CS 125
MEMO AI-89

GRAMMATICAL COMPLEXITY AND INFERENCE

BY

JEROME A. FELDMAN
JAMES GIPS
JAMES J. HORNING
STEPHEN REDER

SPONSORED BY

ADVANCED RESEARCH PROJECTS AGENCY

ARPA ORDER NO. 457

TECHNICAL REPORT NO. CS 125

JUNE 1969

STANFORD ARTIFICIAL INTELLIGENCE PROJECT

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

DISCLAIMER NOTICE

THIS DOCUMENT IS THE BEST QUALITY AVAILABLE.

COPY FURNISHED CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

CS 125
STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AI-89

JUNE 1969

GRAMMATICAL COMPLEXITY AND INFERENCE

by

Jerome A. Feldman
James Gip
James J. H[illegible]g
Stephe[illegible]der

Computer Science Department
Stanford University

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Department of Defense (SD-183).

Abstract

The problem of inferring a grammar for a set of symbol strings is considered and a number of new decidability results obtained. Several notions of grammatical complexity and their properties are studied. The question of learning the least complex grammar for a set of strings is investigated leading to a variety of positive and negative results. This work is part of a continuing effort to study the problems of representation and generalization through the grammatical inference question. Appendices A and B and Section 2a.0 are primarily the work of Reder, Sections 2b and 3d of Horning, Section 4 and Appendix C of Gips, and the remainder the responsibility of Feldman.

GRAMMATICAL COMPLEXITY AND INFERENCE

BLANK PAGE

1. Preliminaries

1a. Introduction

The problem of generalization (induction, concept formation) has interested workers from a wide range of fields. In this paper, a particular form of generalization, grammatical inference, is discussed. The notion of grammatical complexity is introduced to help measure which grammar is the best one for a given set of strings.

The grammatical inference problem is easy to state; one is interested in algorithms for choosing the best grammar from a given class for a sequence of symbol strings. For example, we would like to discover that the sequence of strings

car, cdr, caar, cdadr, cddadadr, etc.

can be described by the rule: each string is a 'c' followed by any sequence of 'a's and 'd's followed by 'r'. Or in Backus-Naur Form

<string> ::= c <seq> r

<seq> ::= a | d | <seq> a | <seq> d

The question of how to infer a grammar and to measure how well you've done it will be the main topics of this paper.

The grammatical inference problem has received relatively little attention. The main theoretical formulation to date has been that of Gold [67] which will be discussed in Section 3. Solomonoff [64] considers the problem as a special case of sequence extrapolation; we have argued against this notion [Feldman 67] but are indebted to Solomonoff for some of the basic ideas on grammatical complexity in Section 2. There has also been some related work in Computer Science [Amarel 62, London 64] and

Psychology [Miller 6, Suppes 56]. There is, of course, a vast literature on pattern recognition [Uhr 66], but it has been exclusively concerned with pattern descriptions which are structurally simpler than grammars.

Early studies of grammatical inference referred to it as a form of induction. The term "induction" has been used as a description of generalization processes. Unfortunately, it has also been used in dozens of other ways and is threatening to become meaningless. We favor restricting the term "induction" to statistical modes of inference such as those of Solomonoff [64] as is done currently in Philosophy. The particular model which we found most appropriate is the hypothetico-deductive-empirical (HDE) mode of inference. An HDE inference consists of forming hypotheses, deducing conclusions about the data and testing these conclusions for validity. This characterizes the scientific method and is quite close to the "scientific induction" of Lederberg and Feigenbaum [68]. In our case a hypothesis is a grammar rule, a deduction is a derivation, and the data are the sample strings.

The results of this paper are one part of a many-pronged attack on the grammatical inference problem [Feldman 67]. The results here are largely theoretical, but include a heuristic program to infer grammars. Other efforts involve psychological study of human grammatical inference. We also hope to be able to relate theoretical results with the heuristics of the program and to consider how these relate to human learning of language and other theories. To the extent that e.g. pictures [Miller and Shaw 68] are well represented by grammars, the grammatical inference work may be of some practical use in pattern recognition.

1b. Definitions, Notation

This paper makes use of ideas from several research areas, and it is impossible to agree with all their notational conventions. We deviate from the usual formulation of context free grammars in requiring all vocabularies to be subsets of a fixed collection of symbols. There is no loss of generality in doing this, but many results in the literature would require careful consideration of substitution rules [cf. Church 56].

The universal terminal alphabet $\mathcal{T}$ is the set of symbols $\{a, a_1, a_2 \ldots\}$. The universal variable alphabet $\mathcal{V}$ is the set of symbols $\{X = Z, Z_1, Z_2 \ldots\}$.

We will also use the following notational conventions. The string of zero symbols is denoted by e, the empty set by $\emptyset$. If S is any set of symbols, S^* is the set of finite strings of symbols from S and $S^+ = S^* - e$.

A context free grammar (cfg) is a quadruple $G = (V,T,X,P)$ where V,T are finite sets, $V \subset \mathcal{V} \cup \mathcal{T}$, $T = \mathcal{T} \cap V$, $X \in V-T$, and P is a finite set of productions (rules) of the form $Z \to w$, with $Z \in V-T$, $w \in V^*$. In such a production, Z is called the left side and w the right side. We will abbreviate a set of productions $Z \to w_1, Z \to w_2, \ldots, Z \to w_k$ with the same left side as $Z \to w_1 \mid w_2 \mid \ldots w_k$.

If G is a cfg, and $w,y \in V^*$ we write $w \underset{G}{\Rightarrow} y$ if there exists $t \in V^*$, $Z \in V-T$ and w_1, w_2 in V^* such that $w = w_1 Z w_2$, $y = w_1 t w_2$ and the rule $Z \to t$ is in P. The string y is called an intermediate string. The transitive closure of $\underset{G}{\Rightarrow}$ is written $\overset{*}{\underset{G}{\Rightarrow}}$. In either case the subscript "G" may be omitted if there is only one grammar under consideration.

If $w \overset{*}{\underset{G}{\Rightarrow}} y$, $y \epsilon T^+$, we also say there is a derivation of y from w in G . In this case, there is also a derivation of y from w in which each rule has as its left side, the leftmost $Z \epsilon V-T$ of the preceding intermediate string [Ginsburg 66, p. 30]. This leftmost derivation is denoted $d(y,w,G)$, and when $w = X$ will be abbreviated to $d(y,G)$. We will be exclusively concerned with leftmost derivations. If $d(y,w,G) = \langle p_1, p_2, \ldots, p_k \rangle$ with $p_j \epsilon P$ we define the derivation length $\ell_d = k$. The length $\ell(y)$ is the number of symbols in y .

The language $L(G)$ generated by a cfg $G = (V,T,X,P)$ is defined by

$$L(G) = \{y \mid y \epsilon T^+ \text{ and } X \overset{*}{\underset{G}{\Rightarrow}} y\} .$$

We will sometimes omit mention of the grammar. The definition implies that we will be dealing with only e-free languages. With this restriction and some well-known results on cfg we can significantly constrain the form of cfg to be studied here.

Def 1b1 A cfg, $G = (V,T,X,P)$ is said to be totally reduced and we write $G \epsilon \mathfrak{R}$ iff.

a) P contains no rule of the form $Z \rightarrow e$

b) P contains no rule of the form $Z_i \rightarrow Z_j$

c) If $X \overset{*}{\Rightarrow} w$, $w \epsilon V^*$, there is a $y \epsilon T^+$ such that $w \overset{*}{\Rightarrow} y$

d) Each $Z \epsilon V-T$, $a \epsilon T$, and $p \epsilon P$ is used in at least one $d(y,G)$, where y is in $L(G)$.

It is well-known that any e-free language derivable from some cfg can be derived from a cfg in $\mathfrak{R}$. We will restrict ourselves to $G \epsilon \mathfrak{R}$ unless otherwise mentioned.

Lemma 1b2 For any $G \in \mathfrak{R}$ and any $y \in L(G)$ the derivation length

$$\ell_d(y) \leq 2 \cdot \ell(y) .$$

Proof Consider any derivation of y, $d(y,G) = \langle p_1 \ldots p_k \rangle$. Each p_i must either (a) add to the length of the intermediate string or (b) replace a variable by one or more terminal symbols. Since no $p \in P$ can reduce the length of an intermediate string, there are at most $\ell(y)$ instances of (a). In addition, there can be at most $\ell(y)$ variables in an intermediate string and thus $\ell(y)$ instances of (b).

There is an extension of the notion of ordered sequence which will be useful. A sequence $\langle y_1, y_2, \ldots \rangle$ is said to be approximately ordered by a function $f(y)$ iff for each $k \geq 1$ there is an integer $\tau \geq k$ such that $t > \tau$ implies

$$f(y_t) \geq f(y_k) .$$

Lemma 1b3 If $\langle y_i \rangle$ is a sequence which is approximately ordered by f and if $\langle f(y_i) \rangle$ is positive and bounded then there is a C such that

$$\lim_{i \to \infty} f(y_i) = C .$$

Proof We know $\langle f(y_i) \rangle$ has a finite lim sup, call it C. If there is a j such that $f(y_j) = C$, then by approximate ordering there is a τ such that $t > \tau$ implies $f(y_t) = C$ and the lemma is proved.

Suppose the lim sup C is not attained. Let $\epsilon > 0$ be given, then there is a y_k such that $C-\epsilon \geq f(y_k)$ because C is a cluster point. But then there must be a τ_1 such that $t > \tau_1$ implies

$$C-\epsilon \leq f(y_t) .$$

Further, there are at most a finite number of i such that $f(y_i) > C$ because C is the lim sup of a bounded sequence. Let τ_2 be the maximum index of these and let $\tau = \max(\tau_1, \tau_2)$ then for all $t > \tau$ we have

$$C-\epsilon \leq f(y_t) \leq C ,$$

and the lemma is proved. We will be especially interested in cases where $\tau(k)$ is effectively computable.

Finally, we must introduce a number of definitions relating to enumerations of languages. An information sequence of a language L, $I(L)$ is a sequence of symbols from the set

$$\{+y \mid y \in L\} \cup \{-y \mid y \in T^+ - L\} .$$

A positive information sequence $I_+(L)$ is an information sequence of L containing only strings of the form $+y$. Notice that if we bound the number of occurences of any string y in $I(L)$ then $I(L)$ is approximately ordered by $\ell(y)$. The set of all (positive) information sequences for $L \subset T^+$ is denoted $(\mathcal{I}_+)\mathcal{I}$. In Gold [67], $\mathcal{I}_+$ was called the set of text presentations and $\mathcal{I}$ the set of informant presentations. Let $I(L)$ be a (positive) information sequence, we define a (positive)

sample $S_t(I)$ to be the unordered set: $S_t(I) = \{^{+}y_1, \ldots, ^{+}y_t\}$. A bounded sequence is one in which there is a bound on the maximum number of occurences of a string. The set of (positive) bounded information sequences is denoted $(\mathcal{J}_+)\mathcal{J}$. An information sequence is complete if each string in T^+ occurs in the sequence.

A positive information sequence is complete for a language if each sentence of the language occurs in the sequence. Unless explicitly stated, we restrict ourselves to complete sequences. Information sequences and samples will occur in Section 2c and will play a central role in Section 3.

Each positive sample can be associated with a frequency distribution over its elements as follows:

For each $+y_i \epsilon S_t(I)$, $f(I,y_i,0) = 0$

$$f(I,y_i,\tau) = f(I,y_i,\tau-1) + \begin{cases} 0 & \text{if } y_\tau \neq y_i \\ 1 & \text{if } y_\tau = y_i \end{cases} .$$

$f(I,y_i,t)/t$ is the relative frequency of y_i in the first t strings of I. An information sequence I is convergent iff

$$\lim_{t \to \infty} f(I,y_i,t)/t = P_i$$

exists and is non-zero for each $y_i \epsilon I$. The set of positive convergent information sequences is denoted K_+.

Additional Notation

$n(X)$: if X is a finite set of objects (e.g. strings), then $n(X)$ is the number of objects in X ; $n(X)$ is the cardinality measure for finite sets.

r: $r = n(T)$ = the number of terminal symbols in the alphabet T .

$L_k (k = 0,1,2,\ldots)$: $L_k = L \cap T^k$; L_k is the subset of the language L which contains only strings of length k .

$L_k(\alpha)$: $L_k(\alpha) = L_k \cap \alpha T^*$, $L_k(\alpha)$ is that subset of L_k which is prefixed by $\alpha \in T^*$; $L_k(e) = L_k$.

$T^k(\alpha)$: $T^k(\alpha) = \alpha T^* \cap T^k$

2. Grammatical Complexity

2a.o Introductory Measures

There are a number of ways in which one could measure the complexity or information content of an abstract language. One traditional way is to consider the relative sizes of various subsets of the language and develop size measures for languages. Examples of size measures will be considered shortly. Other types of complexity measures can be developed in terms of time and space bounds on the automata associated with a language; studies of this type are currently quite popular (e.g. Hartmanis [68]). Other possible complexity measures could be based on the complexity of algebraic decomposition of the automata associated with a language.

At this point a distinction should be made between complexity measures of a language and complexity measures of a grammar. To be independent of the various grammar(s) for L, a language measure of L should be sensitive only to the content of the subsets of L, not to the structural form of the elements of these subsets. Measures based on the grammars or automata associated with a language often do not characterize the language, since the value of the measure can vary among weakly equivalent grammars (automata) of the language. The class of size measures of languages is one example of language measures which proves useful in studies of complexity. We consider briefly two particular size measures for arbitrary languages $L \subset T^*$.

First-order (density) size measure

Consider the sequence $\langle d^{(k)} \rangle$, where $d^{(k)}$ is the proportion of strings of length k which are in the language L being measured:

$$d^{(k)} = \frac{n(L_k)}{n(T^k)} = \frac{n(L_k)}{r^k} .$$ Suppose the sequence $\langle d^{(k)} \rangle$ converges to a limit d , so that $d = \lim_{k \to \infty} \frac{n(L_k)}{r^k}$; then we would like to define d as the density of the language L , which can assume values in the unit interval $0 \leq d \leq 1$. The density is intuitively the limiting proportion of strings in the language.

There are often, however, languages which seem to contain a well-defined limiting proportion of strings, yet for which the sequence $\langle d^{(k)} \rangle$ does not converge. As a trivial example, consider the language which consists of precisely those strings of even length; in some sense it seems that half of the strings are in the language, but the sequence $\langle d^{(k)} \rangle = \langle \ldots,1,0,1,0,1,\ldots \rangle$ does not converge to any limit, let alone the desired limit of $\frac{1}{2}$. The sequence $\langle s^{(k)} \rangle = \langle \frac{1}{k} \sum_{i=1}^{k} d^{(i)} \rangle$ does, however, converge to the desired limit of $\frac{1}{2}$, since $s^{(k)} = \begin{cases} \frac{1}{2} & , k \text{ even} \\ \frac{1}{2} + \frac{1}{k} & , k \text{ odd} \end{cases}$.

The sequence $\langle d^{(k)} \rangle$ is said to be Cesàro-summable to $\frac{1}{2}$ (see, for example, Kemeny, Snell and Knapp [66]). Since $s^{(k)}$ is the arithmetic mean of the first k proportions, it seems reasonable to interpret the (unique) value to which the sequence $\langle d^{(k)} \rangle$ is summable as the density. This example motivates the following definition of density:

If the sequence $\langle d^{(k)} \rangle$ is Cesàro-summable to d, then d is defined as the first-order (density) size measure of the language. If the sequence is not Cesàro-summable, then the measure is undefined.

Clearly if $\langle d^{(k)} \rangle$ converges to a limit d, then it must also be Cesàro-summable to d. Cesàro-summability is well-known to be equivalent to other types of sequence summability (e.g. Euler-summability) in the sense that, if the sequence sums to a value by one method, then it must also sum to the same value by the other methods. Although occasionally useful, we will not discuss other types of summability.

Suppose that $\langle d^{(k)} \rangle$ is an ultimately periodic sequence with period p, so that $\lim_{k \to \infty} d^{(kp+q)} = d_q$, $q = 0,\ldots,p-1$. Then it can be shown that $\langle d^{(k)} \rangle$ is Cesàro-summable to $d = \frac{1}{p} \sum_{q=0}^{p-1} d_q$, which again illustrates the usefulness of allowing Cesaro-summability as a more general convergence criterion than the commonly used simple "limit" . We shall adopt the notation $b = \operatorname{clim}_{k \to \infty} b^{(k)}$ to indicate that the sequence $\langle b^{(k)} \rangle$ is Cesàro-summable to b.

It is difficult to develop useful existence conditions for the density measure of an arbitrary language $L \subset T^*$, since L clearly can be chosen in such a way that the sequence $\langle d^{(k)} \rangle$ fails to exhibit any stationary behavior. Existence conditions become more tractable when L is assumed to be associated with a certain class of grammars or automata. For example, it is shown in Appendix B that the density measure exists for all finite-state languages (if Cesàro-summability is allowed to be a condition for convergence).

The density measure can be useful as a means of comparing the relative size of languages. But relative size discrimination by means of density breaks down if the languages have either zero or unity density. Most languages we have occasion to investigate have zero density; accordingly, a more sensitive size measure is required for comparison of the relative sizes of zero-density languages (which could be used to compare unity density languages by comparing their zero density complements.)

Second-Order (logarithmic density) size measure

When the densities of two languages are zero, a more sensitive measure is needed to compare their relative sizes. Consider transforming the sequence $\langle d^{(k)} \rangle$ into a logxlog scaled sequence $\langle h^{(k)} \rangle$, where $h^{(k)} = \dfrac{\log n\ (L_k)}{\log n\ (T^k)} = \dfrac{1}{\log r} \cdot \dfrac{\log n\ (L_k)}{k}$

($\log n\ (L_k)$ is taken as zero if $L_k = \emptyset$). We define the second-order (logarithmic density) size measure h of L as

$$h = \operatorname*{clim}_{k \to \infty} h^{(k)}$$

(h is undefined if $\langle h^{(k)} \rangle$ is not Cesàro-summable). The quantity $C = (\log r)\, h$ is the familiar measure termed the channel (coding) capacity of L (we have extended the standard definition of C by permitting Cesàro-summability of the sequence $\left\langle \dfrac{\log n\ (L_k)}{k} \right\rangle = \log r \langle h^{(k)} \rangle$ rather than just strict convergence). When it exists, logarithmic density satisfies $0 \leq h \leq 1$. Furthermore, it can be shown that

(i) $\forall L \subset T^*$, if d exists and $d > 0$, then h exists and $h = 1$

(ii) if both d and h exist and $h = 1$, then $d > 0$

(iii) if both d and h exist and $d = 0$, then $h < 1$

We thus see that logarithmic density is a useful size measure among minimal (zero) density languages, while density is a useful size measure among maximal (unity) logarithmic density languages.

The logarithmic density (and thus the channel capacity) of a language is strictly a size measure, and is not essentially an information-theoretic language measure as the name channel capacity seems to suggest. The channel capacity is the maximum possible (limiting) mean rate of information transmitted/symbol across a discrete noiseless channel. Several authors have termed the quantity C (or h) the entropy (or relative entropy) of the language, a somewhat misleading terminology; in terms of classical information theory, C is the maximum rate (per symbol) of entropy for possible "stochastic grammars" of the language. There are, at least for some classes of languages, stochastic representations of grammars for the language which achieve this maximum entropy rate (channel capacity). In terms of "selective information theory" (Luce 60, Chomsky and Miller 63b), C is indeed the entropy rate of the language. We emphasize that several stochastic grammars (automata) for a given language may have different entropy rates, but C is an upper bound for them.

Other size measures

The first and second-order size measures of a language L can be generalized as functions of a given string $\alpha \in T^*$:

$$d(\alpha) = \lim_{k \to \infty} \frac{n(L_k(\alpha))}{n(T^k(\alpha))} = \lim_{k \to \infty} \frac{n(L_k(\alpha))}{r^{k-\ell(\alpha)}}$$

$$h(\alpha) = \lim_{k \to \infty} \frac{\log n(L_k(\alpha))}{\log n(T^k(\alpha))} = \frac{1}{\log r} \lim_{k \to \infty} \frac{\log n\ (L_k(\alpha)}{k-\ell(\alpha)}$$

(Note: where Cesàro-summability is used, it is understood that summation begins with $k = \ell(\alpha)+1$ rather than with $k=1$).

Note that substituting $\alpha=e$ into $d(\cdot)$ and $h(\cdot)$ yields the size measures d and h, respectively. Discussion of $d(\alpha)$ and $h(\alpha)$ with respect to stochastic grammars and selective information theory is an interesting topic, but unfortunately exceeds the scope of this presentation.

Remarks:

Chomsky and Miller [58] claimed that the probability of a randomly chosen string of length k being in any given regular language converges to either zero or one as k increases without bound. This claim is equivalent to stating that the density of any regular language is either zero or unity. To our surprise we have encountered restatement of this claim by later authors (e.g. Kuich and Walk 65). The claim is false, as is shown in Appendix B. There appears to be two sources of error in Chomsky and Miller's development. First, there seems to be some confusion between first and second order size measure with respect to probability; Chomsky and Miller's argument was based on channel capacity (second-order measure)

rather than on first-order density; density is equivalent to the limiting proportion of strings in the language. Second, a matrix or "equational" representation of finite-state grammars was used by Chomsky and Miller – indeed, has been used extensively in the literature – which is inadequate for the class of all finite-state grammars; there are regular languages which cannot be generated by any grammar associated with the matrix representation. The interested reader is referred to Appendix A for examples of regular languages for which the representation is not adequate, and for a suggested matrix representation which is adequate for all finite-state languages.

2a. Introductory Definitions and Examples

The concern here is with a representational measure of complexity. We will be interested in the following questions. How well does a given grammar fit a sample? How complicated is a grammar? What is the most satisfactory grammar from a given class for some sample set of strings? The results of this section are of some intrinsic interest and will be very valuable in the grammatical inference problem considered in Section 3. The techniques described here, although discussed in terms of grammars, seem applicable to a broad class of problems involving the fitting of a model to data, [cf. Feldman 67]. The particular measures studied here are related to Bayes Theorem and to the measures of Solomonoff [64].

Def 2a1 Let $G = (V,T,X,P)$ a cfg; the alternative set $A(p)$ of a production $p \epsilon P$ of the form $Z \rightarrow w$ is the set of productions in P with the same left side, Z, i.e., $A(Z \rightarrow w) = \{(Z \rightarrow x) \epsilon P\}$.

We will be interested in measures which depend on the alternative set, and for most of the discussion will be concerned with a very restricted class of such functions.

Def 2a2 A function $\rho(p)$ is a density iff

1) ρ is defined for all $p \epsilon P$ for any $G \epsilon \mathfrak{R}$
2) $0 \leq \rho < \infty$
3) For each $p \epsilon P$, $\sum_{p' \epsilon A(p)} 2^{-\rho(p')} = 1$.

A density is intended to describe how precisely a grammar "fits" a set of strings. The description of a set of strings in terms of a grammar will be more complex if the grammar generates many strings besides those in the set. Each step in a derivation will be considered more complex in a grammar which allows many derivations from that non-terminal (has a large alternative set). It is also possible to consider ρ from an information-theoretic point of view; $\rho(p)$ is a measure of the information required to select p from the set of productions with the same left side, i.e., $2^{-\rho(p)}$ is the probability of a particular alternative.

It is this information theoretic approach which gives rise to the specific density used here. If we assume that all productions with the same left part are equally likely, we get a local measure

$$\sigma(p) = \log_2(b(p))$$

where $b(p)$ is the cardinality of $A(p)$.

Another possibility is to assign some a priori likelihoods to each production p. This could be based on some complexity measure on p itself (such as its length). We will concentrate on proving properties of the general density ρ, but will use σ in the samples. Before presenting examples, we must extend the notion of density to a complexity measure for derivations.

Let $d(y,G) = \langle p_1,\ldots,p_h\rangle$ be a derivation of y and let $\rho(p)$ be a density, we define

$$m(d,y,G) = \sum_{j=1}^{h} \rho(p_j) \quad .$$

We can now define the complexity of a string relative to a grammar.

Def 2a3 Let $y\in T^+$. If $y\notin L(G)$ we define the complexity $\mu(y,g)$ to be ∞. If $y\in L(G)$ and the derivations of y are $d_1(y,G),\ldots,d_k(y,G)$ we define

$$\mu(y,G) = \frac{1}{k}\sum_{i=1}^{k} m(d_i,y,G) \quad .$$

Def 2a4 Let $S = \{y_1,\ldots,y_n\} \subset T^+$ the complexity of the set S relative to G, $\mu(S,G)$ is defined by

$$\mu(S,G) = \sum_{i=1}^{n} \mu(y_i,G) \quad .$$

Thus the complexity of a string is the average of the complexity of its derivations; the complexity of a set is the sum of the complexities of its members.

If S is a finite subset of T^+ $\mu(S,G) = \infty$ iff $S - L(G) \neq \emptyset$.

The value of $\mu(y,G)$ is a measure of the complexity of a derivation of y from G and might be usable as a measure of grammatical complexity. We defer the discussion of the relative merits of various complexity measures until Section 3a.

Example 2a5 Let $G = (\{X\},\{a,b\},X,\{X \to a \mid b \mid aX \mid bX\})$.

This is the universal grammar over $\{a,b\}$. For this grammar, any string of length n requires a sequence of n productions in its unique derivation. If we use the density σ as ρ, each production p has $\rho(p) = \log_2(4) = 2$. Thus each $y \in \{a,b\}^*$ has

$$\mu(y,G) = 2 \cdot \ell(y) \quad .$$

Let $H = (\{X,Z_1\},\{a,b\},X,\{X \to b \mid aZ_1 \mid bX,\ Z_1 \to a \mid aX \mid bZ_1\})$. This is the "even number of a's" grammar. Similar reasoning to the above will show that for any string y with an even number of a's:

$$\mu(y,H) = \log_2(3) \cdot \ell(y) \quad .$$

The example indicates that μ corresponds to our intuition in declaring the universal grammar to have more complex derivations of strings having only an even number of a's. There is, however, a potential problem in the fact that H itself seems more complex than G. We have, so far, considered only the complexity of derivations. If, as in the grammatical inference problem, only a finite set of strings is available for testing, a very complex grammar may yield the lowest value of μ. For example, the grammar which simply lists the sample set (ad hoc grammar) will have

a very low measure. In the next section we will expand the notion of grammatical complexity to include a measure of the complexity of the grammar itself.

2b. Grammar-grammar, Complexity of Grammars

We will define the complexity of a grammar as the complexity of its derivation in some grammar-grammar, $\bar{G}$. The choice of $\bar{G}$ will determine which subclass of the context-free grammars is under consideration. Typical subclasses include the linear grammars, grammars in some standard form, and grammars restricted to a fixed number of variables.

Def 2b1 A grammar-grammar $\bar{G} = (\bar{V},\bar{T},\bar{X},\bar{P})$ on the terminal alphabet T is defined to be a cfg such that

1) $(\bar{V}-\bar{T}) \cap \mathbb{U} = \emptyset$

2) $\bar{T} \subset \mathbb{U} \cup T \cup \{\rightarrow\} \cup \{,\}$

where $\mathbb{U}$ is the universe of variable symbols and "," is used to separate the rules of P.

It would be possible to sharpen this definition, e.g. to allow only $Z \in \mathbb{U}$ to appear to the left of "→" in a string. It is not possible, however, to force $\bar{G}$ to produce only $G \in \mathcal{R}$, with a context-free $\bar{G}$. There is the additional problem that $\bar{V}$ must be finite so a given $\bar{G}$ will only generate a class of languages with a fixed number of variables. The following definitions modify the grammar-grammar concept and make it more suitable for our purposes. It is also convenient to have the production arrow for grammar-grammars be "::=".

Def 2b2 A sequence of grammar-grammars $\bar{C} = \{\bar{G}_1, \bar{G}_2, \ldots\}$ is a collection iff. There is a $\bar{S}$ such that for each $\bar{G}_i$

1) $\bar{Z} ::= Z_0 \mid Z_1 \mid \ldots Z_{i-1}$ in $\bar{G}_i$.

2) $\bar{Z}$ appears in no other left sides.

3) No $Z \epsilon \mathbb{W}$ appears in any other rule.

4) The $\bar{G}_i$ are identical except for the rule described in 1).

The intent here is that $\bar{Z}$ is the variable in all $\bar{G}_i$ which produces the i variables of the G_i .

Def 2b3 A representation class C is defined as

$$C = \left(\bigcup_{\bar{G} \epsilon \bar{C}} L(\bar{G}) \right) \cap R$$

where $\bar{C}$ is a collection. Thus, C is a set of grammars defined by a collection $\bar{C}$ such that for any $G \epsilon C$, there is a $\bar{G} \epsilon \bar{C}$ such that $G \epsilon L(\bar{G}) \cap R$.

This definition allows subfamilies of cfg with an unbounded number of variables to constitute a representation class. For any $G \epsilon R$ and any class C it is decidable whether $G \epsilon C$. More frequently we will be interested in studying all the grammars in some class C . We will sometimes write $G(k)$ for $G \epsilon C$ such that $G \epsilon L(\bar{G}_k) \cap R$.

The intrinsic complexity of a grammar G can now be defined as the complexity of its derivation from an appropriate grammar-grammar, $\mu(G, \bar{G})$ using $\rho = \sigma$ as density. The choice of the grammar-grammar $\bar{G}$ will depend on the set of grammars being compared. We now derive expressions for $\mu(G, \bar{G})$ for a number of interesting subclasses C of R on a fixed terminal alphabet $T = \{a_0, \ldots, a_{m-1}\}$.

For all the examples we will have $\bar{G}_n = (\bar{V},\bar{T},\bar{X},\bar{P})$ with

$$\bar{V} = \{\bar{X},Q,R,N,T\} \cup \bar{T}$$

$$\bar{T} = \{Z_0,\ldots,Z_{n-1},a_0,\ldots,a_{m-1},\rightarrow\} \cup \{,\}$$

The general cfg with n variables can be derived from the collection $\bar{C} = \{CF_n\}$. The productions $\bar{P}$ of CF_n are

$$\bar{X} ::= Q \mid \bar{X},Q$$

$$Q ::= N \rightarrow R$$

$$N ::= Z_0 \mid \ldots Z_{n-1}$$

$$R ::= T \mid N \mid TR \mid NR$$

$$T ::= a_0 \mid \ldots a_{m-1}$$

For a grammar G in $L(CF_n)$ which has k productions, whose right sides have a total of k_1 variables and k_2 terminals we have

$$\mu(G,CF_n) = k \cdot (\log_2(n) + \log_2(2)) + k_1 \cdot (\log_2(4) + \log_2(n)) + k_2 \cdot (\log_2(4) + \log_2(m)) .$$

For cfg in Greibach Standard 2-form (S2) and in modified Operator 2-form (O2) the measures have very similar expressions. The productions are:

S2	O2
$\bar{X} ::= Q \mid \bar{X},Q$	$X ::= Q \mid \bar{X},Q$
$Q ::= N \rightarrow R$	$Q ::= N \rightarrow R$
$N ::= Z_0 \mid Z_1 \mid \ldots Z_{n-1}$	$N ::= Z_0 \mid \ldots Z_{n-1}$
$R ::= T \mid TN \mid TNN$	$R ::= T \mid TN \mid NTN \mid NT$
$T ::= a_0 \mid \ldots a_{m-1}$	$T ::= a_0 \mid \ldots a_{m-1}$

and if a grammar G has k productions and k_1, k_2, k_3 rules whose right sides are of length 1, 2, 3 respectively, then

$$\mu(G,S2_n) = k(\log_2(n)+\log_2(3)+\log_2(m)+\log_2(2)) + (k_2 + 2k_3)\log_2(n)$$

$$\mu(G,O2_n) = k(\log_2(n)+\log_2(4)+\log_2(m)+\log_2(2)) + (k_2 + 2k_3)\log_2(n) .$$

Similarly, the linear grammars (LN) and finite state grammars (FS) have nearly identical $\bar{G}$. The productions are:

LN	FS
$\bar{X} ::= Q \mid \bar{X},Q$	$\bar{X} ::= Q \mid \bar{X},Q$
$Q ::= N \rightarrow R$	$Q ::= N \rightarrow R$
$N ::= Z_o \mid \ldots Z_{n-1}$	$N ::= Z_o \mid \ldots Z_{n-1}$
$T ::= a_o \mid \ldots a_{m-1}$	$T ::= a_o \mid \ldots a_{m-1}$
$R ::= T \mid TN \mid NT$	$R ::= T \mid TN$

and if a grammar G has k productions and k_1, k_2 rules whose right sides are of length 1, 2 respectively, then

$$\mu(G,LN_n) = k(\log_2(n)+\log_2(3)+\log_2(m)+\log_2(2)) + k_2\log_2(n)$$

$$\mu(G,FS_n) = k(\log_2(n)+\log_2(2)+\log_2(m)+\log_2(2)) + k_2\log_2(n) .$$

Finally, the productions and measures for Chomsky normal form (C2) are:

C2

$\bar{X} ::= Q \mid \bar{X},Q$

$Q ::= N \rightarrow R$

$N ::= Z_o \mid \ldots Z_{n-1}$

$R ::= T \mid NN$

$T ::= a_o \mid \ldots a_{m-1}$

$$\mu(G,C2_n) = k(\log_2(n))+\log_2(2)+2k_2(\log_2(n)+\log_2(2))+k_1(\log_2(m)) .$$

Example 2b4 Returning to our example of the universal grammar on strings (Example 2a5) with an even number of a's, we can now measure the complexity of the grammars G, H. We must first determine the appropriate class of grammars and parameters (n,m) to use in the comparison. We have assumed that the terminal alphabet (and thus m) is known. Since both grammars are finite-state, the $\mathbf{C}$ called FS above is most appropriate. Now H (the "even a's" grammar) has two non-terminals. We use $n = 2$ for it and get the result:

$$m = 2\ ,\quad n = 2\ ,\quad k = 6\ ,\quad k_2 = 4$$

$$\mu(H,FS_2) = 6(\log_2(2)+\log_2(2)+\log_2(2)+\log_2(2)) + 4 \cdot \log_2(2) = 28\ .$$

For the universal grammar G which requires only one non-terminal we could use $n = 1$ or $n = 2$. The results are:

$$\mu(G,FS_1) = 12$$

$$\mu(G,FS_2) = 18\ .$$

Although G is simpler than H by either measure, there is a question of which measure to choose. We can see from the formulas derived above for $\mu(G,\bar{G})$ that choosing the smallest possible n produces a bias in favor of grammars with few non-terminals. This seems desirable and has been adopted for use in this paper.

We will need the following lemma in Section 3 which deals with grammatical inference.

Lemma 2b5 Let $\mathbf{C} \subset \mathbf{R}$ be defined by a grammar-grammar $\bar{G}$ in Standard 2-form (S2), then there is an enumeration $\mathcal{E}$ of $\mathbf{C}$ which is approximately ordered by $\mu(G,\bar{G})$ in an effective manner

Proof If C is finite the problem is trivial. If C is infinite $\mu(G,\bar{G})$ is unbounded on C. Given the grammar-grammar $\bar{G}$, one can define a generating algorithm which will approximately order $L(\bar{G})$ by the length of its strings (grammars). Let $\mathcal{L}$ be the restriction of this approximate order to $G \in \mathcal{R}$, $\mathcal{L}$ is an enumeration of C. Now if G_i in $\mathcal{L}$ is given we must show there is an effective way to find k such that $j > k$ implies

$$\mu(G_j,\bar{G}) \geq \mu(G_i,\bar{G}) \quad .$$

Let r= the minimum density of $p \in \bar{P}$ and let h be such that

$$h \cdot r \geq \mu(G_i,\bar{G}) \quad .$$

We can effectively find k such that $j > k$ implies $\ell(G_j) \geq h$, because $\mathcal{L}$ is approximately ordered by $\ell(G)$. Also for S2 we have $\ell_d(G_j) = \ell(G_j)$ and thus

$$\mu(G_j,\bar{F}) \geq h \cdot r \geq \mu(G_i,\bar{G}) \quad .$$

The two complexity measures developed here (the intrinsic complexity of a grammar and the complexity of a set of strings relative to a grammar) can be combined to form an overall measure of how well some grammar fits a set of strings. The problem of what combination of $\mu(G,\bar{G})$ and $\mu(S,G)$ to use in an overall measure will be discussed in Section 3c. For the present we will be content with an example.

Def 2b6 Let G be a grammar in a class C defined by $\bar{C}$. Let S be a subset of T^+, then we define the measure $m_C(S,G)$ by

$$m_C(S,G) = \mu(S,G) + \mu(G,\bar{G}) \quad .$$

We can now reconsider Example 2b4 using m_C . The universal grammar G is simpler than H , but leads to more complex derivations. We can then investigate which sets S will cause one to prefer H to G as a grammar for S , i.e., make

$$m_{FS}(S,H) < m_{FS}(S,G) \quad .$$

Using Def. 2b6 and the intrinsic complexities computed for H,G this is equivalent to finding S such that

$$\mu(S,H) + 28 < \mu(S,G) + 12$$

or

$$\mu(S,G) - \mu(S,H) > 16 \ .$$

Now from the results of 2a5 this is satisfied by any set of strings S satisfying

$$\sum_{y \in S} \ell(y) > 39 \ .$$

Although it involves getting ahead of ourselves somewhat, we should consider this example more closely. In general, $m_C(S,G)$ will depend on the nature of S rather than some simple property as in this case. Here we have shown that any sample including 39 or more symbols and having only strings with an even number of a's makes H preferable to G . Notice that a single string with an odd number of a's will make $\mu(S,H) = \infty$. The result above says nothing about other grammars which might be better than both G and H on some set S ; this is the grammatical inference problem and is the subject of Section 3. We first introduce a variation on complexity measures which plays a major role in the discussion of grammatical inference.

2c. Normalized Complexity Measures

The complexity measures introduced in the last section increase without bound with the length of strings. To overcome this difficulty we introduce a normalized complexity measure; this measure is bounded so we may also study its limiting behavior as the sample set of strings approaches the language.

Def 2c1 The normalized complexity $\eta(y,G)$ of a string $y \in T^+$ relative to a grammar G is defined by

$$\eta(y,G) = \mu(y,G)/\ell(y)$$

where $\mu(y,G)$ is defined in 2a3 and $\ell(y)$ is the length of y.

The definition of η is extended to sets, S, of strings by

$$\eta(S,G) = \mu(S,G)/\sum_{y \in S} \ell(y) \quad .$$

Lemma 2c2 For any $G \in \mathfrak{R}$, $y \in S \subset L(G)$ there are constants $r,q > 0$ such that

(a) $r \leq \eta(y,G) \leq q$

(b) $r \leq \eta(S,G) \leq q$.

Proof (a) By 1b2 the derivation length $\ell_d(y)$ is not greater than $2 \cdot \ell(y)$. If B is the maximum $\rho(p)$ in G then

$$q = 2 \cdot B$$

satisfies the right half of (a) because if there are k derivations of a string y, we have:

$$\eta(y,G) = \frac{1}{k \cdot \ell(y)} \sum_{i=1}^{k} \sum_{j=1}^{\ell_d(y_i)} \rho(p_{ij})$$

$$\leq \frac{1}{k \cdot \ell(y)} \sum_{i=1}^{k} \sum_{j=1}^{2 \cdot \ell(y)} B$$

$$= 2 \cdot B \quad .$$

Let $k(p)$ be the number of terminal symbols appearing in production p. Let r be the minimum over G of $\rho(p)/k(p)$, then r satisfies the left side of (a). The proof of (b) follows by straightforward analysis from (a) and the definitions.

The introduction of the normalized complexity measure $\eta(S_t,G)$ enables us to study the behavior of η as the sets S_t approach $L(G)$. When the limit exists we will write

$$\eta(L,G) = \lim_{t \to \infty} \eta(S_t,G) \quad .$$

The following example will show that the limit may not exist.

Example 2c3 Let $G = (\{a,c,X,Z_1\},\{X,Z_1\},X,P)$ where P contains

$$X \to a \mid aX \mid cZ_1 \mid c$$

$$Z_1 \to cZ_1 \mid c$$

and let the density $\alpha = \sigma$. The language $L(G)$ is the set of all strings containing a finite number of a's followed by a finite number of c's. We will show that there are information sequences for which $\eta(S_t,G)$ does not converge.

Let a^n be a string of a's of length n and c^m be a string of c's of length m. Then

$$\mu(a^n,G) = n \cdot \log_2(4)$$

$$\mu(c^m,G) = \log_2(4) + (m-1)\log_2(2) \quad .$$

On a sequence of strings of the form a^i, we have $\eta(S,G)$ converging to 2 and on a sequence of c^i, $\eta(S,G)$ converges to 1. We will now show how to choose an information sequence which includes every string in $L(G)$ exactly once and for which $\eta(S_t,G)$ fails to converge. The first string is "a" and the subsequent strings are chosen as follows.

After choosing a string a^i we choose all strings of $L(G)$ of length up to i and compute $\eta(S_t,G)$ on this set S_t of strings. There is a string c^j which, if chosen as the $(t+1)$st element of I, will cause $\eta(S_{t+1},G)$ to be less than 1.4. For example, if $S_1 = \{a\}$, then $S_2 = \{a,c\}$ and j must be such that

$$\frac{\log_2 4 + \log_2 4 + \log_2 4 + (j-1)}{2+j} < 1.4$$

which is satisfied by $j = 7$ and $S_3 = \{a,c,ccccccc\}$. We then select all new strings of length up to j and compute $\eta(S_{t_2},G)$. There is an integer j_2 such that

$$\eta(S_{t_2} + \{a^{j_2}\},G) > 1.6 \quad .$$

By continuing this process one can produce an information sequence on which $\eta(S,G)$ fails to converge.

In the example above, the failure of $\eta(S,G)$ to converge depended on three factors: the density ρ, the derivation length ℓ_d and the

information sequence $I(L)$. By restricting these factors in various ways, one can show that there are cases where $\eta(S,G)$ is known to converge. We first examine the case where $\rho(p)$ is constant; this amounts to using the length of a derivation as a complexity measure. We will use the notation $\hat{\ell}_d(y)$ to denote the average derivation length of a string y .

Theorem 2c4 Let $G \in \mathfrak{G}$ be such that $\rho(p) = r$, a constant for all $p \in P$, then for any $I(G)$ for which

$$\lim_{t\to\infty} \frac{\sum \hat{\ell}_d(y_i)}{\sum \ell(y_i)} = C$$

the limit of $\eta(S_t,G)$ exists, and

$$\lim_{t\to\infty} \eta(S_t,G) = rC_1 \quad .$$

Proof By definition

$$\eta(S_t,G) = \frac{\sum_{i=1}^{t} \frac{1}{k_i} \sum_{h=1}^{k_i} \sum_{j=1}^{\ell_d(y_{ih})} \rho(p_{ihj})}{\sum_{i=1}^{t} \ell(y_i)}$$

but with $\rho(p_{ihj}) = r$ this collapses to

$$\eta(S_t,G) = r \frac{\sum_{i=1}^{t} \frac{1}{k_i} \sum_{h=1}^{k_i} \ell_d(y_i h)}{\sum_{i=1}^{t} \ell(y_i)}$$

$$= \frac{r \sum_{i=1}^{t} \hat{\ell}_d(y_i)}{\sum_{i=1}^{t} \ell(y_i)}$$

which proves the theorem.

Corollary 2c5. Let $G \in \mathfrak{R}$ be such that

1) $\rho(p) = r$ a constant for all $p \in P$

2) $\ell_d(y) = a \cdot \ell(y) + b$; a, b positive constants.

Then for any $I(G)$ we have

$$\lim_{t \to \infty} \eta(S_t, G) = ra \quad .$$

This shows that for a constant density ρ and grammars whose ℓ_d is simple, the normalized complexity measure always converges. This is interesting because many classes of grammars satisfy Condition 2 of Corollary 2c5.

For the Chomsky standard form C2 , we have $\ell_d(y) = 2\ell(y) - 1$. For each of the representation classes FS, LN, G2, S2 we have $\ell_d(y) = \ell(y)$. These relations are immediate consequences of the form of productions for each class. We now consider the results of allowing ρ to be non-constant.

We present two versions of the conditions for the convergence of $\eta(S,G)$ with non-constant ρ . The first, Theorem 2c7, is simple to prove and illustrates the nature of the problem. The second, Theorem 2c8, is more useful when it applies.

Def 2c6 Let $u_{ih}(p_j)$ be the number of uses of production j in derivation h of the string y_i . Also let $\hat{u}_i(p_j)$ be the average of $u_{ih}(p_j)$ over the derivations of y_i .

Theorem 2c7 Let $G \in \mathbf{R}$ be such that $P = \{p_1,\ldots,p_s\}$, i.e., there are s productions in the grammar. A sufficient condition for the limit as $t \to \infty$ of $\eta(S_t,G)$ to exist is that for $j = 1,2,\ldots,s$ the following limit exists

$$(1) \qquad \lim_{t \to \infty} \frac{\sum_{i=1}^{t} \hat{u}_i(p_j)}{\sum_{i=1}^{t} \ell(y_i)}$$

Proof One can rewrite the definition of $\eta(S_t,G)$ as:

$$\eta(S_t,G) = \frac{\sum_{i=1}^{t} \frac{1}{k_i} \sum_{h=1}^{k_i} \sum_{j=1}^{s} u_{ih}(p_j) \cdot \rho(p_j)}{\sum_{i=1}^{t} \ell(y_i)} .$$

Reversing the sums over h,j and using the definition of $\hat{u}_i(p_j)$ gives

$$\eta(S_t,G) = \frac{\sum_{i=1}^{t} \sum_{j=1}^{s} \hat{u}_i(p_j) \cdot \rho(p_j)}{\sum_{i=1}^{t} \ell(y_i)} .$$

Now reversing the order to summation again and separating out the contributions of each production p_j as $\eta_j(S_t,G)$ we have

$$\eta_j(S_t,G) = \frac{\rho(p_j) \sum_{i=1}^{t} \hat{u}_i(p_j)}{\sum_{i=1}^{t} \ell(y_i)}$$

from which the theorem is apparent. The condition of Theorem 2c7 is that some average number of uses of a production in deriving a set

of strings should converge. The difficulty is that $\hat{u}_i(p_j)$ it is hard to establish for a given grammar and information sequence.

A more reasonable condition to establish is the ratio of the uses of p_j to the total number of steps in deriving the set S_t. That is

$$f_j(S_t) = \frac{\sum_{i=1}^{t} \sum_{h=1}^{k_i} u_{ih}(p_j)}{\sum_{i=1}^{t} \sum_{h=1}^{k_i} \ell_d(y_{ih})} \quad .$$

Thus the frequency of a production p_j in deriving the set of string S_t is the total number of uses of p_j divided by the number of production steps used for the set S_t. We will use this definition to establish a condition under which $\eta(S_t,G)$ converges and then discuss $f_j(S_t)$ further.

Theorem 2c8 Let $G\epsilon\mathcal{R}$ be unambiguous and be such that $P = \{p_1,\ldots,p_s\}$ and $\ell_d(y) = a \cdot \ell(y)+b$ for all $y\epsilon L(G)$. Further, let $I(G)$ be a bounded information sequence such that

$$\lim_{t\to\infty} f_j(S_t) = C_j \quad \text{for each production } p_j\epsilon P \ ,$$

then

$$\lim_{t\to\infty} \eta(S_t,G) = C \quad .$$

Proof Since G is unambiguous, all $k_i = 1$ and

$$f_j(S_t) = \frac{\sum_{i=1}^{t} u_i(p_j)}{\sum_{i=1}^{t} \ell_d(y_i)} \quad .$$

Separating the contributions of each p_j as in the proof of 2c4 we have:

$$\eta_j(S_t,G) = \frac{\rho(p_j)\sum_{i=1}^{t} u_i(p_j)}{\sum_{i=1}^{t} \ell(y_i)}$$

$$\eta_j(S_t,G) = \frac{\rho(p_j)\sum_{i=1}^{t} u_i(p_j)}{\sum_{i=1}^{t} \ell(y_i)} \quad .$$

Also:

$$f_j(S_t) = \frac{\sum_{i=1}^{t} u_i(p_j)}{\sum_{i=1}^{t} (a \cdot \ell(y_i)+b)}$$

$$= \frac{\sum_{i=1}^{t} u_i(p_j)}{a\sum_{i=1}^{t} \ell(y_i) + b \cdot t} \quad .$$

The advantage of Theorem 2c8 is that the convergence of $f_j(S_t)$ may be provable under fairly general conditions. We are now attempting to use stochastic matrix results to establish such conditions. Theorem 2c8 does not hold for ambiguous languages; this situation is symptomatic of a number of problems arising from ambiguity and will be discussed in some detail.

Even very simple grammars may have ambiguity (k_i) which grows exponentially with the length of y_i. An example is

$$H = (\{Z,a\},\{a\},Z,\{Z \rightarrow a|aZ|Za\}) \quad .$$

Since we defined $\eta_j(S_t,G)$ in terms of the <u>average</u> number of uses of p_j, the value of k_i has essentially no effect on η. For $f_j(S_t)$, however, the <u>total</u> number of uses of a production is used. Consider the grammar of Example 2c3 with one additional production rule:

$$X \rightarrow Xa$$

In this grammar, each string k "a"'s has 2^k derivations. By methods like those of 2c3 it is easy to show there is an information sequence for which $f_j(S_t)$ converges and $\eta_j(S_t,G)$ does not, which fact refutes Theorem 2c8 for ambiguous grammars.

The choice of $\eta(S_t,G)$ as a function of the average complexity of the derivations of a string is open to question. Other possible choices would be the sum, maximum, minimum and a weighted sum. The choice of definition of η has important implications for the entire grammatical complexity problem. This issue is touched on in Section 3d and will be further discussed in Horning's dissertation.

3. Grammatical Inference

3a. Introduction, Basic Model and Terminology

The problem of inferring a grammar for a set of strings is just beginning to receive serious attention. Our purpose here is to establish a number of decidability results as a foundation for the heuristic methods of grammatical inference now being programmed. These results are extensions of the work of [Gold 67] who describes his study as follows:

> Many definitions of learnability are possible, but only the following is considered here: Time is quantized and has a finite starting time. At each time the learner receives a unit of information and is to make a guess as to the identity of the unknown language on the basis of the information received so far. This process continues forever. The class of languages will be considered learnable with respect to the specified method of information presentation if there is an algorithm that the learner can use to make his guesses, the algorithm having the following property: Given any language of the class, there is some finite time after which the guesses will all be the same and they will be correct.

Gold's definition of learnability derives from his earlier work on limiting recursion [Gold 65]. We will present some new results using this definition and show that by relaxing some of its conditions, one can greatly enlarge the class of solvable cases of the grammatical inference problem.

In addition to the concepts previously defined, we will need a number of new ones. We assume time is quantized and is expressed by

$$t = 1, 2, 3 \ldots$$

A grammatical inference device D is a function from samples S_t into the set of grammars $\{G\}$ in some class C. The grammatical inference

problem is modelled as follows: An information sequence is presented to the device D at the rate of one element per time step. At each time, t, we compute

$$A_t = D(S_t(I), C) .$$

We say that a class of languages, L(C), is identifiable in the limit, if there is a function D such that for any $G \in C$ and any information sequence $I(L(G)) \in \mathcal{I}$ there exists a τ such that $t > \tau$ implies both

a) $A_t = A_\tau$

b) $L(A_\tau) = L(G)$.

This differs from the function D being recursive in the following way. A recursive function D would, at some τ, be able to ignore all further information, i.e., would be able to stop and demonstrate the right answer. Since we have allowed an information sequence to contain repetitions of a string, not even the class of finite languages is recursively identifiable.

Before considering the properties of inference devices, let us look at the notion of information sequence. Gold [Gold 67] has shown that there is no effect in the limit on learnability caused by the difference between an ordered (e.g. by length) I and a random one for $I \in \mathcal{I}$. He also shows that in this case allowing the device D to select the next string y to appear as $\stackrel{+}{-}$ y in I does not change things. While these different methods of informing (teaching) the device do not affect the learnability of languages in the limit, they do have powerful effects on the heuristics of efficient learning. Solomonoff [64] considers the grammatical inference problem a special case of sequence extrapolation and his methods rely heavily on the order of presentation of examples. Another crucial consideration is

whether the information sequence contains complete information. The effects of complete samples is the subject of the next section.

3b. New Results on Grammatical Inference

The main results of [Gold 67] deal with the great difference in learnability effected by allowing information sequences with negative instances, $I \in \mathcal{I}$, (informant presentation) rather than just positive instances, $I \in \mathcal{I}^+$, (text presentation). We will informally outline certain key proofs and then extend them in various ways.

All of the methods are based on the denumerability of various classes of grammars; the primitive recursive, context-sensitive, context-free, and any other class we might be concerned with here can be enumerated. Let $\mathcal{G} = \{G_1, \ldots\}$ be an enumeration of such a class. Also let $\mathcal{I} = \{I\}$ be the set of all complete information sequences over some alphabet T (each $y \in T^+$ occurs as $\pm y$ in every I). A class C of grammars is admissible iff C is denumerable and for all $G \in C$, $y \in T^+$ the relation $y \in L(G)$ is effectively computable. A grammar G is compatible with a set of strings $S = S_+ \cup S_-$ iff $S_+ \subset L(G)$ and $S_- \subset T^+ - L(G)$.

Theorem 3b1 (Gold) For any admissible C there is a device $D(S,C)$ such that for any $G \in C$ and any $I(L(G)) \in \mathcal{I}$, $L(G)$ is identifiable in the limit through I.

Proof The device D simply sequences through the enumeration $\mathcal{G}$ of C. At each time, T, there is a first $G \in \mathcal{G}$ which is compatible with $S_t(I)$, it is the guess A_t of D at time t. At some time τ,

A_τ will be such that $L(A_\tau) = L(G)$. Then A_τ will be compatible with the remainder of the information and will be the constant result of D .

Thus with informant presentation, a very wide class of grammars can be learned in the limit. By restricting the information to only $I \in \mathcal{I}_+$ we give up learnability in the limit almost entirely. Let everything be as before except that the set of information sequences $\mathcal{I}_+ = \{I\}$ contains only sequences of the form $<+y_1, +y_2, \ldots>$.

Theorem 3b2 (Gold) Under these conditions any class C generating all finite languages and any one infinite language L_∞ is not learnable in the limit.

Proof We show that for any D , there is a sequence I , which will make D change its value A_t an infinite number of times for L_∞ . Since D must infer all finite languages there is a sample which causes it to yield some $G(L_1)$ such that $L_1 \subset L_\infty$. Now consider an information sequence which then presents some string $x \in L_\infty - L_1$, repeatedly. At some time t , $D(S_t, C)$ must yield a grammar of $L_1 \cup (x) = L_2$ because all finite languages are inferred. This construction can be repeated indefinitely, yielding an information sequence I which will change the value of D an infinite number of times.

This unlearnability result is so strong that we were led to try to consider it further. The remainder of this section is devoted to the study of conditions under which learnability from positive sequences only is

attainable. Let us first consider the repeated occurrence of a string y, in an information sequence I. The proof above is based on the possibility of having some string occur indefinitely often; it does not seem unreasonable to bound the number of occurrences of any string in an information sequence and thus restrict our attention to $\mathcal{I}_+$.

By restricting consideration to bounded information sequences, we have made the problem of identifying finite languages trivial. The classes of grammars which are now identifiable in the limit can be characterized by the following two lemmas.

Lemma 3b3 Any class of cfg $C \subset \mathcal{R}$ which contains only a finite number of grammars which generate infinite languages is identifiable in the limit from any $I(L(G)) \in \mathcal{I}_+$.

Proof The device $D(S_t, C)$ which will identify C in the limit will be defined. Let $\mathcal{A}$ be an enumeration of the grammars of C which generate infinite languages. At each time t, the device D will form a guess A_t as follows. A_t is the first grammar in $\mathcal{A}$ which is compatible with S_t and which generates the minimum number of strings of length less than or equal to k, where k is the length of the longest string S_t. If the language $L(G)$ is finite then $I(L(G))$ terminates at some t and a grammar for $L(G)$ can be picked out of $C - \mathcal{A}$; we will now consider the case where $L(G)$ is infinite. If $H \in C$ is any language such that $L(G) - L(H) = \{y\} \neq \emptyset$, then after the first appearance of a y in $I(L(G))$, H will never be guessed by D. If $H \in C$ is such that $L(G) \subset L(H)$ there is a length k_1 such that for all

$k > k_1$, H generates more strings of length less than or equal to k than G and thus H will not be guessed by D . Thus D will eventually guess only the first grammar $A \in \mathcal{G}$ such that $L(G) = L(A)$ and the lemma is proved.

Thus requiring an information sequence to be bounded has produced a somewhat larger class of inferrable languages. Although some infinite sets of infinite languages can be identified in the limit, the following lemma shows that there are some very simple classes which cannot be identified in the limit from $I \in \mathcal{I}_+$.

Lemma 3b4 The finite state languages are not identifiable in the limit, from $I \in \mathcal{I}_+$.

Proof The proof is an adaptation of Gold's proof of Lemma 3b2. We form a subclass of the finite state languages for which D will change its value an infinite number of times. Let this class $C = \{H_i\}$ be defined as follows.

$L(H_0) = a^*b^*$ (any sequence of a's followed by any sequence of b's)

and

for $i > 0$, $L(H_i) = \bigcup_{j=1}^{i} a^j b^*$

The languages H_i , $i \geq 0$ all have finite state grammars. We will show that for any D(S,FS) which will identify in the limit all the H_i , $i > 0$ there is an $I(H_0)$ which will cause D to change its guess an infinite number of times. The sequence $I(H_0)$ starts with enough $y \in L(H_1)$ to cause D to guess H_1 ; the assumption that D infers H_1

guarantees the existence of such a sample. Then $I(H_0)$ continues with enough $y \in L(H_2)$ to cause D to guess H_2, etc. Any $I(H_0)$ of this nature would cause D to change its guess an infinite number of times.

The class of languages learnable from positive information sequences will now be extended by introducing a weaker notion of learnability. The comparison of the two definitions of learnability will be deferred until after the theorems. For the remainder of Section 3 we will restrict ourselves to bounded information sequences and to the class $\mathcal{R}$ of completely reduced context-free grammars. Several of the results could be made more general, but these are sufficient for our purposes and allow of simpler treatment.

Def 3.5 A language $L(G)$ in a class C is approachable from above by a device D iff for each $H \in C$ such that $L(G) \subset L(H)$ and each information sequence $I(L(G))$, there is a τ such that $t > \tau$ implies

$$D(S_t(I), C) \neq H .$$

Thus a language is approachable from above if every grammar producing a larger language is eventually rejected. We can define approachable from below in a somewhat similar manner:

Def 3.6 A language $L(G)$ in a class C is approachable from below iff for each $H \in C$ such that $L(G) - L(H) \neq \emptyset$ and each $I(L(G))$ there is a τ such that $t > \tau$ implies

$$D(S_t(I), C) \neq H .$$

That is, any grammar H, whose language does not contain $L(G)$ is eventually rejected. This condition is trivially incorporated in any reasonable device for positive information sequences. This is because any $y \in L(G) - L(H)$ will eventually appear in every $I(L(G))$.

Def 3b7 A language $L(G)$ is approachable if it is approachable from above and below. A class $L(C)$ of languages is approachable iff there is a device $D(S,C)$ under which each $L(G) \in L(C)$ is approachable through any $I(L(G)) \in \mathcal{I}_+$.

Theorem 3b8 For any admissable class of grammars $C \subset R$ there is a device $D(S,C)$ such that for any $G \in C$ and $I(L(G)) \in \mathcal{I}_+$, $L(G)$ is approachable through I.

Proof For $L(G)$ finite the problem is trivial. Assume $L(G)$ is infinite. Let $I(L(G)) = \langle y_1, y_2, \ldots \rangle \in \mathcal{I}_+$. Let $\mathcal{G}$ be an enumeration of C and for each G in $\mathcal{G}$ define $n_k(G)$ to be the number of strings of length k generated by the grammar G and

$$N_k(G) = \sum_{j=1}^{k} n_j(G)$$

The device $D(S_t,C)$ proceeds as follows. At each time, t, D will choose the next grammar G_t from $\mathcal{G}$ and the next string $y_t \in I(G)$ forming the sample

$$S_t = S_{t-1} \cup \{y_t\} .$$

It will also compute $\ell_t = \max(\ell(y))$ over $y \in S_t$. The device will also form the set of possible guesses Q_t

$$Q_t = \{G \mid G \in \{G_1, \ldots G_t\} \text{ and } S_t \subset L(G)\} .$$

If α_t is empty, the device will choose more grammars from $\mathcal{G}$ until α_t is non-empty. Finally the device will compute its guess A_t at time t by choosing one of the grammars G in α_t for which $N_{\ell_t}(G)$ is minimal. The procedure for breaking ties is immaterial.

The fact that D is effective follows easily from textbook results. We now show that A_t approaches G from above. That is, if $H \in C$ is such that $L(G) \subsetneq L(H)$ there is a time τ such that

(1) $\qquad t > \tau$ implies $A_t \neq H$.

If $L(G) \subsetneq L(H)$ there is an integer h such that $k \geq h$ implies

$$N_k(G) > N_k(H) .$$

Let τ_1 be the first value of t for which $\ell_t = h$ and τ_2 be the first value of t for which G appears in $\mathcal{G}$. Then $\tau = \max(\tau_1, \tau_2)$ is a finite value of time for which (1) holds. Since $L(G)$ is always approachable from below through any complete positive information sequence, the theorem is proved.

The procedure used by the device D in the proof above can be made more efficient in a number of ways. Since a finite language necessarily has a finite information sequence over $\mathcal{G}_+$, D could restrict its guesses to grammars which produced infinite languages. In practice, one would break ties for A_t by choosing the best grammar relative to some complexity measure such as those of Section 2. The question of inferring "good" grammars will be discussed in Section 3c.

There is a progressive weakening of the formal counterpart of the intuitive concept of "learning a grammar" as one goes from recursive to limiting identifiable to approachable. An inference device which can identify a class of languages in the limit will find a correct grammar, but will not know that it has done so. If the device can approach a class of languages, it may not ever settle on a correct grammar, but will get progressively closer as the sample size grows. Unfortunately, this is the best kind of result possible in the absence of negative information.

The device D used in the proof of Theorem 3b5 could make use of negative strings to reduce the set α_t considered acceptable to time t. One might conjecture that there is a device that would use negative strings in an information sequence without knowing whether or not it was complete (that is, whether all or only some of the negative strings occur) and achieve the behavior of Theorem 3b1 for complete sequences and of 3b8 for incomplete ones. This conjecture is false; an argument similar to the proof of Lemma 3b4 will show that:

Corollary 3b9 If D is a device which will approach any finite state language L(G) for any $I(L(G)) \in \mathcal{S}_+$ then there is a finite state grammar H and an information sequence $I(H) \in \mathcal{S}_+$ which will cause D to change its guess an infinite number of times.

Intuitively, the device of Theorem 3b1 adopts a very conservative strategy; it chooses the first grammar which is compatible with the sample. It succeeds because the negative strings in a complete sample guarantee that any incorrect grammar will ultimately be incompatible. The device of Theorem 3b6 does not have this guarantee, so it must constantly look for

"better" grammars and thus cannot be guaranteed to eventually remain at the same value. The question of learning good grammars and making good guesses is the subject of the next section.

3c. Learning Good Grammars

The preceding discussion has established the solvability of the grammatical inference problem under a variety of conditions. We now extend these results by considering when a good grammar (in the sense of Section 2) can be learned.

There are several properties which would be desirable in an overall measure which was an increasing function of both intrinsic complexity, $\eta(S,G)$ and derivational complexity, $\mu(G,\bar{G})$. For a fixed grammar, the complexity of a sample should be bounded so that the convergence results of Section 2c are applicable. Finally, the relative weight given to the components of the measure should be able to be specified in advance. Another important property of a measure, effectiveness, is actually a consequence of the other requirements and the general conditions of the problem as the following lemma and theorem will show.

Lemma 3c1 Let $\mathcal{G} = (G_i)$ be any enumeration of a class $C \subset R$ which is approximately ordered by length and let S_t be a sample of some $I(L(G))$, $G \in \mathcal{G}$. Then there is a computable index k such that $j > k$ implies there is an $h \leq k$ such that

$$\eta(S_t,G_h) \leq \eta(S_t,G_j) \quad .$$

Proof The proof is based on the fact that if a grammar is too large, there must be some redundant rules. Let

$$U(S_t) = \sum_{i=1}^{t} 2 \cdot \ell(y_i) .$$

From Lemma 1b2 we know that the total number of uses of productions in deriving S_t is less than $U(S_t)$. Therefore, if one chooses an index k such that $j > k$ implies the number of productions in G_j is greater than $U(S_t)$, the condition of the lemma is satisfied. Such a k is computable since $\mathcal{G}$ is effectively approximately ordered by the length of grammars.

Theorem 3c2 Let $C \subset R$ and $\mathcal{G} = \{G_i\}$ be an effective approximate ordering of C by $\mu(G,\bar{G})$. Also let $f(\eta(S,G),\mu(G,\bar{G}))$ be any monotonic function of both its arguments. Then for any $G \in C$, $S_t \subset I(G)$ there is a computable index k such that any grammar G_i such that

$$f(\eta(S_t,G_i),\mu(G_i,\bar{G})) \text{ is minimal}$$

has an index $i < k$ in $\mathcal{G}$.

Proof By lemma 3c1 above, there is a k_1 such that the G_i minimizing $\eta(S_t,G)$ occur before k_1. Let M be the largest value of $\mu(G_i,\bar{G})$ occurring before k_1, i.e.,

$$M = \max_{1 \leq i \leq k_1} (\mu(G_i,\bar{G})) .$$

Now, by lemma 2b5 there is an index k such that $j > k$ implies

$$\mu(G_j,\bar{G}) > M .$$

The minimum value of $f(\eta(S_t,G_i),\mu(G_i,\bar{G}))$ must occur with index less than k. since for each $j > k$ there is an $h \leq k$ such that both $\eta(S_t,G_h) < \eta(S_t,G_j)$ and $\mu(G_h,\bar{G}) < \mu(G_j,\bar{G})$.

The requirement that a goodness measure be an increasing function of both intrinsic complexity $\mu(G,\bar{G})$ and derivational complexity $\eta(S,G)$ seem to be a natural one. The particular choice of a goodness function is less clear. Consider a device D which enumerates the class C of candidate grammars by generating them in order of length from $\bar{G}$. Although $\eta(S,G)$ is a normalized complexity measure and is bounded for a fixed grammar, the bound increases approximately as the length of grammars. Although $\mu(G,\bar{G})$ also increases with length it does so in a different manner. A comparison between the growth rates of $\mu(G,\bar{G})$ and $\eta(S,G)$ would be very helpful in choosing a goodness function. In the absence of any knowledge of growth rates, we will be content to use a particular class of goodness functions which seems reasonable.

Def 3c3 A goodness measure $\gamma(S,G)$ is defined as

$$\gamma(S,G) = a \cdot \eta(S,G) + b \cdot \mu(G,\bar{G})$$

where $0 \leq a,b \leq 1$.

It follows from previous results that goodness measure γ is an increasing function of η, μ and is bounded for fixed G. By Theorem 3c2, the minimum $\gamma(S,G)$ for fixed S and $G \subset C$, a complexity class, is effectively computable. Thus γ is an adequate goodness measure by the criteria laid down above. We now study the conditions under which best grammars, as measured by γ, can be learned by an effective device $D(S,C)$.

Theorem 3c4 Under the conditions of Theorem 3b2 ($G \in C$, $I(L(G)) \subset \vartheta$) . If $\gamma(S_t,G_i)$ converges as $t \to \infty$ for every G_i such that $L(G_i) = L(G)$ then there is a device $D(S_t,C)$ which will identify in the limit the grammar G_j such that $L(G_j) = L(G)$ and $\gamma(L,G_j)$ is minimal over C .

Proof The device D will use $\bar{G}$ for the enumeration $\mathcal{E}$ of C as before and will at each time t form S_t . There is a first G_q which is compatible with S_t and by lemma 3c2 there is a $k_1(q)$ such that $i > k_1$ implies $\gamma(S_t,G_i) > \gamma(S_t,G_q)$. The device D then chooses the first grammar in $\{G_1,\ldots G_{k_1}\}$ which has the minimal value of γ as its guess A_t .

Now there is a first G_q such that $L(G_q) = L(G)$ and $\gamma(L(G),G_q) = c_q$ exists. But there is also an index $k(q)$ such that $i > k(q)$ implies $b \cdot \mu(G_i,\bar{G}) \geq c_q$, i.e., intrinsic complexity alone exceeds c_q at some point.

Thus the device D will never consider more than the grammars $G_1,\ldots G_k$ as possible guesses. Any G_i such that $L(G_i) \neq L(G)$ will eventually be eliminated by the complete information sequence I(G) . There are then a finite number of G_i , all of which generate L(G) ; for each of these, $\gamma(S_t,G_i)$ converges to a limit c_i . Let the first occurrence of the minimum $(c_i) = c_j$ be a G_j . For any G_i such that $c_i = c_j+e$ there is an index $r(i)$ after which $\gamma(S_{r(i)},G_i) > \gamma(S_{r(i)},G_j)$. Let w be the largest of the $r_{(i)}$, then for all $t > w$ the guess A_t will be precisely G_j and the theorem is proved.

Corollary 3c5 If the measure $\gamma(S_t,G) = \mu(G,\bar{G})$, (only intrinsic complexity is considered) the device of Theorem 3c4 will always identify the best grammar in the limit, the grammar of lowest intrinsic complexity producing the correct language.

Corollary 3c6 The device of Theorem 3c4 will approach the best grammar, even if the limit of $\gamma(S_t,G)$ does not exist.

The requirement that the limit of $\gamma(S_t,G)$ exist seems to be necessary in general. If γ does not converge, the device can be caused to oscillate its guesses between a finite number of different grammars for the target language. There is a possibility that for complete information sequences, $\gamma(S_t,G)$ can always be made to converge. It is based on the following conjecture: the measure $\gamma(S_t,G)$ will always converge on an information sequence which presents strings in strict order of length. If the conjecture is true then the device of Theorem 3c4 would be able to wait until all positive and negative strings of length up to k were seen, then compute $\gamma(S_{t_k},G)$ and be assured of convergence.

The final set of questions relate to the learning of best grammars from positive information sequences. In the discussion of Theorem 3b5 we remarked that a goodness measure like γ could be used to break ties among compatible grammars producing the minimum number of strings of a fixed length, q. The device described there will approach the correct grammar, but will not make the best guess at each time, t. By making q a slowly increasing function of t one can produce a device which will tend to produce better guesses at each time, t, at the cost of rejecting overbroad grammars later in the sequence. One

might conjecture that the complexity measure alone would eventually eliminate overbroad grammars. We now present an example to show that a device using only the complexity measure γ and a positive information sequence may fail to approach the correct grammar.

Example 3c7.

Let $\mathbf{C}$ be $L(FS) \cap \mathbf{R}$, the finite state grammars in standard form. Let

$$G = (\{X\},\{a,b\},X,\{X \to a|b|aX|bX\}) .$$

The universal grammar of Examples 2a5, 2b4 has $\mu(G,FS_1) = 8$ and the upper bound on $\eta(S,G)$ is $\log_2(4) = 2$. In fact, for this simple grammar $\eta(S,G)$ is exactly 2. Thus for any set $S_t \subset L(G)$

$$\gamma(S_t,G) = 10 .$$

We now show that by removing one string from $L(G)$ we get a language L' such that for any H such that $L' = L(H)$ and any sample $S_t \subset L'$

$$\gamma(S_t,H) > \gamma(S_t,G) .$$

That is, any device using γ as a selection criterion will select the universal grammar G over the correct grammar H. To prove this rigorously we would have to account for all possible grammars of L' (which the results of this section show to be possible) but we will be content with the following argument.

Consider $L' = L(G)$-aaaaaaaa. Any grammar of L' that is in $\mathbf{C}$ can have only one terminal symbol per production. It must also have enough states (non-terminals) to count to eight. This apparently requires a grammar with $\gamma > 10$.

In any event, there is a string of a's long enough so that its unique non-membership requires a grammar of intrinsic complexity greater than 10. This example also indicates that the difference of two grammars might have a lower measure than any single grammar of the class, even when such a grammar exists. This question of combinations of grammars deserves considerably more attention.

3d. Using Frequency Information to Assist Inference

Previous sections have presented successively weaker definitions of learnability: recursive, identifiable in the limit, approachable. All of these definitions are "strong", however, in that they require that the device (eventually) satisfy the criterion for every information sequence in some class. In fact, the non-learnability results of Theorem 3b2, Lemma 3b4, and Corollary 3b9 depend upon the construction of particular pathological information sequences.

In practice, however, a device whose performance is superior on "most" information sequences need not be rejected because it fails on a few sequences, provided that they are "sufficiently improbable". We are generally more interested in the "expected behavior" of a device than in its worst case behavior. To study these properties of devices we must define more carefully our notions of "most", "sufficiently improbable", and "expected behavior". In this section we start with a probabilistic notion of information sequence, which leads naturally to a Bayesian inference device using the frequency of occurrence of strings to assist in inference. We also sketch a number of basic results which will be explored further in [Horning 69].

There are many other motivations for using the frequencies of the strings in a positive information sequence (text presentation) to assist in grammatical inference:

(a) Since more information from the sequence is used, grammars may be discriminated earlier.

(b) The significance of "missing strings" can be evaluated.

(c) Inference can be conducted even in the presence of noise.

(d) Grammars for the same language may be discriminated on the basis of their agreement with observed frequencies.

(e) Complexity can be related to efficient encoding, and various results from information theory applied.

We shall assume that the elements of an information sequence are independent and identically distributed random variables (iidrv condition).

Lemma 3d1 The iidrv condition implies convergence with probability $> 1 - \epsilon$ for any $\epsilon > 0$.

Proof See sequel.

Let $\pi = \{\pi_1, \pi_2, \ldots\}$ be a denumerable set of probability distributions for strings in T^+ such that the conditional probability of a string, $P(y_i|\pi_j)$, and the a priori probability of a distribution, $P(\pi_j)$, are both computable. Under the iidrv condition, the partial information sequence

$$I_k(t) = \langle y_{k1}, y_{k2}, \ldots, y_{kt} \rangle$$

has the conditional probability

$$P(I_k(t)|\pi_j) = \prod_{\tau=1}^{t} P(y_{k\tau}|\ \pi_j)$$

$$= \prod_i P(y_i|\pi_j)^{f(I_k,y_i,t)}$$

As is well-known, the probability distribution for information sequences under the distribution π' for strings corresponds to the multinomial $(P(y_1|\pi') + P(y_2|\pi') + \ldots)^t$ or, distinguishing $P(y_i|\pi')$, to the binomial

$$\text{I)} \quad (P_i + \Sigma_i')^t = \sum_{fi} \binom{t}{fi} P_i^{fi} \cdot [\Sigma_i']^{t-fi}$$

where

$$P_i = P(y_i|\pi'), \quad \Sigma_i' = \sum_j P(y_j|\pi') - P_i .$$

Taking $\partial/\partial P_i$ of both sides:

$$\text{II)} \quad t \cdot (P_i + \Sigma_i')^{t-1} = \sum_{fi} fi \binom{t}{fi} P_i^{fi-1} \cdot [\Sigma_i']^{t-fi}$$

Multiplying by P_i :

$$\text{III)} \quad P_i \cdot t \cdot (P_i + \Sigma_i')^{t-1} = \sum_{fi} fi \binom{t}{fi} P_i^{fi} \cdot [\Sigma_i']^{t-fi}$$

Again taking $\partial/\partial P_i$ and multiplying by P_i

$$\text{IV)} \quad P_i \cdot t \cdot (P_i + \Sigma_i')^{t-2} \cdot [(P_i + \Sigma_i') + P_i \cdot (t-1)]$$

$$= \sum_{fi} fi^2 \binom{t}{fi} P_i^{fi} \cdot [\Sigma_i']^{t-fi} .$$

Since $P_i + \Sigma'_i = 1$ we can simplify III) and IV):

III)' $$\sum_{fi} ri\binom{t}{fi} \cdot P_i^{fi} \cdot [\Sigma'_i]^{t-fi} = P_i \cdot t$$

IV)' $$\sum_{fi} f_i^2\binom{t}{fi} \cdot P_i^{fi} \cdot [\Sigma'_i]^{t-fi} = P_i \cdot t \cdot (1 + P_i \cdot (t-1)) .$$

The left sides of these equations define expectation values under π' for f_i and f_i^2 so we have

V) $$E\pi'(f(I,y_i,t)/t) = P(y_i|\pi')$$

VI) $$E\pi'([f(I,y_i,t)/t - P(y_i|\pi')]^2)$$

$$= E\pi'([f(I,y_i,t)/t]^2) - 2E_{\pi'}(f(I,y_i,t)/t)$$

$$* P(y_i|\pi') + P(y_i|\pi')^2$$

$$= P(y_i|\pi') * t * (1 + P(y_i|\pi') * (t-1))/t^2 - P(y_i|\pi')^2$$

$$= [P(y_i|\pi')^2 + P(y_i|\pi')]/t$$

Equation VI) defines the expected variance of f_i/t . Since $P(y_i|\pi') \leq 1$ we can bound it by

VI)' $$E_{\pi'}([f(I,y_i,t)/t - P(y_i|\pi')]^2) \leq 2\, P(y_i|\pi')/t .$$

We can use this to bound $\epsilon_i(\delta)$, the probability of an information sequence with $|f(I,y_i,t)/t - P(y_i|\pi')| \geq \delta$

VII) $$\epsilon_i(\delta) \cdot \delta^2 \leq E_{\pi'}([f(I,y_i,t)/t - P(y_i|\pi')]^2)$$

$$\leq 2\, P(y_i|\pi')/t$$

and $\epsilon(\delta)$, the probability that <u>any</u> f_i/t is off by δ or more.

VIII) $$\epsilon(\delta) \leq \sum_i \epsilon_i(\delta) \leq 2/(t \cdot \delta^2) .$$

Given any $\epsilon > 0$, $\delta > 0$ if $\tau = 2/\epsilon\delta^2$ then $t > \tau$ assures that the total probability of information sequences of length t in which the relative frequency of any string deviates by δ or more from its probability in π' is less than ϵ. This completes the proof of Lemma 3d1: "The iidrv condition implies convergence with probability $> 1 - \epsilon$ for any $\epsilon > 0$." It is in fact a slightly stronger result, because we have also showed the relative frequency distribution to which "practically all" sequences converge is π', the distribution of the random variable.

Returning to the case of a fixed information sequence, we note that Bayes Theorem can be used to compute the conditional probability of a distribution

$$P(I_k(t),\pi_j) = P(I_k(t)|\pi_j) \cdot P(\pi_j) = P(\pi_j|I_k(t)) \cdot P(I_k(t))$$

or

$$P(\pi_j|I_k(t)) = P(I_k(t)|\pi_j) \cdot \frac{P(\pi_j)}{P(I_k(t))}$$

where

$$P(I_k(t)) = \sum_{\pi_\ell \in \pi} P(\pi_\ell) \cdot P(I_k(t)|\pi_\ell)$$

and

$$P(I_k(t)|\pi_j) = \prod_i P(y_i|\pi_j)^{f(I_k,y_i,t)} \quad .$$

To use this formulation for grammatical inference we must relate the probability distributions π_j and the a priori probabilities $P(\pi_j)$ to grammatical complexity.

At each step of a derivation a production -- one of the finite set with the correct left part -- is selected. If production p_i is selected from this set with probability $P(p_i)$, the specification requires $\rho(p_i) = -\log_2(P(p_i))$ bits of information. The probability of a derivation is the product of the probabilities of its individual steps, so if $d(y,G) = \langle p_1, \ldots p_k \rangle$ then $P(d(y,G)) = \prod_{i=1}^{k} P(p_i)$ and $-\log_2(P(d(y,G)) = \mathcal{M}(d,y,G)$ where (as before) $\mathcal{M}(d,y,G) = \sum_{i=1}^{k} \rho(p_i)$.

Def 3d2 Let $y \in T^+$, if $y \notin L(G)$ we define the conditional probability $p(y|G)$ to be zero; if $y \in L(G)$ and has the derivations $d_1(y,G) \ldots d_k(y,G)$ we define

$$P(y|G) = \sum_{i=1}^{k} P(d_i(y,G)) \quad .$$

Let $\hat{\mu}(y,G) = -\log_2(P(y|G))$. If y is unambiguous with respect to G then $k = 1$ and $\hat{\mu}(y,G) = \mu(y,G)$ [Def 2a3]; in the ambiguous case, $\hat{\mu}$ provides at least as plausible a definition of complexity as does μ .

As we did in Section 2b, we define the intrinsic complexity of a grammar in terms of its derivation from a grammar-grammar. Note, however, that for our purposes, grammars which differ only in the order of their productions, or in the systematic renaming of their non-terminals (except the distinguished non-terminal!) are completely equivalent. The equivalence class of a grammar with k productions and n non-terminals contains $k!(n-1)!$ equi-probable grammars. We are always interested in

$$P(\pi_j|\bar{G}) = k!(n-1)!\ P(G_j|\bar{G}_n) \cdot P(\bar{G}_n)$$

since all of these grammars yield the same distribution, π_j . For a fixed collection we must specify the probability of $\bar{G}_n$ with n non-terminals. A reasonable choice is $P(\bar{G}_n) = 2^{-n}$.

$$P(\pi_j) = k!(n-1)! \cdot P(G_j|\bar{G}_n) \cdot P(\bar{G}_n)$$

$$= k!(n-1)! \cdot 2^{-\hat{\mu}(G_j,\bar{G}_n)} \cdot 2^{-n}$$

Define

$$\bar{\mu}(G_j,\bar{G}_n) = \hat{\mu}(G_j,\bar{G}_n) + n - \log_2(t!(n-1)!)$$

then

$$P(\pi_j) = 2^{-\bar{\mu}(G_j,\bar{G}_n)}$$

$$P(y_i|\pi_j) = 2^{-\hat{\mu}(y_i,G_j)}$$

By our formula for conditional probability

$$P(\pi_j|I_k(t)) = \frac{2^{-\bar{\mu}(G_j,\bar{G})} \cdot \prod_i [2^{-\hat{\mu}(y_i,G_j)}]^{f(I_k,y_i,t)}}{P(\bar{I}_k(t))}$$

Taking logarithms

$$-\log_2(P(\pi_j|I_k(t))) = \bar{\mu}(G_j,\bar{G}) + \log_2(P(I_k(t))) + \sum_i f(I_k,y_i,t) \cdot \hat{\mu}(y_i,G_j)$$

Except for a term independent of the grammar $(\log_2(P(I_k(t))))$, this corresponds rather closely to our previous measure of fit [Def 2b5], weighted by the frequency of occurrence of strings. Let $\hat{\mu}(I_k(t),G_j) = \sum_i f(I_k,y_i,t) \cdot \hat{\mu}(y_i,G_j)$ and $\hat{M} = -\log_2(P(\pi_j|I_k(t)))$, then

$$\hat{M}(\pi_j,I_k(t)) = \bar{\mu}(G_j,\bar{G}) + \hat{\mu}(I_k(t),G_j) + \log_2(P(I_k(t))) .$$

To compute $P(I_k(t))$ we must enumerate the distributions $\pi_1,\pi_2,\ldots$

$$P(I_k(t)) = \sum_j P(\pi_j) \cdot P(I_k(t)|\pi_j) .$$

This is not generally practical. However, this term drops out when we compare the relative probabilities of grammars

$$\frac{P(\pi_j|I_k(t))}{P(\pi_\ell|\bar{I}_k(t))} = 2^{-[\hat{M}(\pi_j,I_k(t))-\hat{M}(\pi_\ell,I_k(t))]} = 2^{-[\bar{M}(\pi_j,I_k(t))-\bar{M}(\pi_\ell,I_k(t))]}$$

where

$$\tilde{M}(\pi_j, I_k(t)) = \tilde{\mu}(G_j) + \hat{\mu}(I_k(t), G_j) \ .$$

As in Section 2c, the grammar with the smallest total complexity $\tilde{M}$ is preferred.

We can compute a lower bound for $\hat{\mu}(I_k(t), G_j)$, independent of the particular class of grammars involved, by the method of La Grange

$$L = \lambda \cdot \sum_i P(y_i|G) + \sum_i \hat{\mu}(y_i, G) \cdot f(I_k, y_i, t)$$

$$= \lambda \cdot \sum_i P(y_i|G) - \sum_i \log_2[P(y_i|G)] \cdot f(I_k, y_i, t)$$

$$\frac{\partial L}{\partial P(y_i|G)} = \lambda - \frac{f(I_k, y_i, t)}{P(y_i|G)} = 0$$

$$P(y_i|G) = f(I_k, y_i, t)/\lambda \quad .$$

But

$$\sum_i P(y_i|G) = 1$$

$$\sum_i f(I_k, y_i, t)/\lambda = 1$$

$$\lambda = \sum_i f(I_k, y_i, t) = t$$

$$P(y_i|G) = f(I_k, y_i, t)/t \quad .$$

Substituting, we have

$$\hat{\mu}_{min}(I_k(t)) = -\sum_i \log_2[f(I_k,y_i,t)/t] \cdot f(I_k,y_i,t)$$

$$= t \cdot \log_2(t) - \sum_i f(I_k,y_i,t) \cdot \log_2(f(I_k,y_i,t))$$

$$\hat{\mu}(I_k(t),G_j) \geq t \cdot H(I_k(t))$$

where

$$H(I_k(t)) = -\sum_i \left[\frac{f(I_k,y_i,t)}{t}\right] \cdot \log_2\left[\frac{f(I_k,y_i,t)}{t}\right]$$

is a local "entropy" measure. It would seem that $\hat{\mu}_{min}$ is a "natural" normalization for complexities.

We may, in the course of inference, require an estimate of $\hat{M}$ (as well as the value of $\bar{M}$) without enumerating the π_j.

$$\hat{M}(\pi_j,I_k(t)) = \bar{M}(\pi_j,I_k(t)) + \log_2[P(I_k(t))]$$

$$P(I_k(t)) = \sum_j P(\pi_j) \cdot P(I_k(t)|\pi_j)$$

In general, we will know some $\{\pi_r\}$ which have been rejected -- because $P(I_k(t)|\pi_r) = 0$ -- and some $\{\pi_c\}$ which are under consideration.

Let

$$P_r = \sum_{\pi_j \in \{\pi_r\}} P(\pi_j) \quad , \quad P_c = \sum_{\pi_j \in \{\pi_c\}} P(\pi_j)$$

$$P_u = 1 - P_r - P_c \quad , \quad P_c(I_k(t)) = \sum_{\pi_j \in \{\pi_c\}} P(\pi_j) \cdot P(I_k(t)|\pi_j)$$

then

$$P_c(I_k(t)) \leq P(I_k(t)) \leq P_c(I_k(t)) + P_u \cdot 2^{-t \cdot H(I_k(t))}$$

Thus, although our inference measure can never be "sure", it can compute a confidence measure for its best grammar.

Noise

If the distribution of noise (error) strings is known, i.e., π_n and P_n are given such that elements of the information sequence are drawn with probability P_n from the distribution π_n and probability $(1 - P_n)$ from the "true" distribution π_T then we have

$$P(y_i|\pi_j, P_n, \pi_n) = (1 - P_n)P(y_i|\pi_j) + P_n \cdot P(y_i|\pi_n) .$$

We can substitute this for $P(y_i|\pi_j)$ in all of our formulas and still conduct inference.

If P_n is small, we will introduce very little error by the approximation

$$P(y_i|\pi_j, P_n, \pi_n) \approx \begin{cases} P(y_i|\pi_j) & \text{if} \quad P(y_i|\pi_j) > 0 \\ P_n \cdot P(y_i|\pi_n) & \text{otherwise} \end{cases}$$

i.e., strings not generated by the grammar are given their "noise" probabilities, otherwise noise is ignored.

BLANK PAGE

4. Programs for Grammatical Inference

4a. Introduction and Definition of Pivot Grammars

The development of programs for grammatical inference provided the original motivation for the theoretical work presented above and is of continuing interest. The programs completed so far are quite primitive and were written to test some basic ideas. There are a number of obvious extensions. Given a proper formulation, the grammatical inference problem can be characterized as a heuristic search problem and the various known techniques [Newell 68] applied.

An early paper [Feldman 67] described a number of strategies for inferring finite state and linear grammars. They can be characterized as constructive as opposed to the enumerative strategies stressed in this paper. Thus they solve the problem "Build a reasonable grammar for ..." rather than "Find the best grammar for ...". The first program, GRIN1, embodies these strategies in an inference program for finite state grammars. Rather than extend these simple techniques to linear grammars we considered the problem for a somewhat more general class: the pivot grammars. A pivot grammar is an operator grammar in which a terminal symbol which separates non-terminals in a production appears in no other way. More formally:

Def 4a1 A pivot grammar $G = (V,T,X,P)$ is a grammar in operator 2-form (cf. Section 2b) such that the set of terminal symbols, T, is partitioned into two sets T_p, T_o such that

1) $a \in T_p$ implies a appears only in rules of the form

$$Z_1 \rightarrow Z_2\ a\ Z_3$$

2) $a \in T_o$ implies a appears only

$Z_1 \rightarrow a\ Z_2$

or $Z_1 \rightarrow Z_3\ a$

or $Z_1 \rightarrow a$.

The linear grammars are exactly the pivot grammars for which $T_p = \phi$. The pivot languages are much broader than the linear languages. For example, the following pivot grammar defines a language which is not generated by any linear grammar.

Example 4a2 Let $G = (V,T,X,P)$ where

$V = \{X,Z_1,Z_2,(,),-,a\}$

$T = \{(,),-,a\}$

and P contains the production rules

$X \rightarrow Z_1 - Z_1$

$Z_1 \rightarrow (Z_2 | a$

$Z_2 \rightarrow X)$.

Sample strings from L(G) include

a-a, (a-a)-a, (a-(a-a)) - (a-a)

The context-free grammars used to define programming languages are, for the most part, expressible in pivot form. The principal problems are situations like the use of '-' as both a unary and infix binary operator. Our interest in pivot grammars arises from the relative ease with which they are inferred. The second program described below, GRIN2, is an inference device for pivot grammars.

The programs described below are implementations of only our most basic ideas on grammatical inference. No use is made of ill-formed

strings or frequency information. The entire program is situation-static in three important ways.

1) Only one set of strings is presented, no new strings are added.

2) The program does not propose new strings for outside appraisal.

3) The algorithms themselves are deterministic, with no backtracking.

The addition of these and various other features would be straightforward but time-consuming. In the absence of a pressing need for grammatical inference programs, we will continue to concentrate on the theoretical and programming questions which seem to be most basic. A formulation of grammatical inference as a general heuristic search problem will be presented after the current programs are described.

4b. Program descriptions

GRIN1 infers an unambiguous finite state grammar for the set of terminal symbol strings. The program is an implementation of the algorithm proposed in [Feldman 67]. The algorithm is merely sketched here; the reader is directed to the original source for a more complete version and further examples.

The input to the program is a list of symbol strings. The output of the program is a finite state grammar, the language of which is a "reasonable" generalization of these strings.

All of the productions of the final grammar are of the form:

$$Z_1 \rightarrow a\ Z_2$$

or

$$Z_1 \rightarrow a$$

where Z_1, Z_2 are non-terminals

a is a terminal.

The program temporarily utilizes other productions ("Residues") of the form:

$Z_1 \rightarrow a_1\ a_2\ a_3 \cdots a_n$ where $a_1, a_2, \ldots, a_n$ are terminals.

At all times during the inference process a non-terminal has either all residue or all non-residue right sides (e.g. it will not construct productions $Z_1 \rightarrow a_1 Z_2$ and $Z_3 \rightarrow a_2 a_3$ where Z_1, Z_2, Z_3 are non-terminal, a_1, a_2, a_3 are terminals, $Z_1 \equiv Z_3$).

In the explanation of the algorithm, the set of strings {caaab, bbaab, caab, bbab, cab, bbb, cb} will be used as an example. X will be the distinguished non-terminal in the grammar to be constructed.

The main strategy of the algorithm is to first construct a non-recursive grammar that generates exactly the given strings, and then to merge non-terminals to get a simpler, recursive grammar that generates an infinite set of strings.

The algorithm has been divided into three parts. Part 1 forms the non-recursive grammar, Part 2 converts this to a recursive grammar which is then simplified by Part 3.

In Part 1, a non-recursive grammar that generates exactly the given sample is constructed. Sample strings are processed in order of decreasing length. Rules are constructed and added to the grammar as they are needed to generate each sample string. The final rule used to generate the longest sample strings is a residue rule with a right side of length 2.

In the example, the first (longest) string in the example is 'caaab' . The following rules would be constructed to generate this string:

$$X \rightarrow cZ_1$$

$$Z_1 \rightarrow aZ_2$$

$$Z_2 \rightarrow aZ_3$$

$$Z_3 \rightarrow ab$$

Z_3 is a residue rule. The second string is 'bbaab' . The following rules would be added to the grammar to generate this string:

$$X \rightarrow bZ_4$$

$$Z_4 \rightarrow bZ_5$$

$$Z_5 \rightarrow aZ_6$$

$$Z_6 \rightarrow ab$$

Z_6 is a residue rule. To generate the third string, 'caab' , the following rule must be added to the grammar:

$$Z_3 \rightarrow b \quad .$$

Proceeding to consider each string in turn we see that the final grammar that is constructed to generate exactly the sample is:

$$X \rightarrow cZ_1 \mid bZ_4$$

$$Z_1 \rightarrow b \mid aZ_2$$

$$Z_2 \rightarrow b \mid aZ_3$$

$$Z_3 \rightarrow b \mid ab$$

$$Z_4 \rightarrow b\ Z_5$$

$$Z_5 \rightarrow b \mid aZ_6$$

$$Z_6 \rightarrow b \mid ab \quad .$$

The residue rules are Z_3 and Z_6 .

In Part 2 a recursive finite state grammar is obtained by merging each residue rule with a non-residue rule of the grammar. The algorithm is

conservative in deciding which non-residue rule should be substituted for a residue rule. The general principle is that after such a substitution the resulting grammar must generate all that the old grammar could plus as few new (short) strings as possible. Wherever the residue non-terminal occurs on the right side of a production, the non-residue non-terminal is substituted. The resulting grammar is recursive and generates an infinite set of strings.

In the example, Z_6 would be merged with Z_5 and Z_3 would be merged with Z_2. The resulting grammar is:

$$X \rightarrow cZ_1 \mid bZ_4$$
$$Z_1 \rightarrow b \mid aZ_2$$
$$Z_2 \rightarrow b \mid aZ_2$$
$$Z_4 \rightarrow bZ_5$$
$$Z_5 \rightarrow b \mid aZ_5 \quad .$$

In Part 3 the grammar from Part 2 is simplified. Equivalent productions are recursively merged. Productions P_m and P_n with left sides Z_m and Z_n are equivalent iff the substitution of Z_m for all occurrences of Z_n in P_n and P_m results in P_n being identical to P_m. By merging P_m and P_n we mean eliminating production P_n from the grammar and substituting Z_m for all remaining occurrences of Z_n. Merging equivalent productions results in no change in the language generated by the grammar.

In the example, the productions with left sides Z_1 and Z_2 are clearly equivalent. After merging Z_1 and Z_2 the new grammar is:

$$X \rightarrow cZ_1 \mid bZ_4$$
$$Z_1 \rightarrow b \mid aZ_1$$
$$Z_4 \rightarrow bZ_5$$
$$Z_5 \rightarrow b \mid aZ_5 \quad .$$

In this grammar, the productions for Z_1 and Z_5 are equivalent: No change in the generated language results from merging Z_1 and Z_5. The new grammar is:

$$X \rightarrow cZ_1 \mid bZ_4$$
$$Z_1 \rightarrow b \mid aZ_1$$
$$Z_4 \rightarrow bZ_1$$

No further merges are possible; this is the final grammar. Note that the seven shortest strings of its language (cb, bbb, cab, bbab, caab, bbaab, caaab) are precisely the strings constituting the sample set.

The program is usually able to infer a grammar which is subjectively reasonable. Several sample runs are listed in Appendix C. The program for pivot grammars, GRIN2, makes use of many of the same techniques.

GRIN2 infers a pivot grammar for a set of terminal symbol strings. In the explanation of the algorithm, the set of strings {a-a, a-(a-a), (a-a)-a, (a-a)-(a-a), a-(a-(a-a)), a-((a-a)-a), (a-(a-a))-a, ((a-a)-a)-a} will be used as an example. X will be taken as the distinguished nonterminal in the grammar to be constructed. It will be assumed that the minus sign is known to be the only pivot terminal symbol in the strings. There are rules for determining which terminal symbols can be a pivot terminal, e.g. (1) A pivot terminal cannot be the first or last symbol of a string. (2) Occurrences of pivot terminals must be separated by at least one non-pivot terminal in each string. These rules are not used here.

The algorithm has two inputs: the list of known strings and a list of the pivots. The output of the algorithm is a pivot grammar.

The main strategy of the algorithm is to find the self-embeddings in the strings. A non-terminal is set aside as the loop non-terminal (LOOPNT). The self-embeddings in the strings will correspond to the appearance of the loop non-terminal in recursive rules in the grammar. Initially, the loop non-terminal is the distinguished non-terminal.

The algorithm has been divided into three parts. Part 1 finds self-embeddings and creates a working set of strings, Part 2 makes some changes in the working set from which it builds a pivot grammar which is then simplified in Part 3.

In Part 1 a working set of strings is built. Each string is examined to see if it has a proper substring which is also a member of the sample set (a valid substring). If it does not it is simply copied into the working set. If a string does have any valid substrings then the longest valid substring is replaced by an instance of LOOPNT and the new string is placed in the working set. Table 1 gives the longest valid substring and the resulting new string for each of the strings in the example set. X , the distinguished non-terminal, is the initial loop non-terminal. If any substitutions have been made, Part 2 of the algorithm is entered.

If no strings have valid substrings, it is determined whether all the strings have an identical first or last symbol. If there is a common first or last symbol, say 'a' , then a rule of the form LOOPNT → aZ or LOOPNT → Za (and possibly LOOPNT → a) is entered in the grammar; LOOPNT is set to Z ; the first or last symbol is removed for each of the strings and the substitution for longest valid substrings is begun again.

TABLE 1

given strings	longest valid substring	new strings
a-a	none	a-a
a-(a-a)	a-a	a-(X)
(a-a)-a	a-a	(X)-a
(a-a)-(a-a)	a-a	(X)-(a-a)
(a-(a-a))-a	a-(a-a)	(X)-a
((a-a)-a)-a	(a-a)-a	(X)-a
a-(a-(a-a))	a-(a-a)	a-(X)
a-((a-a)-a)	(a-a)-a	a-(X)

Results of Part 1 of GRIM2

In Part 2 further substitutions are made for valid substrings and a simple pivot grammar is constructed.

Each of the strings in the working set is examined independently. If a string contains a pivot terminal, the test and substitution process is repeated for the symbols on the side of the pivot not containing the loop non-terminal. In the example, this would result in a substitution of 'X' for the valid substring 'a-a' in the string '(X)-(a-a)'. The working set of strings would now be {a-a,a-(X),(X)-a,(X)-(X)}.

A simple pivot grammar is constructed for the working set of strings. The working strings are processed in succession; productions are created as they are needed to generate one of the new strings. Recall that pivot symbols can only appear in pivot rules. 'LOOP' is used as the starting point in the generation process.

In the example, the first new string, 'a-a' , would result in the productions:

$$X \rightarrow Z_1 - Z_2$$

$$Z_1 \rightarrow a$$

$$Z_2 \rightarrow a \quad .$$

To generate 'a-(X)' the productions

$$Z_2 \rightarrow (Z_3$$

and

$$Z_3 \rightarrow X)$$

must be added. The productions are now:

$$X \rightarrow Z_1 - Z_2$$

$$Z_1 \rightarrow a$$

$$Z_2 \rightarrow a \mid (Z_3$$

$$Z_3 \rightarrow X) \quad .$$

To generate '(X)-a' the productions

$$Z_1 \rightarrow (Z_4$$

and

$$Z_4 \rightarrow X)$$

must be added. The productions are now:

$$X \rightarrow Z_1 - Z_2$$
$$Z_1 \rightarrow a \mid (Z_4$$
$$Z_2 \rightarrow a \mid (Z_3$$
$$Z_3 \rightarrow X)$$
$$Z_4 \rightarrow X) \quad .$$

To generate '(X)-(X)' no further productions need be added.

These productions are added to any productions constructed in Part 1. In the example there were no productions constructed in Part 1; the grammar outputted from Part 2 is:

$$X \rightarrow Z_1 - Z_2$$
$$Z_1 \rightarrow a \mid (Z_4$$
$$Z_2 \rightarrow a \mid (Z_3$$
$$Z_3 \rightarrow X)$$
$$Z_4 \rightarrow X)$$

In Part 3 the grammar from Part 2 is simplified in the same way as in Part 3 of GRIN1; equivalent productions are recursively merged. The language generated by the grammar remains constant.

In the example, the productions $Z_3 \rightarrow X)$ and $Z_4 \rightarrow X)$ are equivalent. $Z_4 \rightarrow X)$ is eliminated and Z_3 is substituted for all occurrences of Z_4 in the grammar. The resulting grammar is

$$X \rightarrow Z_1 - Z_2$$
$$Z_1 \rightarrow a \mid (Z_3$$
$$Z_2 \rightarrow a \mid (Z_3$$
$$Z_3 \rightarrow X) \quad .$$

In the new grammar $Z_1 \rightarrow a|(Z_3$ and $Z_2 \rightarrow a|(Z_3$ have identical right sides. $Z_2 \rightarrow a|(Z_3$ is eliminated and Z_1 is substituted for Z_2. The resulting grammar is

$$X \rightarrow Z_1 - Z_1$$
$$Z_1 \rightarrow a \mid (Z_3$$
$$Z_3 \rightarrow X) \quad .$$

None of these productions are equivalent; this is the final grammar. Note that the language generated by this grammar is identical to the language generated by the grammar of Example 4a2.

4c. Extensions to the programs

The programs described above could be extended in a number of different ways. The most interesting of these depend on the use of the various complexity measures discussed in Section 2. To the extent that we accept these measures, they provide evaluation functions for the grammatical inference device. The existing programs choose simplification rules simply and deterministically. By using a measure like $\gamma(S,G)$ for a sample set, S, of strings and a grammar G, we could allow the program to evaluate several simplifications.

A more difficult problem arises in attempting to study large samples because the number of substitutions to be considered grows exponentially with the number of variables. We suspect that the number of substitutions which are compatible with the sample, while much smaller, also grows exponentially.

The difference in γ caused by a substitution might be a good heuristic for deciding whether or not it should be carried out. This leads naturally to a tree search for the best value of γ over sequences of substitutions, and the usual search heuristics can be applied.

Thus complexity measures can be used in deciding between alternative grammars for the same sample and alternative sequences of substitutions of variables. There is another possibility which is much more important to investigate -- incremental change of grammar. The methods of this section, as well as those in [Feldman 67] deal only with a fixed sample set. If another string is added to the sample, the current programs must start again from scratch. Intuitively, one can think of heuristics for changing a grammar to accomodate the extra string. The problem is that the obvious heuristics all lead to ever more complex grammars. We might be able to use $\gamma(S,G)$ as an objective function and do hill-climbing techniques to search for grammars.

Another important class of problems involve the interaction between the informant and learner. Horning will develop the theory of this further in his dissertation. The interesting programming problems include the learner asking about the well-formedness of strings and the design of optimal teaching sequences. In this, as in its other aspects, the grammatical inference problem is the prototype of a very general situation.

BLANK PAGE

Appendix A: Representations of Finite-State Grammars

In Appendix B we compute the value of the size measures for the finite-state languages. We first need a matrix representation of the languages which aids investigations of the measures. Although one matrix representation has been used extensively in the literature (e.g. Shannon and Weaver 49, Chomsky and Miller 58, Kuich and Walk 65), the representation will be shown to be inadequate for the finite-state languages. The inadequacy of the representation has led several authors to false conclusions about the finite-state languages.

The previous matrix representation for a deterministic finite-state grammar, which we term the "old" representation, is a square matrix of the form $\mathcal{M} = [G_{ij}]$, $i,j=1,\ldots,n$. Each G_{ij} is a subset of the alphabet T, and contains those terminal symbols associated with a single-stage transition from state i to state j. The grammar has n states, one of which is the initial (starting) state (say state 1). The condition that the grammar is deterministic implies that $G_{ij} \cap G_{ij'} = \emptyset$ for $j \neq j'(i=1,\ldots,n)$.

Let $X, Y, Z \subset T^*$. Define $X + Y = X \cup Y$ and define $XY = \{\alpha\, b: \alpha \in X$ and $b \in Y\}$. Thus $X+Y = Y+X$, $X+\emptyset = \emptyset+X = X$, $(X+Y) + Z = X+(Y+Z)$, $X\,\emptyset = \emptyset\, X = \emptyset$, $X\{e\} = \{e\}\, X = X$, $(XY)Z = X(YZ)$, $(X+Y)Z = XZ+YZ$, and $X(Y+Z) = XY+YZ$

The algebraic properties of such systems has been partially investigated using semigroups, and an interesting class of abstract algebras, termed the semi-rings (which are built from two free semi-groups), has been investigated by Reder [68]. The formal properties of such algebras permit

a meaningful definition of matrices over T* in such a way that the class of all n^{th} order matrices over T* is itself a semi-ring. In particular, if $\alpha = [A_{ij}]$ and $\beta = [B_{ij}]$ are n^{th} order matrices over T*, $\alpha\beta$ can be defined as $\alpha\beta = [C_{ij}]$, where $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$ ("Σ" denotes repeated application of "+' described above). If we define $\mathcal{G}^k = \prod_{m=1}^{k} \mathcal{G} = [G_{ij}^{(k)}]$, it can be shown that $G_{ij}^{(k)} \subset T^k$; $G_{ij}^{(k)}$ is precisely the set of strings of length k associated with possible paths of k steps leading from state i to state j. In particular, $G_{1j}^{(k)}$ is the set of strings of length k leading from the initial state to state j, and $\sum_{j=1}^{n} G_{1j}^{(k)} = L_k$, where L is the language generated by the finite-state grammar associated with $\mathcal{G}$.

It is a well-known result that any language L generated by some such $\mathcal{G}$ is a finite-state (regular) language over T*. However, contrary to what seems to be commonly believed, the converse is false. There are regular languages which cannot be generated by some such matrix $\mathcal{G}$. Many of the theorems which have been proved for the class of regular languages have been demonstrated only for those languages capable of being generated by such matrices. As we shall see, serious errors have resulted from a failure to realize the limitations of this representation.

Example of a Regular Language for which the Old Representation is Inadquate

Consider the following finite-state language L over T = {a,b} : L = {$\alpha \in T^*$: α contains an even (including zero) number of a's}. A finite-state grammar for L is:

$$S \rightarrow b|Sb|Xa$$

$$X \rightarrow a|Xb|Sa$$

If we try to construct a matrix $\mathcal{M}$ which generates L, we might try:

$$\mathcal{M} = \begin{array}{c} \\ 1 \\ \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} \{b\} & \{a\} \end{array} \\ \begin{bmatrix} & \\ & \end{bmatrix} \\ \begin{array}{cc} \{a\} & \{b\} \end{array} \end{array}$$

Experimentation with the first few powers $\mathcal{M}^k$ quickly convinces one that $\mathcal{M}$ does not generate L, but rather the entire set T^*. It also becomes clear that no such matrix can, in fact, generate precisely L. L is but one of an infinite number of regular languages for which the representation is inadequate.

To see why the old representation fails we should investigate what features of a matrix permits it to selectively generate certain strings but not others. A string $\alpha \in T^k$ is generated if and only if there is some path of length k leading out of the initial state (into some other state) with which α can be associated. Starting in the initial state, $\alpha \in L_k$ sequentially determines k transitions through the states of the matrix; these transitions are determined by the sequence of terminal symbols which constitute α. If at any time there is no feasible transition possible, α is not in the language generated by that matrix.

Suppose some matrix $\mathcal{M}$ generates a language $L \subset T^*$. Consider the strings that are not in the language: $\bar{L} = T^* - L$. The preceding paragraph illustrates that $\bar{L}$ consists of those strings for which there is no feasible path of transitions within $\mathcal{M}$. Thus the only factor which can cause a string not to be in a language is that it violates some sequential rule (i.e., at some point in the string, there is no feasible transition to be made in $\mathcal{M}$); there is no capability for strings to be "rejected" on the basis of other types of "violations". Specifically, suppose the

"grammaticalness" of a string does not depend on whether there is a path for the string, but rather on where (i.e., in which state) the given path terminates. Such is the case in the "even a's" grammar; no string $\alpha \in T^k$ can violate a sequential rule since, for every string α, there is another string β: $\alpha\beta \in L$. Indeed, the "grammaticalness" of a string α depends on whether its path terminates in state "S" (i.e., even number of "a"'s) or in state "X" (odd number).

We thus see that in addition to sequential violations, a string can be ungrammatical (in terms of a finite-state grammar) if its path through the grammar matrix terminates in a "not in the language" state. Referring back to matrix $\mathcal{M}$, if we designate state "S" (i.e., State 1) as "in the language" and state "X" (i.e. State 2) as "not in the language", matrix $\mathcal{M}$ then generates the desired L; all strings $\alpha \in T^*$ have paths through $\mathcal{M}$, but only those $\alpha \in L$ will have paths terminating in "S".

A New Representation

The lack of generality of the existing matrix representation for finite-state grammars prompts us to develop a broader, fully adequate representation. Specifically, we wish to develop a matrix representation which allows regular languages (and their complements) to be defined with respect to both sequential-type rules and rules pertaining to the particular state in which a string's path terminates. At first glance, it might seem that the capacities needed to implement both sequential and terminal rules are incompatible within a single matrix representation; a sequential rule is presently implemented by selective paths in the matrix

(such that strings not in the language do not have paths in the matrix), while a terminal rule requires that all strings have paths in the matrix. Fortunately, however, these seemingly inconsistent demands can be satisfied simultaneously.

Let $\mathcal{G} = [G_{ij}]$ be an n^{th} order matrix, where each $G_{ij} \subset T$. It is assumed we are dealing with a deterministic grammar, so that $G_{ij} \cap G_{ij'} = \emptyset$, for $j \neq j'$, $i=1, \ldots, n$. By a complete finite-state grammar matrix, we mean that $\bigcup_{j=1}^{n} G_{ij} = T$, $i=1, \ldots, n$. Thus if a matrix $\mathcal{G}$ is complete, each of its rows is a partition of the alphabet T into the n cells of the row. Functionally, completeness of a grammar matrix implies that all strings $\alpha \in T^*$ have paths (derivations) in the matrix; from each state (row) of $\mathcal{G}$, each terminal symbol of T is associated with a feasible transition to another state.

With the n states of $\mathcal{G}$ we wish to associate a state classification. A state classification is a single-valued mapping of the n states into the integers $\{1,\ldots,k\}$. If $C_{\mathcal{G}}$ is a state classification of $\mathcal{G}$, then

$$C_{\mathcal{G}}: \{1,\ldots,n\} \rightarrow \{1,\ldots,k\} \quad n,k \geq 1 ,$$

is called a k-class state classification of $\mathcal{G}$.

The interpretation of $C_{\mathcal{G}}(i) = j$ is that all strings $\alpha \in T^*$ whose paths in $\mathcal{G}$ terminate in state i are classified into the j^{th} terminal class. For complete $\mathcal{G}$ we have the following:

(i) $\mathcal{G}: T^* \rightarrow \{1,\ldots,n\}$

(ii) $C_{\mathcal{G}}: \{1,\ldots,n\} \rightarrow \{1,\ldots,k\}$.

(i) means that $\mathcal{A}$ classifies all strings over T into one of n states (according to the state of $\mathcal{A}$ in which the string's path terminates). (ii) says that the state classification is k-way, that each of the states of $\mathcal{A}$ is associated with a unique terminal class. Taken together, $\mathcal{A}$ and $C_{\mathcal{A}}$ define a composite function $(\mathcal{A},C_{\mathcal{A}})$, which maps each string of T^* into a unique terminal class:

$$(\mathcal{A},C_{\mathcal{A}}): T^* \rightarrow \{1,\ldots,k\}$$

defined by $(\mathcal{A},C_{\mathcal{A}})(\alpha) = C_{\mathcal{A}}(\mathcal{A}(\alpha))$. The pair $(\mathcal{A},C_{\mathcal{A}})$ is defined as a k-class finite-state grammar over T^* (k- depends on $C_{\mathcal{A}}$).

A k-class finite-state grammar partitions the set of all strings T^* into k disjoint, exhaustive subsets. Each of these k subsets is called a terminal class of strings generated by the grammar. It can be shown that each such terminal class of strings is a regular set (finite-state language). These classes will be denoted as $L_{\mathcal{A}}^{(i)}$ (i=1,...,k) or simply by $L^{(i)}$ when the subscript $\mathcal{A}$ is understood; $L_k^{(i)}$ will denote those strings of length k in the i^{th} terminal class.

When k=2, we have a grammar generating strings into two terminal classes, which are usually thought of as the language (L) and its complement ($\overline{L} = T^*-L$). When k=1, all strings are generated into a single terminal class. The languages generated by a single class complete grammar are thus either empty or are the entire set of strings T^*.

In the "old" representation, a sequence of symbols from T failed to be grammatical when it called for a transition to be made which was not feasible; if some number of symbols brought the string's path into state i, and there was no transition out of state i associated with the next symbol

of the string, the string was ungrammatical. In our representation, all sequences of symbols must have paths through the matrix; the completeness of the matrix requires that there be transitions associated with each symbol of T, regardless of the state out of which the transition leads. We need to implement "taboo" transitions into our new matrix representations which correspond to the infeasible transitions of an "old" matrix.

Let $\mathcal{G}=[G_{ij}]$ be an n^{th} order grammar matrix of the "old" representation. Define subsets T_i of T (i=1,...,n) as $T_i = T - \bigcup_{j=1}^{n} G_{ij}$; T_i is thus the set of symbols for which there is no transition out of state i. Let $A_i \subset T^*$ (i=1,...,n); A_i is the set of all strings whose paths through $\mathcal{G}$ end in state i. Then we may describe the complement $\bar{L}$ of the language L generated by $\mathcal{G}$ as: $\bar{L} = \bigcup_{i=1}^{n} A_i T_i T^*$.

Let $\mathcal{K} = [H_{ij}]$ be an n^{th} order complete matrix which has the following properties (the existence of an $\mathcal{K}$ with these properties is self-evident): $\forall \alpha \in A_i$, $\mathcal{K}(\alpha t) = \ell$ [1/] if and only if $t \in T_i$ (i=1,...,n); $H_{\ell j} = \emptyset$ for $j \neq \ell$ and $H_{\ell\ell} = T$. Let $C_{\mathcal{K}}$ be a two-class state classification of $\mathcal{K}$ such that $C_{\mathcal{K}}(j) = \begin{cases} 1 & \text{if } j \neq \ell \\ 2 & \text{if } j = \ell \end{cases}$. Then the terminal classes $L_{\mathcal{K}}^{(1)}$ and $L_{\mathcal{K}}^{(2)}$ are precisely the sets L and $\bar{L}$, respectively; our representation has the capacity for sequential rules. State ℓ of $\mathcal{K}$ corresponds to an "absorbing" state, such that paths entering state ℓ can never leave it, regardless of the ensuing symbol sequence. All strings whose paths enter state ℓ are thus lumped together into the same terminal class. Thus if the terminal class $C_{\mathcal{K}}(\ell)$ corresponds to $\bar{L}$, and transitions into state ℓ occur only when the sequential rules implicit in the grammar are violated, we indeed implement the sequential rules into our representation of the grammar.

1/ by $\mathcal{K}(s)$ we mean that state of $\mathcal{K}$ in which the path of the string s terminates.

Example of the Implementation of a Generating [illegible]

Suppose we have a finite-state language L over the alphabet T = {a,b,c} which consists precisely of those strings in which there are no adjacent occurrences of the same symbol. Thus "abc" "abab", and "bacabacbc" are in L while "abb", "aaba", and "bacabcbc" are not. This language can be generated by a matrix of the "old" representation.

One "old" grammar matrix for L is:

$$\mathcal{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} \phi & \{a\} & \{b\} & \{c\} \\ \phi & \phi & \{b\} & \{c\} \\ \phi & \{a\} & \phi & \{c\} \\ \phi & \{a\} & \{b\} & \phi \end{bmatrix} \end{array}$$

This could be transformed into a complete matrix $\mathcal{A}'$ as follows:

$$\mathcal{A}' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \phi & \{a\} & \{b\} & \{c\} & \phi \\ \phi & \phi & \{b\} & \{c\} & \{a\} \\ \phi & \{a\} & \phi & \{c\} & \{b\} \\ \phi & \{a\} & \{b\} & \phi & \{c\} \\ \phi & \phi & \phi & \phi & \{a,b,c\} \end{bmatrix} \end{array}$$

The two-class grammar $(\mathcal{A}', C)$, where

$$C(i) = \begin{cases} 1 & \text{if } i < 5 \\ 2 & \text{if } i = 5 \end{cases}$$

has $L_{\mathcal{A}'}^{(1)} = L_{\mathcal{A}}$ and $L_{\mathcal{A}'}^{(2)} = \overline{L}_{\mathcal{A}}$.

Functional Partitions and Standard Forms of Complete Matrices

Let $\mathcal{A} = [G_{ij}]$ be a complete matrix over T. State j is said to be accessible from state i, denoted $i \to j$, if and only if there is some sequence of symbols in $T^*-\{e\}$ whose path, when starting in state i, leads to state j. States i and j are said to communicate, denoted $i \leftrightarrow j$, if and only if both $i \to j$ and $j \to i$.

The relation "$\leftrightarrow$" can be seen at once to be both transitive and symmetric on the states of $\mathcal{A}$: $i \leftrightarrow j$ and $j \leftrightarrow k \Rightarrow i \leftrightarrow k$, and $i \leftrightarrow j \Rightarrow j \leftrightarrow i$. Since, however, $i \to i$ need not hold for all states i of $\mathcal{A}$, we cannot claim "$\leftrightarrow$" to be reflexive. Thus "$\leftrightarrow$" is not an equivalence relation on all of the states of arbitrary $\mathcal{A}$.

Define E and F as the (unique) complementary subsets of the states of $\mathcal{A}$:

(a) $\forall i \in F$ and $\forall j \in E$, $j \not\to i$.

(b) $\forall i \in F$, $\exists j \in E$: $i \to j$.

(c) $\forall i \in E$, $i \leftrightarrow i$.

We have the following well-known results from the theory of finite Markov chains:

(i) $\leftrightarrow$ is an equivalence relation on the states in E.

(ii) E may be partitioned by $\leftrightarrow$ into some number f of equivalence classes (of states) $E_1, \ldots, E_f$ such that (a) $\bigcup_{k=1}^{f} E_k = E$, (b) $E_k \cap E_\ell = \emptyset$ for $k \neq \ell$, and (c) for all states i, $j \in E$, $i \leftrightarrow j \Leftrightarrow \exists k : i, j \in E_k$. Thus the E_k are equivalence classes of communicating states. We see that if the path of some string enters an E_k it can never leave that class of states. These classes are called ergodic sets of states.

An ergodic set of states E_k may consist of only one state, in which case that state once entered, can never be left; such a state is called an absorbing state (state 5 of the matrix $\mathcal{M}$ of the preceding section is an example of an absorbing state).

An ergodic set E_k is termed cyclical or periodic if there is some integer $p > 1$ which is the greatest common divisor (g.c.d.) of the lengths of all closed paths in E_k (a closed path is a sequence of transitions from a given state back into itself). The set E_k is then said to have period p. (If $p = 1$, then E_k is said to be aperiodic.) It can be shown that if p is the g.c.d. of the lengths of the closed paths of any one state in E_k, then p is the g.c.d. for all closed paths in E_k.

Now consider the set F of states which are not in E. This set has the property that once a path leaves the set, it can never return to the set. The states in F are called transient states and F is called a transient set of states. Once a path leaves the transient set, it enters some ergodic set and remains there. (State 1 of the $\mathcal{A}$ matrix of the preceding section is transient.)

It is assumed that the initial state is always State 1 of the matrix. We can make the following accessibility assumptions in complete generality:

(i) all states of $\mathcal{A}$ are accessible from the initial state.

(ii) if $\mathcal{A}$ has a transient set of states, the initial state must be in the transient set; otherwise, the initial state would be in some ergodic set and the transient states would be redundant.

(iii) if the initial state is in an ergodic set, then there is only one ergodic set of states in $\mathcal{A}$; otherwise, the additional ergodic sets (and any transient states) would be inaccessible and hence redundant. (It should be clear that any complete $\mathcal{A}$ can have at most one transient set of states and must have at least one ergodic set.)

The states of any complete matrix $\mathcal{P}$ can be rearranged (i.e., relabelled) in such a way that $\mathcal{P}$ is partitioned into one of the following standard forms:

(i)

$$\mathcal{P} = \left[\mathcal{C}_1\right]$$

Single ergodic set of states with transition matrix $\mathcal{P} = \mathcal{C}_1$

or

(ii)

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{FF} & \mathcal{P}_{FE} \\ \phi & \begin{matrix} \mathcal{C}_1 & & \\ & \mathcal{C}_2 & \\ & & \ddots \\ & & & \mathcal{C}_f \end{matrix} \end{bmatrix}$$

where the ϕ's denote regions of null transitions (empty sets), $\mathcal{P}_{FF}$ is the quadratic submatrix of transitions within the set F of transient states, $\mathcal{P}_{FE}$ is the transition matrix from F into the ergodic states, and each $\mathcal{C}_i$ is a quadratic submatrix corresponding to the i^{th} ergodic set of states E_i. Furthermore, it can be shown that each ergodic submatrix $\mathcal{C}_i$ can have its states arranged in such a way that each submatrix $\mathcal{C}_i$ (with period p_i) has the following form:

$$\mathcal{E}_i = \begin{bmatrix} \phi & \mathcal{E}_{i1} & \phi & \cdots \phi \\ \phi & \phi & \mathcal{E}_{i2} & \phi \cdots \phi \\ \cdots & \cdots & \cdots & \cdots \\ \mathcal{E}_{ip_i} & \phi & & \cdots \phi \end{bmatrix}$$

where the ϕ's are null submatrices (quadratic on the main diagonal) and the $\mathcal{E}_i$ are submatrices for transitions between the p_i cyclic subclasses of $\mathcal{E}_i$. If $\mathcal{E}_i$ is aperiodic (i.e., $p_i = 1$), then, of course, the form is degenerate. We also can make the following assumptions about a grammar $(\mathcal{G}, C_{\mathcal{G}})$ with no loss of generality:

(i) if an ergodic set E_i contains more than one state, then not all of the states in E_i are of the same terminal class; otherwise, an identical language would be obtained by lumping all the states of E_i into a single absorbing state.

(ii) there need not be more than one absorbing state for each of the terminal classes of the grammar; otherwise an identical language would be obtained by lumping together all absorbing states corresponding to a given terminal class.

The partitions of the states of $\mathcal{G}$ into various sets (ergodic and transient) is a standard form borrowed from the literature of finite Markov chains and their associated transition matrices. We will later find such partitioning useful for several reasons. We will assume that all complete grammar matrices are placed in one of these standard forms.

Appendix B: Size Measures of Regular Languages

Note: Developments in this appendix make extensive use of the matrix representation introduced in Appendix A.

Connection Matrices

Let $\mathcal{G} = [G_{ij}]$ be a complete n^{th} order matrix over T. Define the n^{th} order matrix N by $N = [n_{ij}] = [n(G_{ij})]$. n_{ij} is the number of one-step transitions from state i to state j of $\mathcal{G}$; n_{ij} is also the number of strings of length 1 associated with this transition. Consider positive integral powers N^k of N: $N^k = [n_{ij}^{(k)}]$ has the following well-known properties:

(i) $\sum_{j=1}^{n} n_{ij}^{(k)} = r^k$, $i = 1,\ldots,n$ and $k = 1,2,3,\ldots$.

(ii) $n_{ij}^{(k)} = n(G_{ij}^{(k)})$; thus $n_{ij}^{(k)}$ is the number of paths of length k from state i to state j, also the number of strings of length k associated with a transition from state i to state j. In particular, if state 1 is the initial state, $n_{1j}^{(k)}$ is the number of strings of length k whose paths terminate in state j.

N is called the connection matrix of $\mathcal{G}$ and N^k is called the k-step connection matrix of $\mathcal{G}$.

Let $(\mathcal{G}, C_{\mathcal{G}})$ be an m-class complete finite-state grammar over T.

Then $L_k^{(i)} = \bigcup_{j \in C_{\mathcal{G}}^{-1}(i)} G_{1j}^{(k)}$ for all $k \geq 1$. $(C_{\mathcal{G}}^{-1}(i) = \{j : C_{\mathcal{G}}(j) = i\})$.

Let d_i be the density of the i^{th} terminal class $L^{(i)}$, $i = 1,\ldots,m$. Provided these densities exist,

it is clear that $\sum_{i=1}^{m} d_i = 1$.

Consider

$$d_i = \lim_{k \to \infty} \frac{n(L_k^{(i)})}{r^k} = \lim_{k \to \infty} \frac{\sum_{j \in C_{\mathcal{G}}^{-1}(\cdot)} n_{1j}^{(k)}}{r^k}$$

$$= \sum_{j \in C_{\mathcal{G}}^{-1}(i)} \lim_{k \to \infty} \frac{n_{1j}^{(k)}}{r^k}$$

Thus the existence of the d_i's for arbitrary finite-state grammars can be established by the existence of $\lim_{k \to \infty} \{\frac{n_{1j}^{(k)}}{r^k}\}$ for all j (in our generalized sense of limit).

Define the matrix P, associated with $\mathcal{G}$, by $P = \frac{1}{r} N$; the elements p_{ij} of P are then related to the elements n_{ij} of N by $p_{ij} = \frac{1}{r} n_{ij}$.

Letting $P^k = [p_{ij}^{(k)}]$ we have $P^k = \frac{1}{r^k} N^k$, so that $p_{ij}^{(k)} = \frac{1}{r^k} n_{ij}^{(k)}$ for all $i,j, = 1,\ldots,n$ and for all $k \geq 1$. Hence questions about the limiting behavior of $\frac{n_{ij}^{(k)}}{r^k}$ as $k \to \infty$ can be answered in terms of the stationarity of $p_{ij}^{(k)}$ as $k \to \infty$. The stationarity of increasing powers of P is easily investigated, since P is a stochastic transition matrix and may be associated with a finite Markov chain.

We may assume that $\mathcal{G}$ is in a standard form (see Appendix A) so that its ergodic sets are readily identifiable as submatrices of $\mathcal{G}$. The relation between P and $\mathcal{G}$ is such that we can assume without loss of generality that P has the form of either

(i) a single ergodic set of n states;

or (ii) a transient set of n_o states, and f ergodic sets of states, consisting of $n_1,\ldots,n_f$ states, respectively: $n = \sum_{i=o}^{f} n_i$. The well-known theory of finite Markov chains supplies the following results:

case (i):

There is a unique matrix $P^* = [p^*_{ij}]$ such that $\lim_{k\to\infty} p_{ij}^{(k)} = p^*_{ij}$ for $i,j = 1,\ldots,n$. If P is cyclical (Appendix A) with period q , then each of the q subsequences $P^{kq+\ell}$ (as $k \to \infty$) converges to a unique limiting matrix $P^*(\ell) = [p^*_{ij(\ell)}]$ in the sense that $\lim_{k\to\infty} p_{ij}^{(kq+\ell)} = p^*_{ij(\ell)} > 0$ $(i,j = 1,\ldots,n \quad ; \quad \ell = 0,\ldots,q-1)$. Thus $p^*_{ij} = \frac{1}{q}\sum_{\ell=0}^{q-1} p^*_{ij(\ell)}$ $(i,j = 1,\ldots,n)$. Thus for aperiodic P , $\lim_{k\to\infty} P^k = P^*$.

It is well-known that $p^*_{ij} = p^*_{i',j} = p_j > 0$ for $i,i',j = 1,\ldots,n$. Thus the p_j represent the limiting proportion of strings whose paths begin in state i and terminate in state j , and is the same for $i = 1,\ldots,n$. These p_j are determined by the system of linear equations

$$\sum_{i=1}^{n} p_i p_{ij} = p_j \qquad (j = 1,\ldots,n)$$

$$\sum_{i=1}^{n} p_i = 1 \qquad .$$

Since these stationary $p_j = \lim_{k\to\infty} p_{ij}^{(k)}$ always exist, and are strictly positive, the densities d_i of the terminal classes of the grammar $(\mathcal{L},C_{\mathcal{L}})$ will therefore always be positive:

$$d_i = \sum_{j\in C_{\mathcal{L}}^{-1}(i)} p_j \qquad i = 1,\ldots,m \quad .$$

The independence of $p^*_{ij} = p_j$ of i implies that the limiting proportion of strings $\alpha\beta$ whose paths terminate in state j is always p_j, independent of α. The significance of this result will be seen shortly.

case (ii):

We first need to compute the constants $u_1,\ldots,u_f$; u_j is the limiting proportion of strings whose paths enter the i^{th} ergodic set E_i. The matrix $\mathcal{P}$ is assumed to be in the standard form (ii) (see Appendix A).

Define a_{ij} as the limiting proportion of paths leading out of state i which lead into the ergodic state j after leaving the transient set of states (i.e., state j is the first ergodic state of the path); a_{ij} is defined for $i = 1,\ldots,n_o$ and $j = n_o+1,\ldots,n$. Then the system

$$a_{ij} = p_{ij} + \sum_{k=1}^{n_o} p_{ik}a_{kj} \qquad (i = 1,\ldots,n_o \; ; \; j = n_o+1,\ldots,n)$$

$$\sum_{j=n_o+1}^{n} a_{ij} = 1 \qquad (i = 1,\ldots,n_o)$$

(refer to Appendix A for notation)

will always yield a unique solution for the a_{ij}. From the a_{ij} the u_i can be computed as $u_i = \sum_{j\in E_i} a_{1j}$ $(i = 1,\ldots,f)$; $\sum_{i=1}^{f} u_i = 1$.

The accessibility assumptions of Appendix A imply that $u_i > 0$ $(i = 1,\ldots,f)$. Any string α whose path enters an ergodic set E_i must remain there. Earlier we saw that the limiting proportion p_j of strings $\alpha\beta$ whose paths enter E_i and terminate in state j of E_i is independent of α; the p_j can be computed from the ergodic submatrix $\mathcal{E}_i$ of E_i (see case (i)). Thus the overall proportion of strings whose paths terminate

in state j of E_i is $p'_j = u_i p_j$ for $j \epsilon E_i$. Since the u_i and p_j are positive, it follows that $p'_j > 0$ for $j \epsilon E$ (and $p_j = 0$ for $j \epsilon F$).

For the i^{th} terminal class of strings $L^{(i)}$, the density d_i is is computed as

$$d_i = \sum_{j=1}^{f} u_j \sum_{k \epsilon E_i \cap C_{\mathcal{L}}^{-1}(i)} p_k = \sum_{k \epsilon E \cap C_{\mathcal{L}}^{-1}(i)} p'_k \quad .$$

The density d_i will thus be zero if and only if all states of $\mathcal{L}$ in the i^{th} terminal class are transient. We thus have

Theorem B: For an arbitrary m-class complete finite-state grammar $(\mathcal{L}, C_{\mathcal{L}})$ over a finite alphabet, the densities $d_1, \ldots, d_m$ always exist; each d_i is positive unless all states in $C_{\mathcal{L}}^{-1}(i)$ are transient, in which case d_i is zero.

When $m = 2$ (i.e., $L^{(1)}$ is the language generated by the grammar and $L^{(2)}$ is its complement), we see that the density of the language always exists and is zero if and only if $C_{\mathcal{L}}^{-1}(1) \subset F$.

Randomly Generated Strings

Chomsky and Miller [58] considered randomly generated strings of length k; such a string is one drawn from the "urn" T^k such that all strings of length k have equal (i.e., r^{-k}) probability of being drawn. Chomsky and Miller claimed that as $k \to \infty$, the limiting probability of a randomly generated string being in any given regular language is always zero or unity. This claim is equivalent to claiming that the density of any regular language is either zero or unity, which has been shown to be false. Two simple counter examples (each with density $1/2$) of non-zero, non-unity

density regular languages are

(i) $S \rightarrow b \mid Sb \mid Xa$, $X \rightarrow a \mid Xb \mid Sa$ } even "a"'s grammar

(ii) $S \rightarrow a \mid Sa \mid Sb$ } grammar for all strings over {a,b} which begin with "a".

For a discussion of second-order (logarithmic density) size measure of a regular language, the reader is referred to Shannon and Weaver [49]. They compute the value of channel (coding) capacity C, which we showed to be proportional to our second-order size measure (Section 2a.0).

Appendix C: Sample Computer Runs

The following examples were run on the Stanford PDP-10 using LISP. The program deals with two sets of strings, the sample set and the set of pivots (cf. Section 4a). The functions GRIN1A , GRIN2A (of zero arguments) apply the algorithms described in Section 4 to the current sets. The functions GRIN1 , GRIN2 accept the sample set to be used as an argument. The function GRINA simply calls both GRIN1A and GRIN2A in succession; GPIN calls GRIN1 and GRIN2 in succession. The auxiliary function PIVOTS specifies the current set of pivot symbols and ADDS causes new strings to be added to the sample set. The symbols G0009 , G0010 , etc. are internally created (by GENSYM) names within LISP; these correspond to the non-terminal symbols Z_1,Z_2 used in the text.

```
(GRIN (A)(A A)(A A A))
(THE FINITE STATE GRAMMAR GENERATED BY GRIN1 IS)
(G0009 IS THE DISTINGUISHED NONTERMINAL)
(G0009 ← A G0009 /  A)

(THE PIVOT GRAMMAR GENERATED BY GRIN2 IS)
(G0012 IS THE DISTINGUISHED NONTERMINAL)
(G0012 ← A G0012 /  A)
NIL
```

```
(GRIN (A)(A B)(A A)(A A B)(A B B)(A A A))
(THE FINITE STATE GRAMMAR GENERATED BY GRIN1 IS)
(G0014 IS THE DISTINGUISHED NONTERMINAL)
(G0014 ← A /  A G0015)
(G0015 ← A /  B /  A A /  B B /  A B)

(THE PIVOT GRAMMAR GENERATED BY GRIN2 IS)
(G0017 IS THE DISTINGUISHED NONTERMINAL)
(G0017 ← A G0017 /  G0017 B /  A)
NIL

(GRIN (B B)(B A B)(B A A B)(B A A A B))
(THE FINITE STATE GRAMMAR GENERATED BY GRIN1 IS)
(G0019 IS THE DISTINGUISHED NONTERMINAL)
(G0019 ← B G0020)
(G0020 ← A G0020 /  B)

(THE PIVOT GRAMMAR GENERATED BY GRIN2 IS)
(G0024 IS THE DISTINGUISHED NONTERMINAL)
(G0024 ← B G0025)
(G0025 ← A G0025 /  B)
NIL

(GRIN (C B)(B B B)(C A B)(B P A B)(C A A B)(B B A A B)(C A A A B))
(THE FINITE STATE GRAMMAR GENERATED BY GRIN1 IS)
(G0027 IS THE DISTINGUISHED NONTERMINAL)
(G0027 ← C G0029 /  B G0028)
(G0028 ← B G0029)
(G0029 ← A G0029 /  B)

(THE PIVOT GRAMMAR GENERATED BY GRIN2 IS)
(G0035 IS THE DISTINGUISHED NONTERMINAL)
(G0035 ← G0036 B)
(G0036 ← G0036 A /  B G0037 /  C)
(G0037 ← B)
NIL
```

```
(GRIN (A A B B)(A B)(A A A B B B))
(THE FINITE STATE GRAMMAR GENERATED BY GRIN1 IS)
(GØØØ9 IS THE DISTINGUISHED NONTERMINAL)
(GØØØ9 ← A GØØ1Ø)
(GØØ1Ø ← B /  A GØØ11)
(GØØ11 ← B GØØ14 /  A GØØ12)
(GØØ12 ← B GØØ11)
(GØØ14 ← B)

(THE PIVOT GRAMMAR GENERATED BY GRIN2 IS)
(GØØ16 IS THE DISTINGUISHED NONTERMINAL)
(GØØ16 ← A GØØ17)
(GØØ17 ← GØØ16 B /  B)
NIL

(ADDS (C)(A C B)(A A C B B)(A A A C B B B))
((A A A C B B B) (A A A B B B) (A A C B B) (A A B B) (A C B) (A B)
(C))

(GRIN2A)
(GØØ19 IS THE DISTINGUISHED NONTERMINAL)
(GØØ19 ← A GØØ2Ø /  C)
(GØØ2Ø ← GØØ19 B /  B)
NIL

(PIVOTS M P)
(M P)

(GRIN2 (A M A)(A)(A M A M A)(A M A M A M A))
(GØØ22 IS THE DISTINGUISHED NONTERMINAL)
(GØØ22 ← A /  GØØ22 M GØØ22)
NIL
```

```
(ADDS (A P A)(A P A P A)(A M A P A)(A P A M A))
((A M A M A M A) (A P A P A) (A M A P A) (A P A M A) (A M A M A) (A
 P A) (A M A) (A))

(GRIN2A)
(GØØ24 IS THE DISTINGUISHED NONTERMINAL)
(GØØ24 ← A /  GØØ24 P GØØ24 /  GØØ24 M GØØ24)
NIL

(GRIN2 (B) (A M  B)(A M A M B)(A M A M A M B))
(GØØ26 IS THE DISTINGUISHED NONTERMINAL)
(GØØ26 ← B /  GØØ27 M GØØ26)
(GØØ27 ← A)
NIL

(PIVOT M P)
(M P)

(GRIN2 (A M A)(L A M A R M A)(A M L A M A R)(L A M A R M L A M A R)
  (L A M L A M A R R M A)(L L A M A R M A R M A)(A M L A M L A M A R R)
  (A M L L A M A R M A R))
(GØØØ9 IS THE DISTINGUISHED NONTERMINAL)
(GØØØ9 ← GØØ1Ø M GØØ1Ø)
(GØØ1Ø ← L GØØ12 /  A)
(GØØ12 ← GØØØ9 R)
NIL

(GRIN2 (C D)(A B D)(A C B D)(A A B B D)(A A C B B D)(A A A B B B D)
  (A A A C B B B D))
(GØØ15 IS THE DISTINGUISHED NONTERMINAL)
(GØØ15 ← GØØ16 D)
(GØØ16 ← A GØØ17 /  C)
(GØØ17 ← GØØ16 B /  B)
NIL
```

References

[1] Aizerman, M. et. al., "Theoretical Foundations of the potential function method in pattern recognition", Automation and Remote Control 25, 821-837 and 1175-1190 (1964).

[2] Amarel, S., "On the Automatic Formation of a Computer Program that Represents a Theory", in Self Organizing Systems, Yovits, Jacobi, and Goldstein, eds., Washington, Spartan, 1962.

[3] Chomsky, N., and G. Miller, "Formal Analysis of Natural Languages", p. 269-493 in Handbook of Math. Psych. II, Luce, Bush, Galantier, eds., New York, Wiley, 1963.

[4] _____, "Some Finitary Models of Language Users", in Luce et.al. (above), 1963b.

[5] _____, "Finite-State Languages" Information and Control 1, 91-112 (1958).

[6] _____, Pattern Conception, Report No. AFCRC-TN-57-57, August 7, 1957.

[7] Church, A., Introduction to Mathematical Logic, Princeton University Press, Princeton, New Jersey, 1956.

[8] Feldman, J. A., "First thoughts on grammatical inference", Stanford A.I. Memo No. 55, August, 1967.

[9] Ginsburg, S., The Mathematical Theory of Context-free Languages, McGraw Hill, New York, 1966.

[10] Gold, M., "Language Identification in the Limit", Information and Control, 10, 447-474 (1967).

[11] _____, "Limiting Recursion", J. Symb. Logic 30, 28-48 (1965).

[12] Gorn, S., "Specification Languages for Mechanical Languages", Commun. ACM 4, Dec. 1961, p. 532-542.

[13] Greibach, S., "A New Normal-form Theorem for Context-free Phrase-Structure Grammars", J. ACM 12, 1 Jan. 65, p. 42-55.

[14] Harrison, M., Introduction to Switching and Automata Theory, New York, McGraw Hill, 1965.

[15] Hartmanis, J., "Computational Complexity of one-tape Turing Machine Computations", J. ACM 15 (April 1968), pp. 325-339.

[16] Hunt, E., Marin, P. Stone, Experiments in Induction, New York, Academic Press, 19 . .

[17] Kemeny, J., J. L. Snell, and A. Knapp, Denumerable Markov Chains, D. Van Nostrand Co., Princeton, N. J., 1966.

[18] Knopp, K., Theory and Application of Infinite Series. Hafner, New York 1948.

[19] Kuich, W. and K. Walk, Block Stochastic Matrices and Associated Finite-State Languages, IBM TR 25.052 (July 1965).

[20] Lederberg, J. and E. Feigenbaum, "Mechanization of Inductive Inference in Organic Chemistry", in Formal Representation of Human Judgement, Kleinmutz, ed., John Wiley, New York, 1968.

[21] London, R., "A Computer Program for Discovering and Proving Sequential Recognition Rules for BNF Grammars", Carnegie Tech., May 1964.

[22] Luce, R. D., "Selective Information Theory", in Developments in Mathematical Psychology, Luce (ed.), The Free Press, Glencoe, Illinois (1960).

[23] Miller, G., and M. Stein, 'Grammarama Memos", Unpublished Internal Memos, Harvard Center for Cognitive Studies, Dec. 1963 and August 1966.

[24] Miller, W., and A. Shaw, "Linguistic Methods in Picture Processing, a Survey", Proc. AFIPS FJCC, 1968, p. 279-291.

[25] Moore, E., "Gedanken Experiments on Sequential Machines", in Automata Studies, Shannon and McCarthy, Princeton, 1956.

[26] Newell, A., "Heuristic Search: Ill-Structured Problems", Progress in O.R. (Vol. 3).

[27] Persson, S., "Some Sequence Extrapolating Programs", Stanford A.I. Memo No. 46, September 1966.

[28] Reuer, S., "Introduction to Semi-rings", Unpublished research, Dept. of Mathematics, Stanford University, 1968.

[29] Reynolds, J., "Grammatical Covering", TM-4 , Argonne National Lab., June 1[illegible].

[30] Shamir, E., "A Remark on Discovery Algorithms for Grammars", Information and Control [illegible], 246-251 (1972).

[31] Shannon, C. and W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana (1949).

[32] Solomonoff, R., "Some Recent Work in Artificial Intelligence", Proc. IEEE V. 54, No. 12, December 1966.

[33] _____. "A Formal Theory of Inductive Inference", Information and Control, 1964, pp. 1-22, 224-254.

[34] _____. "A New Method for Discovering the Grammars of Phrase Structure Languages", Information Processing, June 1959, pp. 285-290.

[35] Suppes, P., "Concept Formation and Bayesian Decisions", in Aspects of Inductive Logic, Hintikka and Suppes, eds., 1956, Amsterdam, North Holland.

[36] Uhr, L., ed., Pattern Recognition, Wiley, New York, 1966.