	0	0	0	0	1	0	0
X_4=[bc]	bc bbcb	0 0	1 1	0 0	0 0	0 0	0 0	0 0	0 0	0 0
X_5=[bac]	bac	0	0	1	0	0	0	0	0	0
X_6=[bbc]	bbc	0	0	0	1	0	0	0	0	0
F=[c]	c bcb bacab bbcbb	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Table 4.2 Node Classification for $[P_S,T_L]$

occurrence is just corresponding to a redundant node previously introduced in reference to a cover graph.

Example 4.2

(1) Taking the NCT=[P_S,T] in Example 4.1 as an instance, we have a base graph g depicted in Figure 4.1.

(2) In the table 4.2, nodes(rows) are classified into 8 blocks according to their values as row vectors : the first block represents the initial node, the last block corresponds to the final node, and each of other blocks corresponds to each node. From the table, it is seen that , for instance, node X_1 connects X_2 with the edge labeled a, or node X_4 connects node F with the edge labeled b, and so forth.

(3) The minimal complete subgraphs g_{X_1} and g_{X_4} are shown in Figure 4.1. The minimal complete subgraphs of nodes X_4 and X_6 are isomorphic as digraphs with colored edges, and X_4 is of the first occurrence, while X_6 is not.

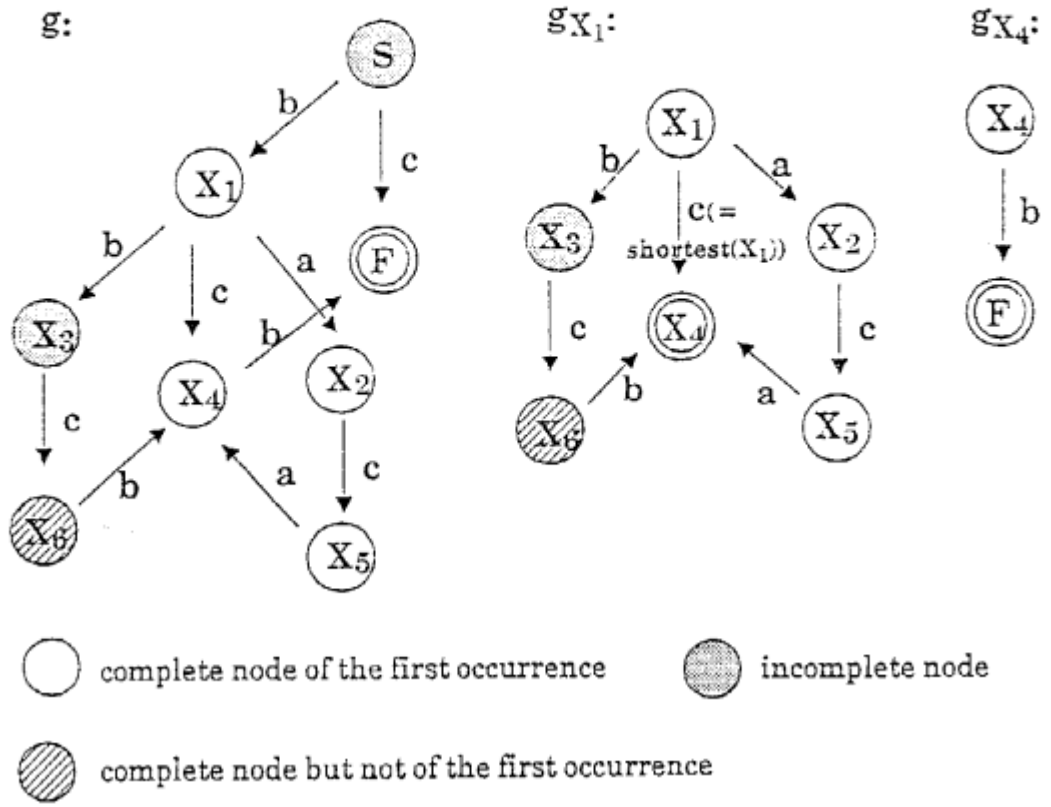


Figure 4.1 Base graph and minimal complete subgraphs

[Observation 2]

From the definition of $[P_S,T_L]$, it is easily seen that (1) a row for the prefix "ε" gives the initial node S, (2)all rows for the elements of P_S form an identical block corresponding to the final node F, (3) each row for each proper prefix gives an intermediate node in the path from S to F, (4) for any column for "v" there exists at least one row "u" such that $T_L(uv)=1$, i.e., uv is in L, (5) for any "u" in P_S and "v" in p-Suffix(P_S), $T_L(uv)=1$ if $v=\varepsilon$, $T_L(uv)=0$ if $v\neq\varepsilon$. (Prefix-free property of L).

[Inductive Inference Algorithm: IIA]

(Preliminary Assumption)

The terminal alphabet $\Sigma=\{a_1,...,a_m\}$ is fixed and the alphabetical order is assumed. The target simple language L is also fixed.

(Notation)

P_S: the set of positive examples of L obtained during the inference process.

COM: the set of complete nodes of the *first occurrence* in a base graph.

INC: the ordered set of incomplete nodes and complete nodes of non-first occurrence in a base graph, where the lexicographic order $\lessdot$ is assumed.

[u]: an element of Node in a base graph, where $S \rightarrow [u] \rightarrow vF$, for some v in Σ^*.

v([u]): m-dimensional row vector $(i_1,...,i_m)$ such that for $1 \leqq \forall j \leqq m$, $i_j=1$ if the answer of the query "ua_j?" is "Yes", and $i_j=0$ otherwise.

Y(u): the set of "yes" answers of *[u]-queries to PMO*, i.e., the set of "yes" answers of PMO to the queries "ua?" for all a in Σ.

after(X): the word of the shortest path from X to F in a base graph.

backbone(X,α): the word w(=shortest(α)) such that w $\in$Prefix(after(X)), where α$\in$COM.

tail(X,α): the string z such that after(X)=backbone(X,α)z.

IDENTIFIED: the set of incomplete and redundant nodes which have been already identified as some complete node.

BEFORE(X): the set of complete nodes appearing before the occurrence of X in the lexicographic order in a base graph.

Now, we need some subprocedures:

(1) **compatibility_check:**

Input: X$\in$INC, α$\in$COM

Output: "True" or "False"

Procedure: if shortest(α) $\in$Prefix(after(X)) and v(X)=v(α),

then return "True" else return "False".

/*Given a node X in INC and a node α in COM, the procedure checks if α is a valid candidate for identifying X or not, whose execution always precedes that of

"identification_check" below, i.e, the compatibility is a necessary condition for identifiability.*/

(2) identification_check:

Input: $[u] \in INC$, $\alpha = [u'] \in COM$

Output: "True" or "False"

Procedure: Let $v = tail([u], \alpha)$ $v' = tail([u'], \alpha)$. Ask DEO to check if $u \backslash L/v = u' \backslash L/v'$ or not;

if the answer is "Yes" then return "True"

else return "False".

/*Given a node [u] in INC and a node α in COM, the procedure tests if [u] is identifiable with α or not by asking DEO.*/

(3) extend_base_graph:

Input: $X \in INC$ and P_S

Output: a base graph $g = (Node, Edge, \Sigma)$ with some additional information on COM,INC,BEFORE

Procedure: make X-queries to PMO;

$P_S \leftarrow P_S \cup Y(X)$;

$N \leftarrow N \cup Prefix(Y(X)), E \leftarrow E \cup p\text{-}Suffix(Y(X))$;

construct $[P_S, T]$ from P_S, N, E;

create a base graph $g = (Node, Edge, \Sigma)$ obtained from $[P_S, T]$;

let New_nodes(X) be the set of nodes newly created by extending X;

using PMO, for all $Y \in$ New_nodes(X) check if Y is complete or not;

$COM \leftarrow COM \cup \{X\} \cup \{Y \in$ New_nodes(X)|Y:complete of the first occurrence$\}$;

$INC \leftarrow INC \cup \{Y \in$ New_nodes(X)|Y:incomplete or complete of the non-first occurrence$\} - \{X\}$, where we suppose that INC is created so that all elements are sorted in the *lexicographic order* defined above ;

for all X ∈INC, compute BEFORE(X);

/*Given an incomplete node X in INC, the procedure constructs an NCT by making X-queries to PMO, then creates a base graph g. At the same time, it updates the additional information on the sets COM, INC and BEFORE(X).*/

Definition(Minimum grammar for L)

A simple grammar G is *minimum for L* iff L=L(G) and the number of nonterminals of G is the smallest.

Notes.

(1) Remember the convention previously described : all simple grammars we are dealing with are assumed to be in 2-standard form.

(2) For a given L, a minimum grammar for L is, in general, not unique.(See, e.g., the grammars considered in Examples 3.4 and 3.6.)

[The outline of the IIA]

We outline the behavior of the algorithm IIA. After initializing all parameters involved, IIA first produces the minimal complete subgraph for S(initial symbol) by calling **extend_base_graph**, i.e., by making [ε]-queries to PMO and creating NCT=[P_S,T], which results in producing the first base graph g obtained from [P_S,T], where P_S(=Y([ε])) is the set of "yes" answers of [ε]-queries to PMO. At the same time, COM, INC, and BEFORE are computed. Suppose IDENTIFIED=INC. Then, for each X in INC−IDENTIFIED, IIA tries to identify X as one of elements from COM by using **compatibility_check** and **identification_check**. In the process, if there exists a node X in INC which is not identified as any of complete nodes in COM, then by calling **extend_base_graph** IIA extends X to make it complete. This is justified by the fact that the existence of an incomplete node un-identified by any complete node in a base graph implies that the (intermediate) graph is not fully extended yet for the characteristic cover graph relevant to the unknown language.

The Algorithm :IIA

```
initialize all parameters involved, i.e.,
P_S ←Ø, COM←Ø, INC←Ø , IDENTIFIED←Ø and N:=E:={ε};
g:= extend_base_graph([ε],Ø);
Repeat
      /* outer while loop */
      while IDENTIFIED≠INC, i.e., there exists a node X in INC not
            identified yet in g do
            begin
               take the top element X in INC−IDENTIFIED;
               /* inner while loop */
               while BEFORE(X)≠Ø do
               begin
                  take α∈BEFORE(X);
                  if compatibility_check(X,α)="False"
                  then BEFORE(X)←BEFORE(X)−{α};
                  else   if identification_check(X,α)="True"
                         then X:=α; BEFORE(X)←BEFORE(X)−{α};
                              and IDENTIFIED←IDENTIFIED∪{X};
                         else BEFORE(X)←BEFORE(X)−{α};
               end
               if X∉IDENTIFIED
               then g:= extend_base_graph(X,P_S);
            end
Until all nodes in INC of g are identified
make a conjecture grammar G from canonical cover graph g';
Halt and output G
```

The basic idea is that IIA behaves such that it may conjecture a partial characteristic cover graph of a minimum grammar G for the unknown language

and construct the corresponding base graph step by step in exactly the same manner as the procedure for constructing cover graph of G does.

As we have seen in the previous section(3.3), once the conjectured characteristic graph of some simple grammar G is obtained from a base graph, one can construct a grammar G' equivalent to G in the manner described above.

4.2.2 Correctness and Complexity of the Algorithm

Lemma 4.1(Size of characteristic cover graph)

For a given simple grammar $G=(N,\Sigma,P,S)$, *let* $CC_G=(N'_G,E'_G,\Sigma)$ *and* $t=max\{lg(w_A)|w_A$ *is a shortest word derivable from A in G,* $A\in N\}$ *and* $m=\#\Sigma$, $n=\#N$. *Then, the number of total nodes of* CC_G *is not greater than* $(2t+1)mn$.

Proof. By the definition of the minimum complete subgraph, without taking account of self-loops, for A' in $N'_G-\{F\}$, a path corresponding to "$A\alpha \rightarrow^{a} B C\alpha \rightarrow^{w_B} C\alpha \rightarrow^{w_C} \alpha$" (for $\alpha\in N^*$) in C_G is the possible longest path in $g_{A'}$, which is clearly not greater than $(2t+1)$. Further, since CC_G contains at most n (different) minimal complete subgraphs, the maximum length of all paths in CC_G is not greater than $(2t+1)n$. Clearly, the initial node of each minimal complete subgraph has at most m (different) paths. Hence, the number of total nodes of CC_G is not greater than $(2t+1)mn$.□

Lemma 4.2

Given a target language L, let G be a conjectured grammar produced from IIA. Then, $L(G)=L$ *holds, i.e., IIA infers a correct grammar G for L.*

(Proof Sketch) Starting with extending the initial node S=[ε], first a base graph g which corresponds to the minimal complete subgraph for S is obtained. Then, each node of g is examined and classified into two categories COM and INC. As IIA proceeds, the number of nodes of a base graph g increases. By consulting DEO, IIA extends incomplete nodes (i.e., introduces new nonterminals) only when they are inevitably neccessary, which guarantees the minimality of the resulting intermediate base graph (a subgraph of the characteristic cover graph of some

minimum simple grammar G for L). Here, an important observation is that *IIA extends g in just the same manner as the procedure for constructing a cover graph of a simple grammar does*, from which the termination of the IIA as well as the correctness comes. (In fact, let G be a grammar for L, and m,n, and t be parameters in Lemma 4.1. Then, from Lemma 4.1 and the incremental feature of the algorithm, it follows that before the number of total nodes of g exceeds $(2t+1)mn$, IIA eventually encounters a correct base graph g from which a minimum grammar equivalent to G is constructed.)□

[Time complexity]

Let m, n and t be the parameters in Lemma 4.1. We analyze the time complexity of the algorithm IIA as follows:

①the subprocedure **compatibility_check** takes at most time $O(mnt)$, since $\#Prerix(after(X)) \leq O(mnt)$; ②the subprocedure **identification_check** is due to the rerivative oracle DEO, and we assume it costs a unit (constant) time; ③the subprocedure **extend_base_graph** takes at most time $O(m^2n^4t^2)$, because the highest order of the time complexity of all subroutines involved in it is $O(mn^2t)^2$, which comes from the time for the task of constructing a table $[P_S,T]$.

For the "inner while loop", it takes at most $O(mt) \times O(n) = O(mnt)$. So, [1] the first "outer while loop", whose primary complexity is due to $O(m^2n^4t^2)$ of the last routine **extend_base_graph**, takes at most $O(m^2n^4t^2) \times O(mnt) = O(m^3n^5t^3)$. Further, for each execution of the body for "repeat", the total time requires at most $O(m^3n^5t^3)$. Since the "repeat" occurs at most $O(n)$ times, the total time for "repeat" takes at most $O(m^3n^5t^3) \times O(n) = O(m^3n^6t^3)$, which eventually gives the time complexity of the whole algorithm. (Note that the time for making G from g' which takes at most $O(mnt)$(the size of CC_G) is negligible.)

The total number of queries, on the other hand, is analyzed as follows. First, **compatibility_check** requires at most $O(m)$ queries. Further, **extend_base_graph** requires at most $O(m^2n^4t^2)$ queries for its primary routine

to extend the table $[P_S,T]$.. Hence, for ③ it takes at most $O(m^2n^4t^2)$ queries, and for [1] it takes at most $O(m^2n^4t^2)\times O(mnt)=O(m^3n^5t^3)$ queries. Thus, the number of queries required for each execution of the body for "repeat"([1]) is at most $O(m^3n^5t^3)$, hence totally at most $O(m^3n^6t^3)(=O(m^3n^5t^3)\times O(n))$ queries are required.

Thus, we have:

Theorem 4.1

The algorithm IIA learns the unknown language in time polynomial in k m, n, and t. In particular, it requires at most polynomial number of queries in m, n, and t, where m,n and t are parameters defined in Lemma 4.1.

4.3 Example Runs

Example 4.3

Let L be a simple language over Σ generated by a simple grammar G considered in Example 3.4, where $\Sigma=\{a,b,c\}$ is fixed.

After setting up the starting condition, i.e., initializing all parameters involved, *Learner*(the algorithm) begins with making prefix-membership queries for "a?","b?", and "c?" in this order. Then, *Teacher*(the prefix-membership oracle) responses with the answers "aca", "bcb" and "c", respectively. So, we have $Y([\varepsilon]) = P_S = \{aca,bcb,c\}$. Then, Learner adds each element of Prefix(Y([ε])) to N and each element of p-Suffix(Y([ε])) to E to construct T_1. Further, by asking for "aa?" and "ba?" and receiving the answers "aacaa" and "bacb", respectively, he knows that nodes X_1 and X_2 are incomplete. Similarly, it turns out that nodes X_3 and X_4 are complete, because for any query of the form "acx?"($x\in\{b,c\}$) or "bcx?"($x\in\{a,c\}$) Teacher responses "No". The base graph g_1(=extend_base_graph([ε],∅)) is pictured in Figure 4.2.

At this moment, we have $COM=\{S,X_3,X_4\}$, $INC=\{X_1,X_2\}$ and $BEFORE(X_1)= BEFORE(X_2)= \{S\}$. Further, since shortest(S)=c, $c\in$

		ε	a	ca	b	cb
S	ε	0	0	0	0	0
F	c	1	0	0	0	0
X_1	a	0	0	1	0	0
X_3	ac	0	1	0	0	0
F	aca	1	0	0	0	0
X_2	b	0	0	0	0	1
X_4	bc	0	0	0	1	0
F	bcb	1	0	0	0	0

Table T_1

		ε	a	ca	b	cb	ab	cab	acab	ba	cba	bcba
S	ε	0	0	0	0	0	0	0	0	0	0	0
F	c	1	0	0	0	0	0	0	0	0	0	0
X_1	a	0	0	1	0	0	0	0	0	0	0	0
X_3	ac	0	1	0	0	0	0	0	0	0	0	0
F	aca	1	0	0	0	0	0	0	0	0	0	0
X_2	b	0	0	0	0	1	0	0	1	0	0	1
X_4	bc	0	0	0	1	0	0	0	0	0	0	0
F	bcb	1	0	0	0	0	0	0	0	0	0	0
X_5	ba	0	0	0	0	0	0	1	0	0	0	0
X_7	bac	0	0	0	0	0	1	0	0	0	0	0
X_4	baca	0	0	0	1	0	0	0	0	0	0	0
F	bacab	1	0	0	0	0	0	0	0	0	0	0
X_6	bb	0	0	0	0	0	0	0	0	0	1	0
X_8	bbc	0	0	0	0	0	0	0	0	1	0	0
X_4	bbcb	0	0	0	1	0	0	0	0	0	0	0
F	bbcbb	1	0	0	0	0	0	0	0	0	0	0

Table T_2

Prefix(after(X_1)) and c∈Prefix(after(X_2)), X_1 and X_2 are both compatible with S (**compatibility_check**(X_1,S)=**compatibility_check**(X_2,S)="True"), and backbone(X_1,S)=backbone(X_2,S)=c and tail(X_1,S)=a, tail(X_2,S)=b. To identify an incomplete node X_1=[a], Learner asks Teacher(derivative oracle) if a\L/a=L or not. Since the answer is "Yes", X_1 is identified as S. Similarly, by making a query if b\L/b=L or not, it turns out that X_2 is not identified as S. Hence,

Learner must extend X_2 to make it complete. The answers of X_2-queries to PMO are "bacab", "bbcbb" and "bcb", and we have $Y(X_2)=\{bacab, bbcbb, bcb\}$. Hence, N is incremented by adding $Prefix(Y(X_2))$, and E is also extended by adding p-$Suffix(Y(X_2))$. Thus, an extended table T_2 is obtained.(See Table T_2.) Now, Learner asks for "baa?" and "bab?" to know whether $X_5=[ba]$ is incomplete or not. Since the answer is "No" for both, he knows that X_5 is complete. Similarly,

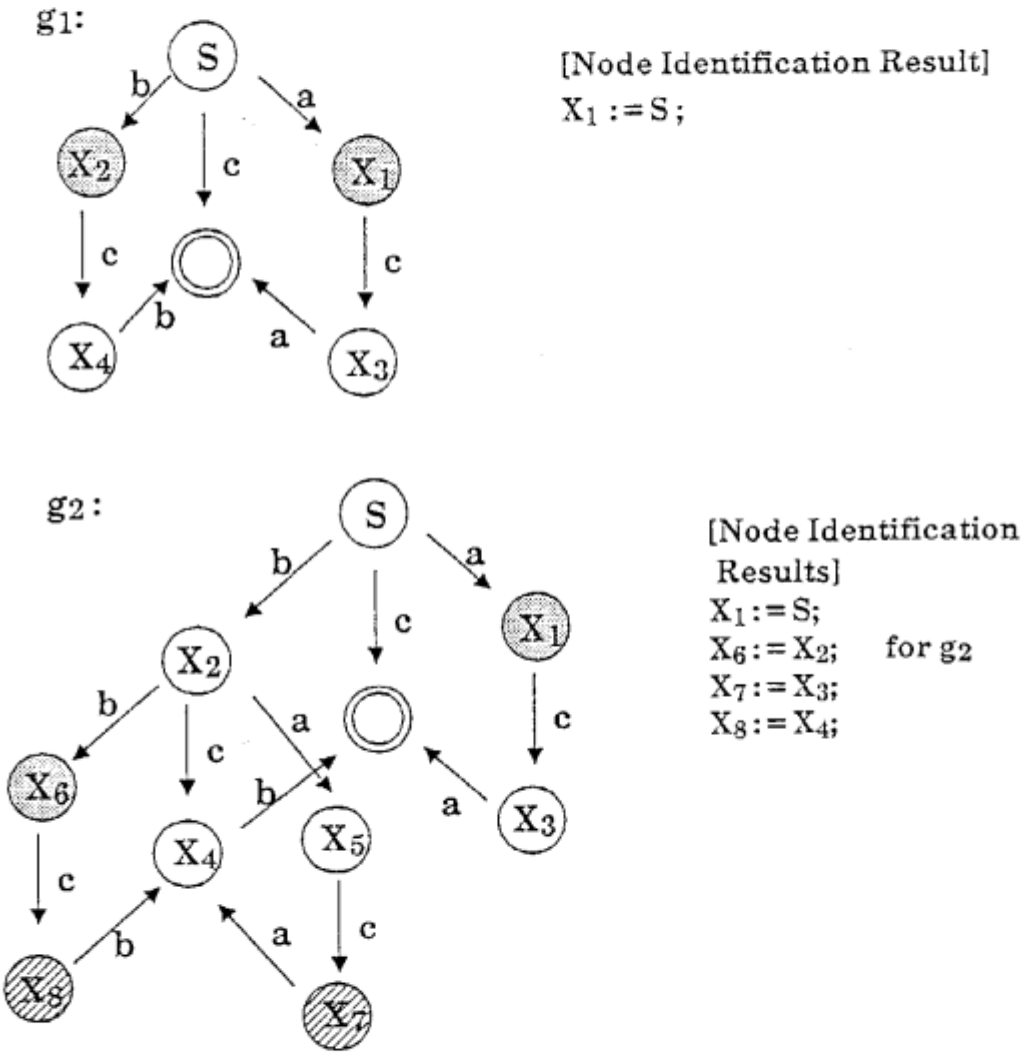


Figure 4.2 Conjectured base graph

it turns out that X_7 and X_8 are complete but not of the first occurrence, while X_6

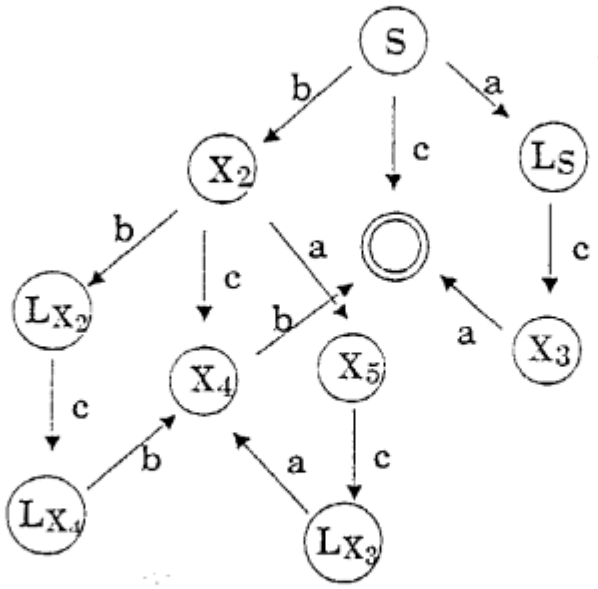
is incomplete. The base graph g_2(=extend_base_graph(X_2,P_S)) created at this moment is given in Figure 4.2. We now have P_S:=bP_{X_2}b$\cup P_S$, P_{X_2}={aca,bcb,c}, COM= {S,X_2,X_3,X_4,X_5}, INC={X_1,X_6,X_7,X_8} and IDENTIFIED={X_1}. Further, BEFORE(X_1)= {S}, BEFORE(X_6)={S,X_2,X_3,X_5}. In order to identify X_6, S is first selected from BEFORE(X_6) and its compatibility is examined by calling compatibility_check(X_6,S). Since shortest(S) = c and c ϵPrefix(after(X_6)), X_6=[bb] is compatible with S. Therefore, the identifiability is checked by calling identification_check(X_6,S). (Note that backbone(X_6,S)=c and tail(X_6,S)=bb.) Since bb\L/bb=L, the subprocedure returns "No". We now have BEFORE(X_6)={X_2,X_3,X_5}. In a similar manner, Learner tries the alternative possibility for the compatibility as well as the identifiablility of X_2 to X_6, and this time X_6 is successfully identified as X_2 because bb\L/bb=b\L/b holds. We now have IDENTIFIED={X_1,X_6}. (Note that X_6 is clearly not compatible with X_3 and although X_6 is compatible with X_5, degree(X_6)=degree(X_5), hence X_5 is dropped out. But, all these are out of question in this case.) Similarly, for X_7,X_8 ϵINC$-$IDENTIFIED it is seen that X_7 and X_8 are identifiable with X_3 and X_4, respectively. Now, since INC=IDENTIFIED hods at this moment, getting out of the outer while loop, Learner produces the conjectured grammar G from its base graph g_2 given in Figure 4.2. The set of rules of G is :

$$S \rightarrow aSX_3|bX_2X_4|c,\ X_2 \rightarrow aX_5|bX_2X_4|c,\ X_3 \rightarrow a,\ X_4 \rightarrow b,\ X_5 \rightarrow cX_3,$$

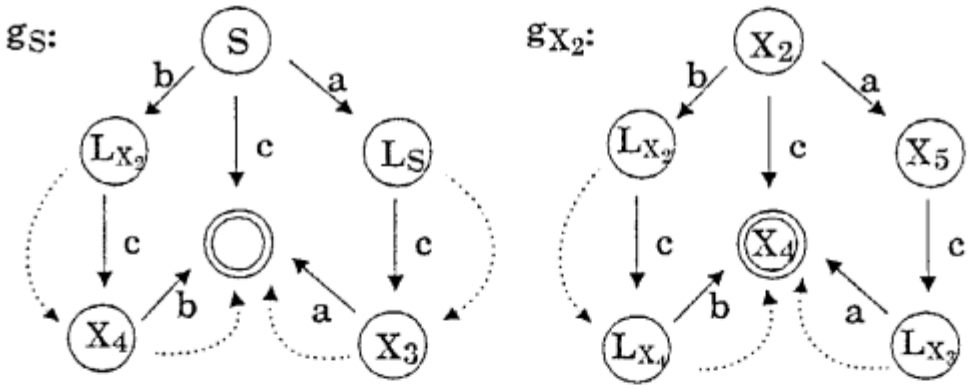
and the inference process terminates. The conjectured characteristic cover graph g'_2 and minimal complete subgraphs involved in g'_2 are given in Figure 4.3.□

Example 4.4

Consider a language L generated by a simple grammar G = ({E},{id,*,+,(,)},P,E), where P={E$\rightarrow$*EE|+EE|(E)|id}. A language L is the set of all arithmetic expressions using operations "*" and "+". (Note that expressions are represented in the prefix notation, and "id", denoting the identifier, is *a terminal symbol* of G. The corresponding simple grammar in 2-standard normal



(a) conjectured characteristic cover graph: g'_2



(b) minimal complete subgraphs for S and X_2

Figure 4.3 Conjecture graphs

form generating L is $G_0 = (\{E,K\}, \{id,*,+,(,)\}, \{E \rightarrow *EE \mid +EE \mid (EK \mid id,\ K \rightarrow)\}, E)$.)
Here, the set {id,(,*,+,)} is assumed to be an ordered set (in this order).

Now, after initializing all the parameters involved, *Learner* begins by making a prefix-membership query "a?" for each symbol a in {id,(,*,+,)} in this

order. Since the *Teacher* answers "id", "(id)", "*idid", "+idid", and "*No*", respectively, Learner creates a table T_1 which gives a base graph g_1 in (a) of Figure 4.4. Now, we have $P_S = Y([\varepsilon]) = \{id, (id), *idid, +idid\}$. Further, by asking for "(*?" and "*(?" and receiving the answers "(*idid)" and "*(id)id", respectively, Learner knows that X_1 and X_2 are incomplete. Similarly, it turns out that X_3 is already complete, because answers to the queries of the form "(idx?", for $x \in \{id,(,*,+\}$, are all "No".

At this moment, we have $P_S := P_S \cup \{"(*idid)","*(id)id"\}$, $COM = \{S,X_3\}$, and $INC = \{X_1,X_2\}$. Since shortest(S) = "id" and "id" $\in$ Prefix(after(X_1)) and "id" $\in$ Prefix(after(X_2)), X_1 (= [(]) and X_2 (= [*] or [+]) are both compatible with S (, while neither is compatible with X_3). Further, tail(X_1,S) = ")" and tail(X_2,S) = "id". So, by asking DEO if $(\backslash L/) = L$ or not, Learner identifies X_1 as S. Similarly, since $*\backslash L/id = L$ holds, X_2 is also identified as S, leading to a conjecture of the characteristic cover graph g'_1 in (b) of Figure 4.4. Thus, a grammar with the set of rules:

$$\{S \rightarrow *SS \mid +SS \mid (S) \mid id\}$$

is eventually produced, and the inference process terminates.□

5. Concluding Remarks

We mention a direct but useful application of the inductive inference of simple grammars. It is well-known that a certain type of syntax for programming languages is defined by context-free grammars, and a numerous work on parsers or compilers for the languages has been reported. Among others, several methods for automatically generating compilers (more precisely, parsers) are discussed. However, the methods proposed so far take as an input a grammar specifying the target language and produce as an output a parser for the language. In contrast to this, our schema for automatically generating parsers is much more "automatic" in the sense that no grammar is necessary and only the knowledge about the target language is required for the prefix-membership oracle and the

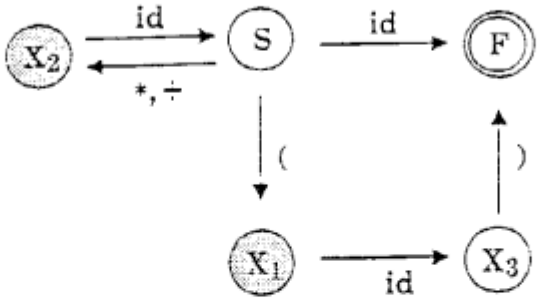
		ε	id	)	id)	idid
S	ε	0	1	0	0	0
F	id	1	0	0	0	0
X_1	(	0	0	0	1	0
X_2	*	0	0	0	0	1
X_2	÷	0	0	0	0	1
X_3	(id	0	0	1	0	0
S	*id	0	1	0	0	0
S	÷id	0	1	0	0	0
F	(id)	1	0	0	0	0
F	*idid	1	0	0	0	0
F	÷idid	1	0	0	0	0

Node characterization table : T_1

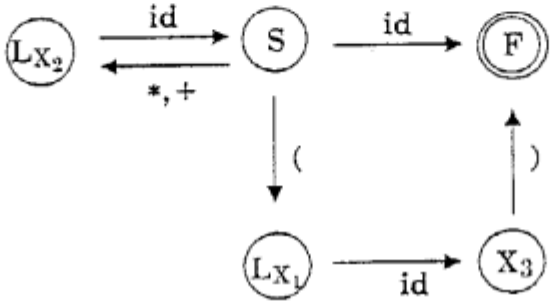
derivative oracle. A diagram for generating a parser from example sentences is pictured in Figure 5.

We have presented an inductive inference algorithm for learning simple grammars in which Teacher is expected to play a role of some kind of oracle more than membership oracle, called prefix-membership oracle. It should be noted that the minimality of the answer string from the oracle plays a significant role, and the discussion based on the similar idea is found in [KK 76]. We believe that the prefix-membership oracle is much easier for users to perform, compared with other problem settings (like an oracle for providing *counter-examples* in [An 87] or *structural examples* consistent with the correct unknown grammar in [CR72],[Fa83],[Sak87]), and is more adequate in the practical applications.

It may be possible to consider a variant of the problem setting and of the algorithm discussed here. For example, the prefix-membership oracle could be replaced with the usual membership oracle at the sacrifice of the increase in the number of queries. Further, as in [An 87], we could employ a problem setting based on the use of queries and counter-examles([Yo88b]).



(a) Conjectured base graph : g_1



Identification result:
$X_1 := S$; $X_2 := S$

(b) Conjectured characteristic cover graph : g_1'

Figure 4.4

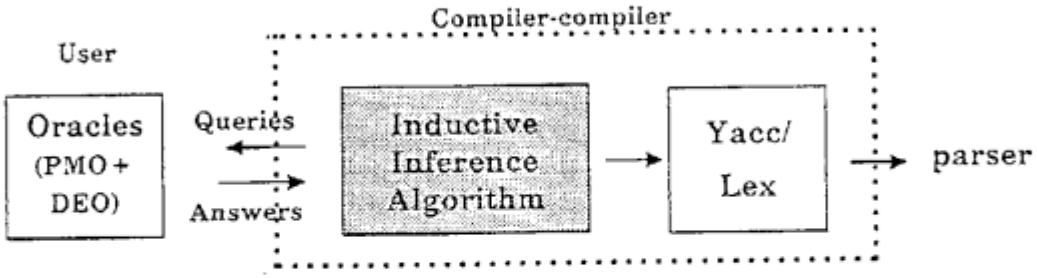


Figure 5 A diagram for compiler-compiler

Finally, it should be remarked that the extension of the algorithm presented here to the inference of the larger class of grammars (such as "strict

deterministic grammars" or "LL(k) grammars" [Ha78]) is important as well as interesting, and actually we are now working on it.

Acknowledgments

The author is grateful to Dr. T. Kitagawa, the president of IIAS-SIS, for ceaseless encouragements. He is also indebted to Dr. H.Enomoto, the director of IIAS-SIS, for providing useful reference papers as well as invaluable advice. Last but not least, discussion with the colleagues Y.Takada, Y.Sakakibara, and H. Ishizaka was very fruitful.

This work is a part of the major research and development of FGCS Project conducted under the program setup by MITI.

References

[An78] Angluin,D., On the Complexity of Minimum Inference of Regular Sets, *Inform. and Control* 39, 337-350 (1978).

[An87] Angluin,D., "Learning regular sets from queries and counter-examples", *Inform. and Computation* 75, 87-106(1987).

[Bi72] Biermann,A.W., An Interactive Finite-State Languages Learner, *Proc. of the First USA-Japan Comput. Conf.*,13-20 (1972).

[CR72] Crespi-Reghizzi,S., An effective model for grammatical inference, in *Information Processing 71*, Elsevier North-Holland, 524-529, 1972.

[ET76] Enomoto,H. and Tomita,E., A Representative Set of Deterministic Finite Automata, *Transactions of IECE*, Vol.59-D, No.9, 660-667 (1976)(in Japanese).

[EDM71] Enomoto,H., Doshita,S. and Matumoto, An estimating method for pushdown automata by finite-state automata, Technical Report of SIG on AL, IECE of Japan, 1971(in Japanese).

[Fa83] Fass,L.F., Learning context-free languages from their structured sentences, *SIGACT News*, Vol.15, No.3, 24-35(1983).

[Go67] Gold,E.M., Language Identification in the Limit, *Inform. and Control* 10, 447-474 (1967).

[Ha78]Harrison,M.A., "Introduction to Formal Language Theory", Addison-Wesley, 1978.

[HU69] Hopcroft,J.E. and Ullman,J.D., "Formal Languages and Their Relation to Automata", Addison-Wesley, 1969.

[Is88] Ishizaka,H., Inductive Inference of Regular Languages Based on Model Inference, in *LNCS*, Springer(to appear), 1988.

[Ki86] Kitagawa,T., "Statistical Information in Inference Process and Data Analysis", Research Report 66, Intern. Inst. for Advanced Study of Soc. Inform. Sci., FUJITSU LIMITED, 1986.

[KK77] Knobe,B. and Knobe,K., A method for Inferring Context-free Grammars, *Inform. and Control* 31, 129-149 (1976).

[KH66] Korenjak,A. and Hopcroft,J.E., Simple Deterministic Languages, in *Proc. of 7th Annual IEEE Conference on Switching and Automata Theory*, 36-46, 1966.

[Sa73] Salomaa,A., "Formal Languages", Academic Press, 1973.

[Sak87] Sakakibara,Y., Inferring Parsers of Context-free Languages from their Structural Examples, Research Report 79, Intern. Inst. for Advanced Study of Soc. Inform. Sci., FUJITSU LIMITED, 1987.

[Sh81] Shapiro,E., "Inductive Inference of Theories from Facts", Technical Report 192, Dept. of Comput. Sci., Yale University, 1981.

[Tak87] Takada,Y., A Constructive Method for Inductive Inference of Linear Languages Based on Control Set, Research Report 78, Intern. Inst. for Advanced Study of Soc. Inform. Sci., FUJITSU LIMITED, 1987

[Ta82] Tanatsugu,K., Inductive Inference of harmonic linear languages, *Intern. J. of Comput. and Inf. Sci.* 6, No.1, 83-93 (1982).

[Ta87] Tanatsugu,K., A grammatical inference for context-free languages based on self-embedding, *Bulletin of Informatics and Cybernetics* Vol.22, No.3-4, 149-164, 1987.

[Wh77] Wharton,R.M., Grammar Enumeration and Inference, *Inform. and Control* 33, 253-272 (1977).

[Wo73] Wood,D., Some Remarks on the KH Algorithm for s-Grammars, *BIT* 13, 476-489, 1973.

[Yo88a] Yokomori,T., Inductive Inference of Context-free Languages Based on Context-free Expression , *Intern. J. of Comput. Math. 24*(to appear), 1988.

[Yo88b] Yokomori,T., Identifying Simple Languages from Queries and Counter-examples, in preparation, 1988.