THE INFORMATION RATE AND OTHER PARAMETERS OF PROBABILISTIC CONTEXT FREE GRAMMARS AND THEIR PARSERS

by

Simon Kestner

Thesis submitted for the Degree of Doctor of Philosophy in Computer Science at the University of Warwick, 1974

ABSTRACT

Probabilistic context-free languages are defined by giving predetermined probabilities (preprobabilities) for the choices that their grammars make when generating.

Chapter 1 shows how to carry out the above definition, and how to calculate some parameters of the language; for instance: average length of work, mean square length, digraph probabilities, entropy.

Chapter 2 introduces generating functions related to grammars. It uses them to derive a condition for which preprobabilities give rise to well-formed probability spaces. Two functions, the length and entropy generating functions are studied in detail. They are algebraic functions, can in general only be defined implicitly, but can be used to give unified explicit methods of calculating all the parameters of chapter 1 (and more).

Chapter 3 defines and shows how to calculate the information rate of a language. As a by-blow, Macmillan's theorem is extended (for a small class of processes) to an analogue of the Central Limit Theorem.

Chapter 4 tries to compare the efficiencies of different parsing algorithms. In a reasonable sense, all deterministic parsers take equal average time to parse, any backtracking parser is slower, but there is no general algorithm for calculating the speed of a backtracking parser.

Acknowledgements

My thanks go to my parents and friends for their encouragement and toleration while I was writing this thesis. To Professor W. Parry and Dr. G. Segal for assistance with a couple of recalcitrant theorems. To Dr. David Park, my supervisor, for his continuing patience and support. And to Mrs. P. Broadbent for her calmness and speed in typing and producing the final form.

CONTENTS

Chapter 1

PROBABILITY SPACES AND GRAMMARS

This chapter shows firstly how to make the set of parses which can be generated by a grammar into a probability space, and secondly how to integrate various functions from the parses if it is assumed that the measure of the set of infinite parses is zero. For instance one such function gives the number of terminals in a parse, and so it becomes possible to calculate the average length of string generated by a grammar. Similarly the variance of the length, the probability that one letter is followed by another, the entropy, and so on can be calculated.

Throughout this chapter a fixed grammar $G = (N,T,P,S)$ will be dealt with where

$$(N,T,P,S) = (\{X_i : i=1,\ldots,n\}, T, \{P_{ij} : i=1,\ldots n; j=1\ldots,n_i\}, X_k)$$

and the length of the right hand side of P_{ij} is n_{ij}. G will be reduced[11,15], that is every non-terminal can generate a terminal string and can also be generated in a string generated by the root, and ϵ-free that is no production has a right hand side of zero length.

1.1 CONSTRUCTION OF A σ-FIELD OVER THE SET OF PARSES

Eventually a measure is desired which assigns a probability to all individual finite parses, and so each singleton set containing one finite parse should be measurable. The most obvious σ-field with this property is the set of all sets of parses. Unfortunately, because there are (in general) uncountably many infinite parses, the only measure functions which can be defined on this σ-field give

zero measure to some set containing all but a countable number of the infinite parses. This is not a desired property, and so some other σ-field must be constructed. So properties of parses will be investigated in order to help with the construction.

1.1.1 Definition

The depth of a node in a parse is defined by induction to be 0 if that node is the root, otherwise one greater than the depth of its immediate ancestor.

Remark: Every node, even of an infinite parse, has finite depth, which is equal to the number of arrows which must be traversed when tracing down to it from the root. Parses can be classified by the depths of their nodes.

1.1.2 Definition

A partial parse is of exact depth d if:

1. Every node is of depth d or less.
2. Every tip whose value is a non-terminal is of depth d.

(Such a node will be called a non-terminal tip.)

1.1.3 Remark

Thus a partial parse of depth d may be thought of as obtained from a full parse by deleting all nodes of depth greater than d. As this can clearly only be done in exactly one way, every parse has just one partial parse of exact depth d which is a part of it. It should also be noted that if the maximum of the depths of the nodes of a full parse is $m \leqslant d$, then that parse satisfies the definition for a partial parse of depth d. Hence such a parse is of exact depth d for all $d \geqslant m$.

1.1.4 Lemma

For any d there are only finitely many partial parses of exact depth d.

Proof: By induction on d.

1.1.5 Remark

The set of parses of exact depth d can now be used to generate a partition $D_d = \{D_d^q\}$ of the set of all full parses Ω, and hence a σ-field $\mathcal{A}_d$ consisting of all unions of elements of D_d.

1.1.6 Definition

D_d^q is the set of all parses which are extensions of the partial parse q of exact depth d.

1.1.7 Lemma

i. $\bigcup_q D_d^q = \Omega$

ii. If $q \neq q'$ then $D_d^q \cap D_d^{q'} = \emptyset$.

Proof: Every parse can be restricted to exactly one partial parse of depth d, and so is in just one set D_d^q.

By the above two lemmas $\{D_d^q\}$ is a finite partition of Ω, and so the set $\mathcal{A}_d$ of all unions of elements like D_d^q is a finite field and so a σ-field.

1.1.8 Lemma

For all d, $\mathcal{A}_{d+1} \supseteq \mathcal{A}_d$.

Proof: Let $D_{d+1}^{q'}$ be an element of the partition D_{d+1}, and q' the restriction of q to nodes of depth d or less. Any extension of q is an extension of q', hence $D_{d+1}^q \subseteq D_d^{q'}$, therefore D_{d+1} is a refinement of D_d, and so $\mathcal{A}_{d+1} \supseteq \mathcal{A}_d$.

1.1.9 Definition

It has now been proved that the sequence of σ-fields $\mathcal{A}_d$ has all the properties which allow a limit to be formed. Let this limit field be $\mathcal{A}$. Then $\mathcal{A}$ has the following properties.

1.1.10 Properties

1. If q is a finite parse then $\{q\} \in \mathcal{A}$.
2. If I is the set of all infinite parses then $I \in \mathcal{A}$.
3. (i) Any function constant on I is measurable.

   (ii) If $\{\Omega, \mathcal{A}\}$ is completed to a measure space $\{\Omega, \mathcal{A}, \mu\}$ and $\mu(I) = 0$ then every function is integrable.

Proofs:

1. As q is finite there is a finite maximum m of depths of nodes of q, and so $\{q\} \in \mathcal{A}_d$ for all $d \geq m$.
2. The complement of I is the set of all finite parses which is the countable union of the singleton sets $\{q\}$, where q is a finite parse.
3. Obvious consequences of 1.

1.2 CONSTRUCTION OF SOME PROBABILITY FUNCTIONS ON THE PARSES

In this paragraph a way of constructing a probability function on the parses will be described informally, in the next formally. A parse can be built step by step from the root. Each step starts with a partial parse and ends up with a new one obtained by expanding just one non-terminal tip. When there are no non-terminal tips left then the procedure stops. Once it has been decided which non-terminal tip to expand, there are in general several productions which will fit. If the production to use is chosen by a random choice and the probabilities that each will be chosen given that it

will fit are specified, then it is possible to calculate the probability that a particular parse will be chosen. It is the product of the probabilities of all the productions which were used to generate it. In the formal definition which follows the preprobability function gives the probability that a particular production will be chosen.

To recall notation, the grammar is still

$$(N,T,P,S) = (\{X_i : i=1,\ldots,n\} \ , \ T, \ \{P_{ij} : i=1,\ldots n; j=1,\ldots n_i\} \ , X_k)$$

where the length of P_{ij} is n_{ij}.

1.2.1 Definition

A preprobability is a function $f : P \rightarrow \{x : x \text{ is real}, \ 0 \leq x \leq 1\}$ such that

$$\sum_{j=1}^{n_i} f(P_{ij}) = 1 \quad \text{for all } i.$$

1.2.2 Remarks

Notice that except for grammars which produce only one parse, all grammars have an uncountably infinite number of preprobability functions.

The preprobability function f can be used to generate a sequence of measures μ_d on the spaces $(\Omega, \mathcal{A}_d)$. The following notation will be used in the construction. If q is any partial parse, in particular of exact depth d, then $t_{ij}(q)$, or more usually just t_{ij} when q can be understood, is the number of occurrences of production P_{ij} used to generate q, and $\mathcal{P}(q)$ is the probability that q be generated, that is

$$\mathcal{P}(q) = \prod_{i=1}^{n} \prod_{j=1}^{n} [f(P_{ij})]^{t_{ij}(q)}$$

1.2.3 Definition

μ_d is a measure to convert the measurable space $(\Omega, \mathcal{A}_d)$ to a measure space $(\Omega, \mathcal{A}_d, \mu_d)$ with the property that

$$\mu_d(D_d^q) = \prod_{i=1}^{n} \prod_{j=1}^{n_i} [f(P_{ij})]^{t_{ij}} = \mathcal{P}(q).$$

1.2.4 Lemma

μ_d exists and is unique.

Proof: The domain of μ_d is the finite set $\mathcal{A}_d$. If $A \in \mathcal{A}_d$ then either $A = D_d^q$ for some parse q, in which case $\mu_d(A)$ is given directly by the property, or else $A = \bigcup_{q \in Q} D_d^q$ for some finite set Q of partial parses, in which case

$$\mu_d(A) = \sum_{q \in Q} \mu_d(D_d^q).$$

1.2.5 Theorem

The sequence of measures μ_d is compatible.

Proof: It is only necessary to show that if A is a generator of $(\Omega, \mathcal{A}_d, \mu_d)$, that is $A = D_d^q$ for some q, then $\mu_{d+1}(A) = \mu_d(A)$. In $(\Omega, \mathcal{A}_{d+1}, \mu_{d+1})$, A is the union $A = \bigcup_{q'} D_{d+1}^{q'}$, where q' varies over all possible partial parses obtained by simultaneously expanding each of the non-terminal tips of q by one production. So it must be shown that $\mathcal{P}(q) = \sum_{q'} \mathcal{P}(q')$. This can easily be done by setting up appropriate notation to denote parses like q but with only some tips expanded, and using an induction.

1.2.6 Corollaries

1. μ_d is a probability measure for all d. Proof: $\mu_0(\Omega) = 1$.
2. The limit μ exists and is a probability measure.

1.3 DECOMPOSITION THEOREMS

Next two theorems will be proved which enable the probability space $(\Omega, \mathcal{A}, \mu)$ to be decomposed. The first will show that it is isomorphic to a weighted sum of spaces, each of which corresponds to one of the productions which is a first possible choice in generating a parse. Then the second will show how each of the new spaces can be decomposed as a product.

To help describe the decomposition as a weighted sum, some new notation will be needed. This will allow a grammar G to be changed to a grammar G^a which has almost the same set of parses, the only difference being that where a parse of G has S at the root, the corresponding parse of G^a has a new non-terminal Z. This will enable an occurrence of S at the root to be easily distinguished from any other occurrence.

1.3.1 Definition

If G is the grammar (N,T,P,S), then G^a is the grammar $(N \cup \{Z\}, T, P \cup Q, Z)$ where Z is a new non-terminal in neither N nor T, and Q is a set of new productions which correspond with those of G with S on the left hand side, so that $(Z \rightarrow \alpha) \in Q$ iff $(S \rightarrow \alpha) \in P$.

1.3.2 Definition

If f is a preprobability on G then the corresponding preprobability f^a on G^a will be such that if $q \in P$ then $f^a(q) = f(q)$ and $f^a(Z \rightarrow \alpha) = f(S \rightarrow \alpha)$ otherwise.

1.3.3 Lemma

The space $(\Omega, \mathcal{A}, \mu)$ generated by G, f is isomorphic to the space $(\Omega^a, \mathcal{A}^a, \mu^a)$ generated by G^a, f^a.

Proof: Let $H: \Omega \rightarrow \Omega^a$ be the bijection which maps a parse of Ω to

the parse of Ω which is similar in all ways except in having Z instead of S at the root. Let $(\Omega^a, \mathcal{A}^a_d, \mu^a_d), (\Omega, \mathcal{A}_d, \mu_d)$ be the spaces generated by the partial parses of depth d. Then it is easy to see that $(\Omega^a, \mathcal{A}^a_d, \mu^a_d)$ is isomorphic to $(\Omega, \mathcal{A}_d, \mu_d)$ under H, and hence that the limits $(\Omega^a, \mathcal{A}^a, \mu^a)$ and $(\Omega, \mathcal{A}, \mu)$ are also isomorphic.

1.3.4 Definition

Let the new productions in G^a of the form $(Z \to \alpha)$ be numbered, so that $Q = \{(Z \to \alpha_i): i=1,\ldots,m\}$. Then G_i^a is the grammar $(Nu\{Z\}, T, Pu\{(Z \to \alpha_i)\}, Z)$ and f_i^a is the preprobability $f_i^a(Z \to \alpha_i) = 1$ else $f_i^a(q) = f(q)$.

Clearly each of the grammars G_i^a has just one of the new productions $(Z \to \alpha_i)$, and this production can and must be used just once at the root of every parse in Ω_i^a.

1.3.5 Lemma

The space $(\Omega^a, \mathcal{A}^a, \mu^a)$ is isomorphic to the weighted sum

$$\left(\sum_{i=1}^m \Omega^a{}_i, \sum_{i=1}^m \mathcal{A}^a{}_i, \sum_{i=1}^m p_i \mu^a{}_i\right)$$

where $p_i = f^a(Z \to \alpha_i)$.

Proof: The isomorphism holds between the spaces generated by partial parses of depth d, and so also between the limits.

1.3.6 Theorem

The space $(\Omega, \mathcal{A}, \mu)$ generated by a grammar G and preprobability f is isomorphic to the weighted sum of spaces $\left(\sum_{i=1}^m \Omega^a{}_i, \sum_{i=1}^m \mathcal{A}^a{}_i, \sum_{i=1}^m p_i \mu^a{}_i\right)$ generated by the grammars $G^a{}_i$ and preprobabilities $f^a{}_i$.

Proof: Both isomorphic to $(\Omega^a, \mathcal{A}^a, \mu^a)$.

1.3.7 Notation

Next will be shown that each of the new spaces $(\Omega^a_i, \mathcal{A}^a_i, \mu^a_i)$ can be further decomposed into a finite product. To reduce the complication of the notation, H will stand for G^a_i and $(\Theta, \mathcal{B}, \nu)$ for $(\Omega^a_i, \mathcal{A}^a_i, \mu^a_i)$. So H is $(Nu\{Z\}, T, Pu\{(Z \to \alpha_i)\}, Z)$. α_i will be $X_1 X_2 \ldots X_{n_i}$, where each X_j may be non-terminal or not.

If Y is any non-terminal of $Nu\{Z\}$ then $H(Y) = (Nu\{Z\}, T, Pu\{(Z \to \alpha_i)\}, Y)$ will be the grammar which is the same as H except that the start symbol is changed to Y. (Notice that as Z appears on the right side of no productions, the set of parses generated by H(Y) is exactly the same as the set generated by $G(Y) = (N, T, P, Y)$ whenever $Z \neq Y$. In particular, H(Y) is not in general reduced.) If f is a pre-probability for H then it is also one for H(Y) for both have the same productions. $(\Theta(Y), \mathcal{B}(Y), \nu(Y))$ will be the space generated by H(Y) and f.

If Y is a terminal, then $(\Theta(Y), \mathcal{B}(Y), \nu(Y))$ is the trivial space; with Y the sole element of $\Theta(Y)$, $\mathcal{B}(Y)$ the two element power set of $\Theta(Y)$ and $\nu(Y)$ the trivial measure.

1.3.8 Theorem

The space $(\Theta, \mathcal{B}, \nu)$ is isomorphic to the product

$$\left(\prod_{j=1}^{n_i} \Theta(X_j), \prod_{j=1}^{n_i} \mathcal{B}(X_j), \prod_{j=1}^{n_i} \nu(X_j) \right)$$

Proof: Firstly, the set Θ is isomorphic to the set $\prod_{j=1}^{n_i} \Theta(X_j)$. For let θ be a parse such that $\theta \in \Theta$. As the start symbol of H is Z, and the only production in H involving Z is $(Z \to X_1 \ldots X_{n_i})$, θ must have Z at its root and $X_1, \ldots, X_{n_i}$ in order as the direct descendants of Z. If θ_j is the parse of $H(X_j)$ which is the same as θ restricted to X_j and its descendants, then it is clear that

the function $I:\Theta \to \prod_{j=1}^{n_i} \Theta(X_j)$ with the property $I(\Theta) = (\Theta_1,\ldots,\Theta_{n_i})$ has a well-defined inverse and so is an isomorphism. It is also clear that if $(\Theta, \mathcal{B}_{d+1}, \mathcal{P}_{d+1})$ is the space defined by partitions of depth d+1, similarly $(\Theta(X_j), \mathcal{B}(X_j)_d, \mathcal{P}(X_j)_d)$ that generated by partial parses of depth d, then $(\Theta, \mathcal{B}_{d+1}, \mathcal{P}_{d+1})$ is isomorphic to

$$(\prod_{j=1}^{n_i} \Theta(X_j), \prod_{j=1}^{n_i} \mathcal{B}(X_j)_d, \prod_{j=1}^{n_i} \mathcal{P}(X_j)_d) .$$

Therefore the two limits of these sequences are isomorphic.

1.4 BACKUS NAUR FORM AND THE SPACE ISOMORPHISM

1.4.1 Notation

To consolidate quickly what has been done, yet more notation is needed. In the sums of products of spaces the Σ's and Π's can be taken outside the sequence brackets, thus for instance

$$\sum_{i=1}^{n} p_i(\Omega_i, \mathcal{A}_i, \mu_i) \text{ means } (\sum_{i=1}^{n}\Omega_i, \sum_{i=1}^{n}\mathcal{A}_i, \sum_{i=1}^{n} p_i\mu_i).$$

G is still $(\{X_i,i=1,\ldots,n\},T,\{P_{ij}:i=1,\ldots,n;j=1,\ldots,n_i\},X_k)$ where the length of P_{ij} is n_{ij}, so P_{ij} may be written $(X_i \to X_{ij1} X_{ij2}\ldots X_{ijn_{ij}})$ where X_i stands for a non-terminal, but X_{ijk} (which should perhaps more strictly be written $X_{h(i,j,k)}$ may also stand for a terminal. $G(X_i)$ stands for the grammar with X_i instead of X_k as start symbol (so that $G(X_k)$ is in fact G). $\mathcal{S}_i$ stands for the space $(\Omega_i, \mathcal{A}_i, \mu_i)$ generated by the grammar $G(X_i)$ and the preprobability function f, $\mathcal{S}_{ijk}$ stands for the space generated by $G(X_{ijk})$ and f if X_{ijk} is a non-terminal, and the trivial space where Ω_{ijk} contains just one element otherwise. $\mathcal{S}_{ij}$ stands for the space generated by the grammar $G(X_i)$ with the restriction that the production P_{ij} is always used first.

1.4.2 Remark

A variant of Backus Naur Form (BNF) will be described which allows the preprobabilities of the productions to be displayed.

1.4.3 Definition

If a grammar has the productions $(X_i \to \alpha_{ij})$ and the preprobability function $f(X_i \to \alpha_{ij}) = p_{ij}$ then its BNF is the set of n formulas

$$X_i ::= \{p_{i1}\}\alpha_{i1} \mid \{p_{i2}\}\alpha_{i2} \mid \dots \mid \{p_{in_i}\}\alpha_{in_i} .$$

1.4.4 Notation

For the sake of manipulation it should be noted that α_{ij} is a sequence of signs $X_{ij1}\, X_{ij2} \dots X_{ijn_{ij}}$ and so may be written $\prod_{k=1}^{n_{ij}} X_{ijk}$. Similarly the sequence of alternatives $\{p_{ij}\}\alpha_{ij}$ may be written $\sum_{j=1}^{n_i} p_{ij}\alpha_{ij}$, and so the BNF of a grammar may be expressed by the formulas

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} .$$

The above notation is rather powerful and enables the above two theorems to be summed up in the very striking form.

1.4.5 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} \qquad (i=1,\dots,n)$$

then the spaces generated satisfy the recursive equations

$$\mathcal{S}_i \cong \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} \mathcal{S}_{ijk} \qquad (i=1,\dots,n)$$

(where $\cong$ means is isomorphic to).

Proof: Theorem 1.3.6 showed

that $\mathcal{S}_i \cong \sum_{j=1}^{n_i} p_{ij} \mathcal{S}_{ij}$ and theorem 1.3.8 that $\mathcal{S}_{ij} \cong \prod_{k=1}^{n_{ij}} \mathcal{S}_{ijk}$.

1.4.6 Example

A particular example is a simple version of the language of assignment statements. G = (N,T,P,S) where

$$T = \{(,),a,b,+,x,=\}$$

$$N = \{S,L,R\}$$

and $P = \{ \langle S \to L=R \rangle, \langle L \to a \rangle, \langle L \to b \rangle, \langle R \to (R+R) \rangle, \langle R \to (RxR) \rangle,$
$\langle R \to a \rangle, \langle R \to b \rangle \}$

If the preprobabilities are in order p_{11}, p_{21}, p_{22}, p_{31}, p_{32}, p_{33}, p_{34} where of course $p_{11} = p_{21} + p_{22} = p_{31}+p_{32}+p_{33}+p_{34}=1$, then the grammar may be written in BNF as

$$S ::= \{p_{11}\} L=R$$
$$L ::= \{p_{21}\} a \mid \{p_{22}\} b$$
$$R ::= \{p_{31}\} (R+R) \mid \{p_{32}\} (RxR) \mid \{p_{33}\} a \mid \{p_{34}\} b$$

and the isomorphism between spaces may be written

$$\mathcal{S}_1 \cong p_{11} \mathcal{S}_2 \mathcal{S}_0 \mathcal{S}_3$$
$$\mathcal{S}_2 \cong p_{21} \mathcal{S}_0 + p_{22} \mathcal{S}_0$$
$$\mathcal{S}_3 \cong p_{31} \mathcal{S}_0 \mathcal{S}_3 \mathcal{S}_0 \mathcal{S}_3 \mathcal{S}_0 + p_{32} \mathcal{S}_0 \mathcal{S}_3 \mathcal{S}_0 \mathcal{S}_3 \mathcal{S}_0 + p_{33} \mathcal{S}_0 + p_{34} \mathcal{S}_0$$

where $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ are the spaces generated by G(S), G(L) and G(R) respectively, and $\mathcal{S}_0$ is the simple space with just one element in the underlying set.

1.5 RECURSIVE EQUATIONS OBTAINED BY INTEGRATION

Suppose that $g_i : \Omega_i \to \mathcal{R}$ for i=1,...,n (where $\mathcal{R}$ is the set of real numbers) is a family of random variables, and that each function g_i can be expressed in terms of the functions g_{ijk} defined on the

component sets Ω_{ijk} of Ω_i. Then sometimes the integrals $\int g_i d\mu_i$ can be expressed in terms of the component integrals $\int g_{ijk}\, d\mu_{ijk}$ to give some recursive equations satisfied by the integral. And again sometimes the numerical value of the integral can be obtained merely by studying the recursive equations. Some examples of this technique follow.

Firstly an equation will be found for the measure of the set of finite parses. v_i will be the function $v_i:\Omega_i \to \{0,1\}$ which has the value 1 on finite parses and 0 on infinite. Similarly $v_{ij}:\Omega_{ij} \to \{0,1\}$ and $v_{ijk}:\Omega_{ijk} \to \{0,1\}$. V_i will be the real number which is the integral of v_i that is

$$V_i = \int_{\Omega_i} v_i d\mu_i \qquad \text{(Similarly } V_{ij} \text{ and } V_{ijk}.\text{)}$$

1.5.1 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} \qquad (i=1,\ldots,n)$$

then the measures V_i of the sets of finite parses satisfy the equations

$$V_i = \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} V_{ijk} \qquad (i=1,\ldots,n)$$

Proof: The Ω_{ij} partition Ω_i

hence $V_i = \int_{\Omega_i} v_i \, d\mu_i = \sum_{j=1}^{n_i} \int_{\Omega_{ij}} v_i \, d\mu_i$

But if $\omega \in \Omega_{ij}$, and ω is finite, then $v_i(\omega) = v_{ij}(\omega) = 1$, and if ω is infinite, then $v_i(\omega) = v_{ij}(\omega) = 0$, so $v_i = v_{ij}$ on Ω_{ij}. Also by definition $\mu_i = p_{ij}\mu_{ij}$ on Ω_{ij} ,

so $$\sum_{j=1}^{n_i} \int_{\Omega_{ij}} v_i \, d\mu_i = \sum_{j=1}^{n_i} \int_{\Omega_{ij}} v_{ij} \, d\, p_{ij}\mu_{ij} = \sum_{j=1}^{n_i} p_{ij} \int_{\Omega_{ij}} v_{ij} \, d\mu_{ij}$$

$$= \sum_{j=1}^{n_i} p_{ij} \, V_{ij}$$

That is $V_i = \sum_j p_{ij} \, V_{ij}$.

Similarly if $\Omega_{ij} \cong \prod_k \Omega_{ijk}$ and if $\omega \in \Omega_{ij}$ and the sequence $(\omega_1, \ldots, \omega_{n_{ij}}) \in \prod_k \Omega_{ijk}$ is its corresponding n_{ij}-tuple of parses then ω is finite only if all the ω_k are finite. In which case $v_{ij}(\omega) = \prod_k v_{ijk}(\omega_k) = 1$. Otherwise ω is infinite, and at least one ω_k is infinite. In which case $v_{ij}(\omega) = \prod_k v_{ijk}(\omega_k) = 0$.

Hence $V_{ij} = \int_{\Omega_{ij}} v_{ij} \, d\mu_{ij} = \int_{\prod_k \Omega_{ijk}} \prod v_{ijk} \, d\prod \mu_{ijk} = \prod_k \int_{\Omega_{ijk}} v_{ijk} \, d\mu_{ijk}$

$$= \prod_k V_{ijk}.$$

Collecting the two results together gives the theorem.

1.5.2 Remark

Unfortunately, the recursive equations for the measure of the finite parses are not in general easy to solve, nor do they always have a unique solution. And it is easy to see by inspection that they always have the solution $V_1 = V_2 = \ldots = V_n = 1$. However the following obviously holds.

1.5.3 Corollary

If the only solution of the above equations with $0 \leq V_i \leq 1$ is $V_i = 1$, then the measure is concentrated on the finite parses.

1.5.4 Example

The equations for the language of assignment statements are

$$V_1 = p_{11}V_2V_0V_3$$

$$V_2 = p_{21}V_0 + p_{22}V_0$$

$$V_3 = p_{31}V_0V_3V_0V_3V_0 + p_{32}V_0V_3V_0V_3V_0 + p_{33}V_0 + p_{34}V_0$$

simplifying by using $p_{11} = p_{21} + p_{22} = p_{31} + p_{32} + p_{33} + p_{34} = V_0 = 1$ and writing q for $p_{31} + p_{32}$

$$V_1 = V_2V_3$$

$$V_2 = 1$$

$$V_3 = qV_3^2 + (1-q)$$

The last equation factors to $[qV_3 - (1-q)][V_3-1] = 0$

so $V_3 = 1$ or $V_3 = \frac{1-q}{q}$.

The second solution may be excluded if $\frac{(1-q)}{q} > 1$, that is if $q < \frac{1}{2}$, in which case it is known that all the measure is concentrated on the finite parses.

1.5.5 Notation

The number of nodes in a parse ω will be denoted by $h(\omega)$. The integral of h over a space Ω_i will be denoted by H_i. So $H_i = \int_{\Omega_i} h d\mu_i$. Similarly H_{ij} and H_{ijk} will be defined as the integrals of h over Ω_{ij} and Ω_{ijk} respectively.

1.5.6 Theorem

If the BNF of a grammar is

$$X_i = \sum_j p_{ij} \prod_k X_{ijk} \qquad (i=1,\ldots,n)$$

then the average number of nodes H_i obey the equations

$$H_i = 1 + \sum_j p_{ij} \sum_k H_{ijk} \qquad (i=1,\ldots,n)$$

Proof: The result is a simplification of

$$H_i = \sum p_{ij}(1 + \sum H_{ijk}) \qquad (i=1,\ldots,n)$$

which can be proved in a similar way to the last theorem. The key step is that if ω is a parse whose direct subparses are $(\omega_1,\ldots,\omega_{n_{ij}})$ then $h(\omega) = 1 + \sum_k h(\omega_k)$. Hence $H_{ij} = 1 + \sum_k H_{ijk}$

1.5.7 Example

The equations for the language of assignment statements simplify (using the previous notation including $p_{31} + p_{32} = q$) to

$$H_1 = 2 + H_2 + H_3$$

$$H_2 = 2$$

$$H_3 = q(2H_3 + 3) + 2 - q$$

Hence $H_3 = 2(\frac{1+q}{1-2q})$ or else $H_3 = +\infty$, and $H_1 = H_3 + 4$.

Now H_3 must be positive, hence if $\frac{1}{2} \leqslant q$ then the only possibility is that $H_3 = +\infty$. Combining this with the previous result it is reasonable to suppose that there are three possibilities:

$q < \frac{1}{2}$ in which case all the measure is on the finite parses and

$$H_1 = 4 + 2(\frac{1+q}{1-2q})$$

$q = \frac{1}{2}$ in which case all the measure is on the finite parses and

$$H_1 = +\infty$$

$q > \frac{1}{2}$ in which case the measure of the finite parses is $\frac{1-q}{q}$.

It should be noted that in the production $P_{ij} = (X_i \rightarrow X_{ij1}\ X_{ij2}\ \ldots\ X_{ijn_{ij}})$, each of the X_{ijk}'s is either a terminal or a non-terminal.

If X_{ijk} is a terminal then it is easy to see that $H_{ijk} = 1$. On the other hand, if X_{ijk} is a non-terminal then it must be some X_l say, and so H_{ijk} is H_l. Thus the equation

$$H_i = 1 + \sum_j p_{ij} \sum_k H_{ijk}$$

may be rewritten as a sum of terms in H_l as

$$H_i = 1 + \sum_j p_{ij}(t_{ij} + \sum_l t_{ijl} H_l)$$

where t_{ij} is the number of terminals amongst the X_{ijk}, and t_{ijl} is the number of occurrences of X_l. This rearranges to the linear equation in the H_i's,

$$H_i = 1 + \sum_j p_{ij} t_{ij} + \sum_l (\sum_j p_{ij} t_{ijl}) H_l$$

This can be rewritten even more clearly by putting

$$a_i = 1 + \sum_j p_{ij} t_{ij} \quad \text{and} \quad A_{il} = \sum_j p_{ij} t_{ijl}$$

as $H_i = \sum_l A_{il} H_l + a_i$.

This equation, because it is linear, has in general only one finite solution. However it may also have some infinite solutions. The matrix $A_{ij} = \sum_l p_{il} t_{ilj}$ constantly reappears, and so the same letter A will be used for it throughout what follows.

1.5.8 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \sum_{k=1}^{n_i} X_{ijk} \qquad (i=1,\ldots,n)$$

then the average length $L_i(L_{ijk})$ of word generated by parses in $\Omega_i(\Omega_{ijk})$ satisfies the equations

$$L_i = \sum_{j=1}^{n_i} p_{ij} \sum_{k=1}^{n_{ij}} L_{ijk} \qquad (i=1,\ldots,n)$$

which may be rewritten as

$$L_i = \sum_{j=1}^{n} A_{ij} L_j + b_i \qquad (i=1,\ldots,n)$$

where $b_i = \sum_{j=1}^{n_i} p_{ij} t_{ij}$ and A_{ij} and t_{ij} are as before.

Proof: There are two points to be noticed in the proof of the first formula. Firstly that if ω is a parse, then the length of the string generated by ω is the same as the number of terminals in ω and so $l(\omega)$ may stand indiscriminately for both. And secondly if $\omega_1,\ldots,\omega_m$ are the immediate subparses of ω then $l(\omega) = \sum_{i=1}^{m} l(\omega_i)$, so that if L_{ij}, L_{ijk} stand respectively for the integrals $\int_{\Omega_{ij}} d\mu_{ij}$ and $\int_{\Omega_{ijk}} d\mu_{ijk}$ then $L_{ij} = \sum_{k=1}^{n_{ij}} L_{ijk}$.

The linear equation can be derived by algebraic manipulation in a similar way to that for the average number of nodes.

1.5.9 Definition

The mean square of a measurable function g which maps a probability space Ω to the reals is defined to be $\int_{\Omega} g^2 \, d\mu$. The variance Var(g) of the same function is defined to be $\int_{\Omega} (g-\mathcal{E}(g))^2 d$ where the real number $\mathcal{E}g = \int_{\Omega} g d\mu$ is the expectation of g. $\mathrm{Var}_i(g)$ may be written for $\int_{\Omega_i} (g-\mathcal{E}g)^2 d\mu_i$, similarly $\mathrm{Var}_{ij}(g)$ and $\mathrm{Var}_{ijk}(g)$.

1.5.10 Lemmas

1. $\mathrm{Var}(g) = \int g^2 d\mu - (\mathcal{E}g)^2$

2. If $g = \sum_{i=1}^{n} g_i$ and the g_i are independent then $\mathrm{Var}(g) = \sum_{i=1}^{n} \mathrm{Var}(g_i)$.

Proofs:

1. $\mathrm{Var}(g) = \int (g-\mathcal{E}g)^2 d\mu = \int g^2 - 2g\mathcal{E}g + (\mathcal{E}g)^2 \, d\mu = \int g^2 d\mu -$

$- 2\mathcal{E}g \int g d\mu + (\mathcal{E}g)^2 \int 1 d\mu = \int g^2 d\mu - 2\mathcal{E}g\,\mathcal{E}g + (\mathcal{E}g)^2$

$= \int g^2 d\mu - (\mathcal{E}g)^2.$

Hence given any two of the three real numbers, var(g), $\mathcal{E}$ g or the mean square of g, the third can easily be calculated.

2. Proved in Loeve, p.12 and there called the Bienaymé equality or Feller, p.230.

1.5.11 Notation

If l is the length function as above, then $L_i^{(2)} = \int_{\Omega_i} l^2 d\mu$ is its mean square. Similarly $L_{ij}^{(2)}$ and $L_{ijk}^{(2)}$ are the corresponding integrals over Ω_{ij} and Ω_{ijk}. Care should be taken over the distinction between $L_i^{(2)}$ meaning $\int_{\Omega_i} l^2 d\mu_i$ and L_i^2 meaning $(\int_{\Omega_i} l \, d\mu_i)^2$. They are not in general the same real number.

1.5.12 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} \qquad (i=1,\ldots,n)$$

then the mean squares $L_i^{(2)}$, $L_{ij}^{(2)}$ and $L_{ijk}^{(2)}$ of the length function l satisfy the equations

$$L_i^{(2)} = \sum_{j=1}^{n_i} p_{ij} \left[(L_{ij})^2 + \sum_{k=1}^{n_{ij}} \left[L_{ijk}^{(2)} - (L_{ijk})^2 \right] \right] \quad (i=1,\ldots,n)$$

Proof: This has the usual two parts, first a relation between the integrals over Ω_i and Ω_{ij}, and second between those over Ω_{ij} and Ω_{ijk}. Trivially if g is any function to the reals, then the function g^2 where $g^2(x) = g(x)^2$ is also such a function, and so

$$\int_{\Omega_i} g^2 d\mu_i = \sum_{j=1}^{n_i} p_{ij} \int_{\Omega_{ij}} g^2 d\mu_{ij} \qquad (i=1,\ldots,n)$$

in particular for $g = l$, and using the above notation

$$L_i^{(2)} = \sum_{j=1}^{n_i} p_{ij} L_{ij}^{(2)} \qquad (i=1,\ldots,n)$$

The above gives a relation between the mean squares of l over Ω_i and Ω_{ij}, the Bienaymé equality can now be used to obtain a relation between the variances of l over Ω_{ij} and Ω_{ijk}. For if $\omega \in \Omega_{ij}$ and $\omega_k \in \Omega_{ijk}$ $(k=1,\ldots,n_{ij})$ are its corresponding subparses, then the functions $l_k(\omega) = l(\omega_k)$ are obviously well defined and independent on Ω_{ij}. Hence

$$\mathrm{Var}_{ij}(l) = \sum_{k=1}^{n_{ij}} \mathrm{Var}_{ij}(l_k)$$

But $\mathrm{Var}_{ij}(l_k) = \mathrm{Var}_{ijk}(l)$, hence using lemma 1.5.10.2

$$L_{ij}^{(2)} - (L_{ij})^2 = \sum_{k=1}^{n_{ij}} [L_{ijk}^{(2)} - (L_{ijk})^2] \qquad \begin{matrix}(i=1,\ldots,n;\\ j=1,\ldots,n_i)\end{matrix}$$

The result follows from the two halves by a simple algebraic rearrangement .

1.5.13 Corollary

The mean squares satisfy the linear equations

$$L_i^{(2)} = \sum_{h=1}^{n} A_{ih} L_h^{(2)} + c_i \qquad (i=1,\ldots,n)$$

where

$$c_i = \sum_{j=1}^{n_i} p_{ij} [t_{ij} + (L_{ij})^2 - \sum_{k=1}^{n_{ij}} (L_{ijk})^2]$$

and all the other signs are as before.

Proof: This is an algebraic manipulation, the only possible slight difficulty being that if X_{ijk} is a terminal then $L^{(2)}_{ijk} = 1$.

1.5.14 Remark

It is often quite easy to find various parameters which give some sort of average structure of the parses, all that is necessary is to find the right function and integrate it; for instance to find the probability that the left most terminal of a parse is x the function $s_x(\omega) = 1$ when the left most terminal of the parse ω is x and 0 otherwise must be chosen.

1.5.15 Theorem

If the BNF of a grammar is

$$X_i ::= \sum p_{ij} \prod X_{ijk}$$

and $S_{i,x}$ means $\int_{\Omega_i} s_x \, d\mu_i$ and similarly $S_{ij,x}$ and $S_{ijk,x}$ then

$$S_{i,x} = \sum_{j=1}^{n_i} p_{ij} \, S_{ij1,x} \qquad (i=1,\ldots,n)$$

Proof: If ω is a parse and $\omega_1,\ldots,\omega_h$ its immediate subparses then $s_x(\omega) = s_x(\omega_1)$. Hence $S_{ij,x} = S_{ij1,x}$.

1.5.16 Corollary

The similar relations

$$T_{i,x} = \sum_{j=1}^{n_i} p_{ij} \, T_{ijn(i,j),x}$$

hold between the probabilities $T_{i,x}$ etc. that x is terminal .

1.5.17 Remark

The frequencies of the various letter pairs can also be found. Care must be taken in the definition of how many letter pairs occur in a string because there are several slightly different possibilities. Firstly there is a difficulty about whether long sequences of the same letter should be broken into overlapping or non-overlapping pairs. Here overlapping pairs are chosen. For instance 'abbbc' is considered to have two overlapping pairs 'bb', not one non-overlapping pair. Secondly it is preferable for the theory of LR(k) grammars if the last symbol of a sequence is built into a pair, for instance the string 'abc' is considered to consist of pairs 'ab', 'bc' and 'c ' rather than just two. However this will be left until the parsing is dealt with, and the less preferable definition chosen.

1.5.18 Notation

c_{xy} is the function which gives the number of pairs 'xy' in a parse. $C_{i,xy}$, $C_{ij,xy}$ and $C_{ijk,xy}$ are the usual integrals $\int_{\Omega_i} c_{xy}\, d\mu_i$, $\int_{\Omega_{ij}} c_{xy}\, d\mu_{ij}$ and $\int_{\Omega_{ijk}} c_{xy}\, d\mu_{ijk}$ respectively.

1.5.19 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} \qquad (i=1,\ldots,n)$$

then the average numbers $C_{i,xy}$, $C_{ijk,xy}$ of terminal pairs 'xy' in a parse satisfy the equations

$$C_{i,xy} = \sum_{j=1}^{n_i} p_{ij} \left[\sum_{k=1}^{n_{ij}} C_{ijk,xy} + \sum_{k=2}^{n_{ij}} T_{ij(k-1),x}\, S_{ijk,y} \right]$$

Proof: This is just a question of noting that if $\omega_1,\ldots,\omega_h$ are the immediate subparses of ω , then every pair 'xy' which occurs in ω either occurs inside some ω_i, or else 'x' is the last symbol in some ω_i and 'y' the first of its successor ω_{i+1}

Thus

$$c_{xy}(\omega) = \sum_{i=1}^{h} c_{xy}(\omega_i) + \sum_{i=2}^{h} t_x(\omega_{i-1})\, s_y(\omega_i)$$

and the result follows by integration .

1.5.20 Corollary

The average numbers of pairs $C_{i,xy}$ obey the linear equations

$$C_{i,xy} = \sum_{h=1}^{n} A_{ih}\, C_{h,xy} + d_{i,xy} \qquad (i=1,\ldots,n)$$

where

$$d_{i,xy} = \sum_{j=1}^{n_i} p_{ij} \left[\sum_{k=2}^{n_{ij}} T_{ij(k-1),x}\, S_{ijk,y}\right]$$

and the other notation is as before.

Proof: If X_{ijk} is a terminal then clearly $C_{ijk,xy} = 0$ whatever x and y may be .

1.5.21 Remark

Finally an equation satisfied by the entropy will be worked out, where the entropy is as defined in the footnote*.

1.5.22 Notation

E_i, E_{ij} and E_{ijk} are the entropies of the sets Ω_i, Ω_{ij}, Ω_{ijk}. q will be a probability function, thus $q_i(\omega) = \mu_i(\{\omega\})$, where it is assumed that any of the measures μ is concentrated on the finite parses ω .

---

* If μ is a discrete measure over a set Ω then its entropy E is

$$\sum_{\omega\in\Omega} -\mu(\omega)\log\mu(\omega)$$

1.5.23 Theorem

If the BNF of a grammar is

$$X_i ::= \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} X_{ijk} \qquad (\text{for } i=1,\ldots,n)$$

then the entropies of the spaces Ω_i obey the relations

$$E_i = -\sum_{j=1}^{n_i} p_{ij} \log p_{ij} + \sum_{j=1}^{n_i} p_{ij} \sum_{k=1}^{n_{ij}} E_{ijk}$$

Proof: By the fundamental theorem on entropies

$$E_{ij} = \sum_{k=1}^{n_{ij}} E_{ijk} .$$

If $\omega \in \Omega_{ij}$ then $q_i(\omega) = p_{ij}\, q_{ij}(\omega)$

so $E_i \stackrel{df}{=} \sum_{\omega \in \Omega_i} - q_i(\omega) \log q_i(\omega)$

$$= \sum_{j=1}^{n_i} \sum_{\omega \in \Omega_{ij}} - p_{ij} q_{ij}(\omega) \log p_{ij} q_{ij}(\omega)$$

$$= -\left[\sum_{j=1}^{n_i} p_{ij} \log p_{ij} \sum_{\omega \in \Omega_{ij}} q_{ij}(\omega)\right] - \left[\sum_{j=1}^{n_i} p_{ij} \sum_{\omega \in \Omega_{ij}} q_{ij}(\omega) \log q_{ij}(\omega)\right]$$

and then because

$$\sum_{\omega \in \Omega_{ij}} q_{ij}(\omega) = \mu_{ij}(\Omega_{ij}) = 1 \text{ and } -\sum_{\omega \in \Omega_{ij}} q_{ij}(\omega) \log q_{ij}(\omega) = E_{ij}$$

$$E_i = -\sum_{j=1}^{n_i} p_{ij} \log p_{ij} + \sum_{j=1}^{n_i} p_{ij} E_{ij}$$

The result follows by collecting together the two halves.

1.5.24 Corollary

The entropies obey the linear equation

$$E_i = \sum_{h=1}^{n} A_{ih} E_h + e_i$$

where $e_i = -\sum_{j=1}^{n_i} p_{ij} \log p_{ij}$

Proof: If X_{ijk} is a terminal then $E_{ijk} = 0$.

1.6 ALL THE PARAMETERS CALCULATED FOR AN EXAMPLE

The various parameters can be calculated for the language of assignment statements as defined before with the BNF:

$$S::= \{p_{11}\}\ L=R$$

$$L::= \{p_{21}\}\ a \mid \{p_{22}\}\ b$$

$$R::= \{p_{31}\}\ (R+R) \mid \{p_{32}\}\ (RxR) \mid \{p_{33}\}\ a \mid \{p_{34}\}\ b$$

Now if S stands for X_1, L for X_2 and R for X_3 then the only non-zero terms A_{ij} are

$$A_{12} = p_{11}t_{112} = 1x1 = 1; \quad A_{13} = p_{11}t_{113} = 1$$

and $$A_{33} = p_{31}t_{313} + p_{32}t_{323} + p_{33}t_{333} + p_{34}t_{343} = 2q$$

where t_{ijk} is the number of non-terminals X_k in production ij, and $q = p_{31} + p_{32}$.

Hence $$I-A = \begin{pmatrix} 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1-2q \end{pmatrix}$$

and $$(I-A)^{-1} = \begin{pmatrix} 1 & 1 & 1/1-2q \\ 0 & 1 & 0 \\ 0 & 0 & 1/1-2q \end{pmatrix}$$

Now the a_i as defined in section 1.5.7 turn out to be

$$a_1 = 1 + 1x1 = 2$$

$$a_2 = 1 + p_{21}1 + p_{22}1 = 2$$

$$a_3 = 1 + 3p_{31} + 3p_{32} + p_{33} + p_{34} = 1 + 3q + (1-q) = 2(1+q)$$

hence the vector H of the average number of nodes which satisfies

$H = (I-A)^{-1} a$ where a is the vector of a_i's is

$$H_1 = 2 + 2 + (\frac{1}{1-2q}) 2(1+q) = 4 + 2(\frac{1+q}{1-2q})$$

$$H_2 = 2$$

$$H_3 = \frac{2(1+q)}{1-2q}$$

which is the same result as before.

Similarly the values b_i to calculate the average lengths are

$$b_1 = 1; \quad b_2 = 1; \quad b_3 = 1+2q$$

and so the vector L of average lengths which satisfies $L = (I-A)^{-1}b$ is

$$L_1 = 1 + 1 + \frac{1+2q}{1-2q}; \quad L_2 = 1; \quad L_3 = \frac{1+2q}{1-2q}$$

To calculate the mean squares, it should just be noted that if X_{ijk} is a terminal then $L_{ijk} = 1$ and if not then X_{ijk} is some non-terminal X_n say and so L_{ijk} is L_n. So for instance in the grammar considered here X_{321} is '(' a terminal and so $L_{321} = 1$ and X_{113} is $R=X_3$ so $L_{113} = L_3$. Similarly the L_{ij} are the sums of the average lengths generated by the component symbols of production ij. So for instance $L_{11} = L_2 + 1 + L_3$. For the assignment statement grammar L_1, L_2 and L_3 have just been calculated above, so:

$$c_1 = p_{11}[t_{11} + (L_{11})^2 - \sum_{k=1}^{3}(L_{11k})^2]$$

$$= 1 + (L_2+1+L_3)^2 - (L_2)^2 - (1) - (L_3)^2$$

$$= 1 + 2(L_2L_3 + L_2 + L_3)$$

$$= 1 + 2(\frac{1+2q}{1-2q} + 1 + \frac{1-2q}{1-2q}) = 3 + 4(\frac{1+2q}{1-2q}).$$

Similarly,

$$c_2 = p_{21}(1+1-1) + p_{22}(1+1-1) = 1$$

and $$c_3 = (p_{31}+p_{32})[3 + (3+2L_3)^2 - 3 - 2(L_3)^2] + (p_{33}+p_{34})$$

which simplifies to

$$\frac{-8q^3 - 20q^2 + 18q + 1}{(1-2q)^2}$$

Hence as the vector $L^{(2)}$ satisfies $L^{(2)} = (I-A)^{-1}c$

$$L_1^{(2)} = 3 + 4\frac{1+2q}{1-2q} + 1 + \frac{-8q^3 - 20q^2 + 18q + 1}{(1-2q)^3}$$

$$L_2^{(2)} = 1$$

$$L_3^{(2)} = \frac{-8q^3 - 20q^2 + 18q + 1}{(1-2q)^3}$$

$T_{i,x}$ and $S_{i,x}$ can for this grammar be worked out directly to give the two tables

$S_{i,x}$	a	b	(
S	p_{21}	p_{23}	
L	p_{21}	p_{22}	
R	p_{33}	p_{34}	q

$T_{i,x}$	a	b	)
S	p_{33}	p_{34}	q
L	p_{21}	p_{22}	
R	p_{33}	p_{34}	q

where the i,x entry of the first table means for instance that if i=1 and so X_1=S and x=a then the probability that a string generated by S starts with a is p_{21}. The blank spaces and omitted columns are all zero.

Using the above tables the following table can be calculated with a little effort whose xy entry gives the average number of pairs xy to be found in a string generated by S. The similar tables for L (which has all 0 entries) and R are not given.

	a	b	=	(	)	+	x
a			p_{21}		$\propto qp_{33}$	$\propto p_{31}p_{33}$	$\propto p_{32}p_{33}$
b			p_{22}		$\propto qp_{34}$	$\propto p_{31}p_{34}$	$\propto p_{32}p_{33}$
=	p_{33}	p_{34}		q			
(	$\propto qp_{33}$	$\propto qp_{34}$		$\propto q^2$			
)					$\propto q^2$	$\propto qp_{31}$	$\propto qp_{32}$
+	$\propto p_{31}p_{33}$	$\propto p_{31}p_{34}$		$\propto qp_{31}$			
x	$\propto p_{32}p_{33}$	$\propto p_{32}p_{34}$		$\propto qp_{32}$			

where $\propto$ stands for $\frac{1}{1-2q}$.

Finally the entropy can be calculated for

$e_1 = 0$

$e_2 = - p_{21}\log p_{21} - p_{22}\log p_{22}$

and $e_3 = - p_{31}\log p_{31} - p_{32}\log p_{32} - p_{33}\log p_{33} - p_{34}\log p_{34}$

and so using the condition for the vector E of entropies $E = (I-A)^{-1}e$

$E_1 = e_2 + e_3/(1-2q)$

$E_2 = e_2$

and $E_3 = e_3/(1-2q)$.

Chapter 2

GENERATING FUNCTIONS FOR CONTEXT FREE GRAMMARS

This chapter has three purposes. Firstly to justify some of the previous calculations and prove that they work in all circumstances (or at least specify in which circumstances they work). Secondly to show how some of the calculations can be done in a more unified way. Thirdly to show how some more parameters of context free grammars can be defined and calculated, in particular an expression for the information rate will be given.

The means used to achieve these ends will be to define some generating functions associated with probabilistic context-free grammars, and investigate their properties. Unfortunately a lot of work is needed to prove properties of these generating functions which are of no direct interest as far as the grammars are concerned, and much of the chapter will be taken up with this work.

2.1 PRELIMINARIES

In this section are listed some standard properties and definitions which are used in this chapter. Firstly some properties of generating functions in one variable, then some theorems to do with algebraicity, and finally some properties of multivariate generating functions.

2.1.1 Univariate Generating Functions - Definitions

If $\langle a_i \rangle_{i=0}^{\infty}$ is an infinite sequence of complex numbers, and z is a complex number, then $\langle a_i z^i \rangle_{i=0}^{\infty}$ and $\langle A_n(z) \rangle_{n=0}^{\infty}$ are two more such sequences, where $A_n(z) = \sum_{i=0}^{n} a_i z^i$. For any particular z the sequence $A_n(z)$ may either converge, oscillate, or diverge.

$A(z)$, called the generating function for the sequence $\langle a_i \rangle$, is that function which is defined only for those z's for which $\langle A_n(z) \rangle$ converges, and whose value is $\lim_{n\to\infty} A_n(z)$ there. It turns out that there is always a real number r (0, positive or $+\infty$), called the radius of convergence of $\langle a_i \rangle$, such that if $|z| < r$ then $A(z)$ is defined, and if $|z| > r$ then $\langle A_n(z) \rangle$ diverges. There is no general rule for what happens when $|z| = r$. The set $\{z : |z| < r\}$ is called the circle of convergence of $\langle a_i \rangle$.

The above states that given any sequence there is a unique function with various properties; the inverse also holds for cases of practical interest. In particular, given any function $A(z)$ defined, differentiable and finite everywhere within some open neighbourhood of the origin, then there is a unique sequence $\langle a_i \rangle$ such that the limit of $A_n(z)$ is the same as $A(z)$ inside the neighbourhood. If $A(z)$ and $B(z)$ are two such functions (defined, differentiable and finite in open neighbourhoods of the origin) then $C(z) = A(z) + B(z)$ and $D(z) = A(z) \times B(z)$ are two more such functions defined at least in the intersection of the neighbourhoods of A and B. It turns out that the sequence $\langle c_i \rangle$ for the function C is the sum of those for A and B, that is $c_i = a_i + b_i$; and that $\langle d_i \rangle$ the sequence for D is the convolution of those for A and B, that is $d_i = \sum_{j=0}^{i} a_j b_{(i-j)}$. $\frac{d}{dz}(A(z))$ is also of the above sort, and the ith term of its sequence is $(i+1)a_{(i+1)}$.

2.1.2 Analytic Continuation

Although the function defined by a sequence is only given inside its circle of convergence, it is often possible to extend its domain. For if z_o is a point inside the circle, then $B(z) = A(z+z_o)$ is another function which is defined, differentiable and finite in

a neighbourhood of the origin. Hence it has a sequence $\langle b_i \rangle$ which in turn defines a new function B'. Now B'=B when both are defined and B'(z) is defined inside a circle of radius r'. By defining $A(z) = B'(z-z_o)$ inside a circle of radius r' about z_o, it often happens that A is defined for new values of z such that $|z| > r$, and the domain of A is extended. If this process of analytic continuation is done again and again, it often happens in practice that the function A can be defined for all z in the complex plane except for a finite number of singular points.

2.1.3 Singular Points

Singular points can be classified according to their properties. Only two sorts are of interest here. A pole at z_o of a function A is a point where A(z) becomes infinite, that is $\lim_{z \to z_o} (\frac{1}{A(z)}) = 0$. A branch point is more complicated. If a function is extended by analytic continuation by going round a loop (circle, jordan curve) in small steps, and there are no branch points inside the loop, then the value of the function at a point after going round the circle is the same as at the beginning. On the other hand, on going round a branchpoint this is not in general so. A function with branch points may be thought of as having a number of layers, some of which are stuck together at a branch point. Some complications are that a point may be simultaneously a pole and a branch point, also if a function has several layers, a point may be a pole on just one layer, or a branch point connecting not all.

2.1.4 Expansion about a Pole

A theorem stated by Feller [7], p.285 allows the limiting behaviour of the sequence $\langle a_i \rangle$ to be obtained from the properties of its generating function A(z) at the pole z_o of A nearest to the origin.

Theorem

If $A(z) = \lim_{n\to\infty} \sum_{i=0}^{n} a_i z^i$ and also $A(z) = U(z)/V(z)$, and the following three conditions hold.

1. z_o is a root of V but not of U.

2. z_o is the root of V strictly smallest in absolute value (i.e. the root of V strictly nearest the origin and hence also the pole of A strictly nearest the origin).

3. z_o is a root of multiplicity r (i.e. $V(z_o) = \frac{d}{dz} V(z_o) = \ldots = \frac{d^{r-1}}{dz^{r-1}} V(z_o) = 0$ but $\frac{d^r}{dz^r} V(z_o) \neq 0$).

Then as $n \to \infty$

$$a_n \to \frac{(-1)^r \, r! \, U(z_o)}{z_o^{n+r} \; \frac{d^r}{dz^r} V(z_o)} \binom{n+r-1}{r-1}$$

Remark

The two particular cases used here have r=1 and r=2. They can be restated more simply as follows.

Corollary

When the three conditions hold and r=1 then

$$a_n \to \frac{-U(z_o)}{z_o^{n+1} \, V'(z_o)}$$

Corollary

When the three conditions hold and r=2 then

$$a_n \to \frac{2 \, U(z_o)(n+1)}{z_o^{n+2} \, V''(z_o)}$$

Remark

In practice the main difficulty in using this theorem is in showing that condition 2 holds.

2.1.5 Algebraic Functions

Definition

A function $A(z)$ from complex numbers to complex numbers is algebraic if it satisfies a non-trivial equation of the form

$$P_o(z) \left(A(z)\right)^m + P_1(z) \left(A(z)\right)^{m-1} + \ldots + P_m(z) = 0$$

for all z, where all the $P_i(z)$ are polynomials in z. (The equation is non-trivial so long as the multinomial $P_o(z)\, w^m + \ldots + P_m(z)$ is not identically zero). Algebraic functions are important for this thesis because of the following:-

Theorem

If A is an algebraic function then its only singularities are poles and branch points.

Corollary

The non-singular points of A form an open subset of the complex plane.

One way to show that some particular functions are algebraic is to use the following theorem, which gives a multi-dimensional analogue of the defining equation. (Condition 2 gives a generalisation of the idea of non-triviality.)

Theorem

If $A_1(z), A_2(z), \ldots, A_n(z)$ are complex functions of the complex variable z, and

1) they obey n equations of the form

$$\mathcal{P}_j(z, A_1(z), \ldots, A_n(z)) \equiv 0 \quad \text{for all } z,\ j=1,\ldots,n;$$

where $\mathcal{P}_j(z, w_1, w_2, \ldots, w_n)$ is a multinomial of the form

$$\mathcal{P}_j(z, w_1, \ldots, w_n) \equiv \sum_{i_o, \ldots i_n \in \mathbb{I}} a^{(j)}_{i_o \ldots i_n} z^{i_o} w_1^{i_1} \ldots w_n^{i_n},$$

where the $a^{(j)}_{i_o \ldots i_n}$ are complex numbers, and only finitely many of them are non zero;

2) the jacobian $\left(\frac{\partial \varphi_j}{\partial w_i}\right) \neq 0$ at at least one point $\langle z_o, A_1(z_o), \ldots, A_n(z_o) \rangle$;

then each of the functions $A_i(z)$ is algebraic.

Comment on Proof

As far as I know, this theorem is not in the literature. I therefore attempted to prove it myself. My method would have constructed polynomials $P^{(i)}_o(z), \ldots, P^{(i)}_{m_i}(z)$, and so given a direct proof that A_i was algebraic. Unfortunately it failed to invariably find non-trivial polynomials in the rare cases when some of the A_i had cusps or similar peculiarities at z_o.

Luckily I managed to interest Dr. G. Segal in the problem and he succeeded in proving it. His proof uses mathematics which is needed nowhere else in this thesis. I have made it appendix 1 .

2.1.6 Multivariate Distribution Generating Functions - Definitions

Definition

Given a multiple series $\langle a_{i_o i_1 \ldots i_n} : i_o, i_1, \ldots, i_n = 0, \ldots \rangle$ of complex numbers, then the multiple sum

$$A(z_o, \ldots, z_n) = \sum_{i_o, \ldots, i_n = 0}^{\infty} a_{i_o \ldots i_n} \; z_o^{i_o} \ldots z_n^{i_n}$$

may be defined for some complex values $\langle z_o, \ldots, z_n \rangle$ by taking the limits of subsums in various suitable ways. The convergence conditions are more complicated than those for one dimensional series, but the sum clearly converges for all values if only finitely many of the $a_{i_\rho \ldots i_n}$ are non zero. This is the only case used here.

Definition

A function V is a vector valued function from a measure space $(\Omega, \mathcal{A}, \mu)$ if it is measurable and maps Ω to n+1-tuples of non-negative integers. If $\langle i_0,\ldots,i_n\rangle$ is such an n+1-tuple, then $V^{-1}(i_0,\ldots,i_n)$ is a measurable subset of Ω, so its measure $\mu(V^{-1}(i_0,\ldots,i_n))$ exists and is a non-negative real number. So the multiple series $\langle\mu(V^{-1}(i_0,\ldots,i_n))\rangle$ may be used to define a generating function $F(z_0,\ldots,z_n)$, which is called the distribution generating function (d.g.f.) of the vector valued function V.

2.1.7 Properties of Distribution Generating Functions

For the purpose of stating these properties, the m functions V_i will map the spaces $(\Omega_i, \mathcal{A}_i, \mu_i)$ to n+1-tuples, and have (d.g.f.)s F_i. Similarly V and F correspond to $(\Omega, \mathcal{A}, \mu)$. The equations are only asserted under the unnecessarily restrictive condition that all (d.g.f.)s mentioned converge.

Theorem

If $(\Omega, \mathcal{A}, \mu)$ is the weighted disjoint sum $\sum_{i=0}^{m} p_i(\Omega_i, \mathcal{A}_i, \mu_i)$ and V is the union of the V_i (i.e. if $\omega\in\Omega_i$ then $V(\omega) = V_i(\omega)$), then

$$F(z_0,\ldots,z_n) = \sum_{i=1}^{m} p_i\, F_i(z_0,\ldots,z_n)$$

Theorem

If $(\Omega, \mathcal{A}, \mu)$ is the product $\prod_{i=0}^{m} (\Omega_i, \mathcal{A}_i, \mu_i)$ and V is defined to be the sum of its components (i.e. for $\omega = \langle\omega_0,\ldots,\omega_m\rangle\in\Omega$, $V(\omega) = \sum_{i=1}^{m} V_i(\omega_i)$), then

$$F(z_0,\ldots,z_n) = \prod_{i=1}^{m} F_i(z_0,\ldots,z_n).$$

Theorem

The mean value of the i^{th} component of V on the space $(\Omega, \mathcal{A}, \mu)$ is

$$\left.\frac{\partial F}{\partial z_i}\right|_{z_o=z_1=\ldots=z_n=1}$$

The Proofs are obtained by rearranging the defining sequences.

2.2 VECTOR VALUED FUNCTIONS ASSOCIATED WITH GRAMMARS AND THE EXTINCTION PROBABILITY

In the theorems so far, it has been assumed whenever necessary that all the measure is concentrated on the finite parses. Luckily there is a simple test to show if this is in fact so. The derivation of this test is complicated, and depends on seeing a grammar as a particular example of a branching process. A derivation of a similar test for branching processes is reported in Harris [13], pp.34-48, but he requires an extra condition which only holds for a small and not linguistically interesting class of grammars. An alternative derivation using results from the theory of markov processes has been obtained by Hutchins [16].

2.2.1 Notational Convention

The reader should recall that the grammar dealt with here has productions $\left\{(X_i \to \prod_{k=1}^{n_{ij}} X_{ijk}) : i=1,\ldots,n;\ j=1,\ldots,n_i\right\}$, where the X_i are the non-terminals of the grammar and X_{ijk} is the k^{th} symbol in the j^{th} production of the i^{th} rule. Each X_{ijk} is either a non-terminal, and so one of the X_i, or a terminal, and hence corresponding to a dummy non-terminal X_o. Ω_i, Ω_{ij} and Ω_{ijk} are the sets of parses stemming from X_i, X_i through the production P_{ij} and X_{ijk}

respectively, where if X_{ijk} is X_o then Ω_{ijk} is Ω_o, or in other words the set containing just the single degenerate parse consisting of just a labelled root.

The pattern of subscripts i, ij, ijk occurs often. In general only symbols with subscript i are explicitly defined (in terms of Ω_i), symbols with subscripts ij and ijk must be assumed to have analogous definitions (in terms of Ω_{ij} or Ω_{ijk}). When X_i is X_{ijk}, the meaning of the symbol with subscript i is the same as, or equal to, or isomorphic to that with subscript i as required. Seldom does the symbol with subscript 0 need a special definition, but it often adds clarity if it is given one.

2.2.2 Vector Valued Functions Associated with Grammars

Definition

An important set of vector valued functions associated with a grammar is $\{V_i^{(d)}\}$, where $V_i^{(d)}$ maps Ω_i to n+1-tuples of integers, and if the parse $P_i \in \Omega_i$, then the j^{th} component of $V_i^{(d)}(P)$ gives the number of occurrences of X_j at depth d in P. (Or if j=0 the number of occurrences of terminals.)

$F_i^{(d)}(z_o,\ldots,z_n)$ is the (d.g.f.) corresponding to $V_i^{(d)}$.

Remark

Because the degenerate parse has a sole terminal at its root,

$F_o^{(o)}(z_o,\ldots,z_n) \equiv z_o$ and $F_o^{(d)}(z_o,\ldots,z_n) \equiv 1$ for $d>0$.

Remark

As d becomes larger, the functions $F_i^{(d)}$ may (but need not) become more and more complicated multinomials in the variables $z_o,\ldots,z_n$. Fortunately all these functions can be constructed in sequence directly from the functions $F_i^{(1)}$. Many of the properties

of the $F_i^{(d)}$ can also be obtained directly from the corresponding properties for $F_i^{(1)}$. The functions $F_i^{(1)}$ are thus particularly important, and will be written F_i for short.

Theorem

$$F_i^{(d+1)}(z_o,\ldots,z_n) \equiv F_i\left(F_o^{(d)}(z_o,\ldots,z_n),\ldots,F_n^{(d)}(z_o,\ldots,z_n)\right)$$

Proof

$V_{ij}^{(d+1)}$ and $V_i^{(d+1)}$ satisfy the conditions of theorem 2.1.7.1

so $$F_i^{(d+1)}(z_o,\ldots,z_n) \equiv \sum_{j=1}^{n_i} p_{ij} F_{ij}^{(d+1)}(z_o,\ldots,z_n)$$

Similarly, given a parse $\mathcal{P}_{ij}$, let $\mathcal{P}_{ijk}$ be the subparse whose root is the symbol X_{ijk} immediately below the root of $\mathcal{P}_{ij}$. Clearly any symbol at depth d+1 in $\mathcal{P}_{ij}$ is at depth d in one of the $\mathcal{P}_{ijk}$. Hence

$$V_{ij}^{(d+1)}(\mathcal{P}_{ij}) = \prod_{k=1}^{n_{ij}} V_{ijk}^{(d)}(\mathcal{P}_{ijk})$$

So using theorem 2.1.7.2

$$F_{ij}^{(d+1)}(z_o,\ldots,z_n) = \prod_{k=1}^{n_{ij}} F_{ijk}^{(d)}(z_o,\ldots,z_n).$$

But by inspection

$$F_i(z_o,\ldots,z_n) = \sum_{j=1}^{n_i} p_{ij} \prod_{k=1}^{n_{ij}} z_{ijk}$$

(where as usual z_{ijk} is z_ℓ if X_{ijk} is X_ℓ, and z_{ijk} is z_o if X_{ijk} is terminal).

The result follows by using that in this case $F_{ijk}^{(d)}$ is $F_\ell^{(d)}$.

Definition

$T_i^{(d)}$ is the probability that a parse in Ω_i has depth less than d.

Definition

$$p(i_o,\ldots,i_n) \stackrel{def}{=} \mu_i\left\{[v_i^{(d)}]^{-1}(i_o,\ldots,i_n)\right\}$$

in words $p(i_o,\ldots,i_n)$ is the probability that a parse starting from X_i shall have i_o occurrences of terminals and i_j occurrences of non-terminal X_j at depth d.

Corollary

$$T_i^{(d+1)} = F_i(T_o^{(d)}, \ldots, T_n^{(d)})$$

Proof

Clearly the depth of a parse is less than d iff it has exactly no symbols at depth d. Hence

$$T_i^{(d)} = p(0,\ldots,0).$$

But $p(0,\ldots,0)$ is the only non-zero term in the sum for $F_i^{(d)}(0,\ldots,0)$. Hence $T_i^{(d)} = F_i^{(d)}(0,\ldots,0)$ and the result is a special case of the previous theorem.

Remark

$T_i^{(o)} = 0$ for all i, and each of the functions $F_i(z_o,\ldots,z_n)$ is known explicitly. So the above corollary allows the vectors $\langle T_i^{(d)} \rangle$ of extinction probabilities to be worked out iteratively. Because each F_i is a rational function of its arguments and the preprobabilities p_{ij}, if all the p_{ij} are rational or algebraic then each $T_i^{(d)}$ has the same property.

Lemma

For each i, the sequence $T_i^{(d)}$ is non-decreasing and bounded above by one.

Proof

The bound exists because $T_i^{(d)}$ is a probability. The probability

that a parse in Ω_i is of depth less than d is of course less or equal to the probability that it is of depth less than d+1.

Corollary

The sequence $T_i^{(d)}$ tends to a finite limit $T_i \leq 1$.

Proof: Straightforward.

Corollary

$$T_i = F_i(T_o,\ldots,T_n)$$

Proof: Consequence of corollary.

Remark

An alternative way to say that all the measure is concentrated on the finite parses is to say that $T_i = 1$.

Lemma

$$F_i^{(d)}(1,\ldots,1) = 1$$

Proof: The left hand side is the sum of the probabilities of all the elements in a probability space.

2.2.3 The Matrix of Means

Definition

$M_{ij}^{(d)}$ for i,j=1,n is the mean number of symbols X_j occurring at depth d in a parse stemming from X_i. If j=0 it is the mean number of terminals, if i=0 the mean number of symbols in the degenerate parse (so $M_{oj}^{(d)} = 0$ for $j=1,\ldots,n$, $M_{oo}^{(o)} = 1$ and for $d>0, M_{oo}^{(d)} = 0$). M_{ij} is short for $M_{ij}^{(1)}$. $M^{(d)}$ and M are short for the matrices $\left(M_{ij}^{(d)}\right)(i,j=0,\ldots,n)$ and $\left(M_{ij}\right)$ $(i,j=0,\ldots,n)$ respectively. M.is an important matrix and will therefore be given a name, 'the matrix

of means'. (The notation M_{ij} comes from ordinary matrix theory, and not from the method of writing subscripts for grammatical symbols. There is no meaning for the symbols M_i and M_{ijk}.)

Remark

As can be easily seen by calculating both, the matrix A which appeared in chapter 1 [1.5.7] is the same as the cofactor of the leading term of M, that is, the matrix $\left(M_{ij}\right)$ $(i,j=1,\ldots,n)$.

Theorem

$$M^{(d)}_{ij} = \left[\frac{\partial}{\partial z_j} F^{(d)}_i \right] (1,..,1)$$

Proof: Special case of theorem 2.1.7.3.

Theorem

$M^{(d)} = M^d$ (where M^d is M multiplied by itself d times).

Proof: By induction. M is $M^{(1)}$ by definition. Suppose $M^{(d)} = M^d$ then $M^{(d+1)}_{ik} =$

$$= \left[\frac{\partial}{\partial z_k} F^{(d+1)}_i\right](1,\ldots,1) \qquad \ldots \text{ by above theorem}$$

$$= \left[\frac{\partial}{\partial z_k} F_i(F^{(d)}_o(z_o,\ldots,z_n),\ldots,F^{(d)}_n(z_o,\ldots,z_n))\right] \Bigg|_{z_o=\ldots=z_n=1}$$

$$\ldots \text{ by theorem 2.2.2.4}$$

$$= \sum_{j=0}^{n} \left(\left\{ \left[\frac{\partial F_i}{\partial z_j} \right] (F^{(d)}_o(1,\ldots,1),\ldots,F^{(d)}_n(1,\ldots,1)) \right\} X \left\{ \frac{\partial F^{(d)}_j}{\partial z_k} (1,\ldots,1) \right\} \right)$$

by the rule for the differentiation of a function of of a function

$$= \sum_{j=0}^{n} M_{ij} M^{(d)}_{jk} .$$

That is $M^{(d+1)} = MM^{(d)} = MM^d = M^{d+1}$. So the theorem holds for all d by induction.

2.2.4 Decomposition of Matrix of Means

Definition

Any square matrix is said to be reducible if it can be put into the form

$$\begin{pmatrix} A_o & 0 \\ B & A_1 \end{pmatrix}$$

by using the same permutation on both rows and columns. In the diagram both A_o and A_1 are square, and 0 is a block of all zero entries.

A square matrix is irreducible if it is not reducible.

Remark

Either A_0 or A_1 or both may themselves be further reducible, in any case every square matrix can eventually be written in the form

$$\begin{pmatrix} A_o & & 0 \\ & A_1 & \\ B & & A_m \end{pmatrix}$$

where all the A_i are irreducible.

Definition

'$\supset$' is a binary relation on the symbols of a context-free grammar. $X_i \supset X_j$ if X_j can occur in a parse stemming from X_i. '$\not\supset$' is the complementary relation, that is $X_i \not\supset X_j$ iff it is not the case that $X_i \supset X_j$.

Lemmas

'$\supset$' is transitive and reflexive. For reduced grammars (the only ones of interest here) if X_n is the start symbol and X_o represents the terminals then for all X_i, $X_n \supset X_i$ and $X_i \supset X_o$.

Proofs: Obvious.

Remark

Once a permutation has been carried out to get M to the form with irreducible blocks on its main diagonal, it is possible to renumber the terminals so that each term occurs in its correct place, X_o still represents the terminals and X_n the start symbol. This will be assumed to have been done in the following theorem which links M and the relation $\supset$.

Theorem

If $X_i \supset X_j$ and $X_j \supset X_i$ then M_{ii} and M_{jj} are both elements of the same irreducible diagonal component A_k of M, and conversely, if the latter holds so does the former.

If $X_i \supset X_j$ but $X_j \not\supset X_i$ then M_{ii} and M_{jj} are parts of different matrices $A_{k(i)}$ and $A_{k(j)}$ respectively, $k(i) > k(j)$ and $i > j$.

Proof: The key step is that $X_i \not\supset X_j$ iff $M_{ij}^{(d)} = 0$ for all d. That is if no parse stemming from X_i contains any symbols X_j, then the average number of symbols X_j at any depth d is also 0.

Remark

The above theorem allows the matrix M to be split into its irreducible components with comparatively little effort. In particular because $X_o \supset X_i$ only if i=0, the matrix A_o is the single number $M_{oo} = 0$ and M is therefore both singular and reducible. In practice M^{oo}, the cofactor of M_{oo}, is also usually reducible, for few grammars have both $X_i \supset X_j$ and $X_j \supset X_i$ for all pairs of non-terminals.

2.2.5 The Eigenvalues of the Matrix of Means

Lemma

All the entries $M_{ij}^{(d)}$ of any of the matrices $M^{(d)}$ are non-negative.

Proof: Each is an average of non-negative numbers.

Theorem

A non-negative matrix A always has a real non-negative eigenvalue r such that no eigenvalue of A has modulus exceeding r. To this 'dominant' eigenvalue there corresponds a non-negative eigenvector. (Where a matrix or vector is non-negative if all its entries are.)

If in addition A is irreducible then the dominant eigenvalue r is a simple root of the characteristic equation (corresponds to exactly one eigenvector).

Proof: This is very long and may be found in Gantmacher [10]. The case when A is irreducible is part of his theorem 2 (p.65) and the general case of theorem 3 (p.85).

Remark

The next two theorems are not used subsequently, but may help to give a feel for the properties of the eigenvalues.

Lemma

Every eigenvalue of a reducible matrix is an eigenvalue of one of its irreducible components and vice versa.

Proof: The key step is that if A splits into two not necessarily irreducible components A_o, A_1, so that

$$A = \begin{pmatrix} A_o & 0 \\ B & A_1 \end{pmatrix}$$

if v_o and v_1 are left and right eigenvectors of A_o and A_1 respectively, so that $v_o A_o = \lambda v_o$ and $A_1 v_1 = \mu v_1$,

then $(v_o, 0)A = \lambda (v_o, 0)$ and $A\begin{pmatrix} v_1 \\ 0 \end{pmatrix} = \mu \begin{pmatrix} v_1 \\ 0 \end{pmatrix}$.

If A_i is not reducible it can again be split, and the identity of eigenvalues carried down till an irreducible component is reached.

Theorem

The dominant eigenvalue of M (the matrix of means) is the greatest of the dominant eigenvalues of its irreducible components.

Theorem

If G(i) is the reduced grammar obtained from G by removing those X_j such that $X_i \not\supset X_j$ and also their productions, then the dominant eigenvalue of M(i) (the matrix of means of G(i)) is less than or equal to that of M.

Proof: The set of irreducible components of M(i) is a subset of the set of irreducible components of M.

2.2.6 Extinction Probability - Part 1

Remark

The next two theorems show that the measure of the set of finite parses is related to r, the dominant eigenvalue of M. If $r<1$ then almost all parses are finite but if $r>1$ then there is a finite probability that a parse does not terminate. These two statements are proved by entirely different methods, so they will be given as two separate theorems, one in this paragraph and the other in the next.

Theorem

If r is the dominant eigenvalue of the matrix of means of a reduced probabilistic context free grammar and $r<1$ then the measure of the set of finite parses is one.

Proof: M can be rewritten in Jordan normal form as $M = TBT^{-1}$. Here the only non-zero terms of B are in blocks B_i on the main diagonal, where each B_i has some eigenvalue α all along its leading diagonal, and 1's on the diagonal beneath. In a diagram

$$B = \begin{pmatrix} B_o & & 0 \\ & \ddots & \\ 0 & & B_p \end{pmatrix} \quad \text{and} \quad B_i = \begin{pmatrix} \alpha & & & \\ 1 & \alpha & & 0 \\ & 1 & \alpha & \\ & & \ddots & \ddots \\ 0 & & & 1 \end{pmatrix}$$

Powers of the B_is are of the form

$$\begin{pmatrix} \alpha^N & & & & \\ N\alpha^{N-1} & \alpha^N & & & \\ \frac{N(N-1)}{2}\alpha^{N-2} & N\alpha^{N-1} & \alpha^N & \ddots & \\ & & \ddots & \ddots & \alpha^N \end{pmatrix}$$

Hence $|B_i^N|$, the matrix of absolute values of entries of B_i^N, has values

$$|B_i^N| \leq \begin{pmatrix} |\alpha|^N & & & 0 \\ N|\alpha|^{N-1} & |\alpha|^N & & \\ N^2|\alpha|^{N-2} & N|\alpha|^{N-1} & \ddots & \\ & \ddots & \ddots & |\alpha|^N \end{pmatrix}$$

Now $\lim_{N\to\infty} |\alpha|^{N-q} = 0$ if $|\alpha| < 1$, and $|\alpha| < 1$ because $|\alpha| < r < 1$.

Hence if a and b are the largest absolute values of entries of T and T^{-1} respectively, and n+1 is the order of M, then given any (small) ϵ it is possible to find a N such that all terms of each $|B_i^N|$, and hence of $|B^N|$, are less than $\epsilon/ab(n+1)^3$. Let (1) be the matrix all of whose entries are 1 .

Then $|M^N| \leq |T|\,|B^N|\,|T^{-1}| \leq \frac{a.b.\epsilon}{a.b.(n+1)^3}(1)(1)(1) = \frac{\epsilon}{n+1}(1)$.

To recall the meaning of the entries in the matrix, the above states that $M_{ij}^{(N)}$, the average number of the symbol X_j at depth N in a parse starting from X_i, is less than $\epsilon/n+1$. In the worst possible case each parse of depth N or greater will have exactly one symbol at depth N, and thus the maximum number of parses will extend beyond this depth. But in this case the measure of the set of parses remaining is ϵ . ϵ can be chosen arbitrarily small, so the measure of the set of infinite parses is zero.

2.2.7 Extinction Probability - Part 2 and Conclusion

Theorem If r is the dominant eigenvalue of the matrix of means of a reduced probabilistic context free grammar and $r > 1$ then the measure of the set of finite parses is less than one.

Proof: The sequence of vectors $_{i=0}^{n}\langle T_i^{(d)}\rangle$ of probabilities that a parse starting from X_i has terminated by depth d tend to the limit vector $_{i=0}^{n}\langle T_i\rangle$ of probabilities that these parses terminate at all.

Suppose $\langle s_i\rangle$ is any point near $\langle 1\rangle$ such that each $s_i < 1$. Then $\langle s_i\rangle = \langle 1\rangle - \langle h_i\rangle$ where all the h_i are positive. This fact combined with that $r > 1$ gives that for some i, $h_i' > h_i$, where $\langle h_i'\rangle = M\langle h_i\rangle$. See HARRIS [13], but also take note of the final paragraphs of this proof. For this proof '$\triangleright$' ('$\triangleleft$') will be used for this relation between vectors (and its converse), that is $\langle a_i\rangle \triangleright \langle b_i\rangle$ means that at least one $a_i > b_i$.

Hence,

$$\langle F_i(1-h_o,\ldots,1-h_n)\rangle - \langle F_i(1,\ldots,1)\rangle \triangleq -M\langle h_i\rangle \triangleleft - \langle h_i\rangle$$

But by lemma 2.2.2.13, $\langle F_i(1,\ldots,1)\rangle = \langle 1,\ldots,1\rangle$, so the above may be rearranged to give

$$\langle F_i(1-h_o,\ldots,1-h_n)\rangle \triangleleft \langle 1-h_i\rangle$$

In other words, $\langle F_i(s_o,\ldots,s_n)\rangle \lhd \langle s_i\rangle$.

Another way of stating that the probability of termination is 1 is to say that $\langle T_i\rangle = \langle 1\rangle$. As $\langle T_i\rangle$ is the limit of the sequence $\langle T_i^{(d)}\rangle, \langle T_i^{(d)}\rangle$ must eventually cluster close enough to $\langle 1\rangle$ to allow it to be used for $\langle s_i\rangle$ to yield

$$\langle F_i(T_o^{(d)},\ldots,T_h^{(d)})\rangle \lhd \langle T_i^{(d)}\rangle$$

or in other words, using corollary 2.2.2.6

$$\langle T_i^{(d+1)}\rangle \lhd \langle T_i^{(d)}\rangle$$

That is, for some particular i, $T_i^{(d+1)} < T_i^{(d)}$, or in words, a parse starting from X_i is more likely to have terminated by depth d than by depth d+1. This is impossible, so $\langle T_i\rangle \neq \langle 1\rangle$.

There are two possible difficulties with the above proof. Firstly the proof that $M\langle h_i\rangle \rhd \langle h_i\rangle$ requires the matrix M to have a singular dominant eigenvalue. As M is not necessarily irreducible, it may be the case that its dominant eigenvalue is shared with two or more of its irreducible components, and so not singular. In this case one such component can be chosen and called the dominant component, and any symbol X_i such that M_{ii} is a part of this component called a dominant symbol. Powers $(A_{(k)})^d$ of the dominant component $(A_{(k)})$ give the average numbers of dominant symbols at depth d in parses stemming from dominant symbols. Because $r(k) > 1$ and r(k) is singular the theorem above holds, so the limit of the probability that a parse starting from a dominant symbol contains no dominant symbol is less than one. Hence the probability that a parse starting from a dominant symbol terminates is less than one, and so, because the grammar is reduced and there is a strictly positive probability of generating any symbol from the start symbol X_n, there is also a finite probability that a parse

starting from X_n does not terminate.

The second difficulty is easier to deal with. The sequence $T_i^{(d)}$ may attain its limit T_i after only a finite number of steps, and so falsify the condition that all $s_i < 1$. However this can only happen when X_i can only generate a finite language in which case the irreducible component corresponding to X_i reduces to the single entry M_{ii}, and $M_{ii}=0$. This clearly cannot be the dominant block and so the above analysis can also be used.

This is as much of this proof as will be given here. The remainder, that is the proof that $M\langle h_i\rangle \triangleright \langle h_i\rangle$, can be found in HARRIS

Corollary

The probability that all parses terminate is one if the vector $(I-M)^{-1}\langle 1\rangle$ is strictly positive, where I is the unit matrix and 1 the unit vector.

This is proved in HUTCHINS [16].

Remark

The last corollary is useful in practice because it is in general easier to invert a matrix than find its eigenvalues. Because a matrix can always be inverted by rational operations in finite time, it also gives a finite test for the condition. If I-M is not invertible, then M has 1 as an eigenvalue, so M fails the test to guarantee finite parses.

Proviso

Because of the above tests it is now reasonable to assume that for any grammars to be dealt with in the sequel, that the parses terminate with probability one, that $r<1$, that $F_i(1,\ldots,1) = 1$ and so on. Cases when $r=1$ will not be considered.

2.3 THE LENGTH GENERATING FUNCTION

The length generating function of a grammar is defined in this section, and some useful properties of it demonstrated. In general it is impossible to give an extrinsic expression for this function, so its properties must be found by indirect proof or by manipulating an intrinsic equation which it satisfies. However the function is well behaved, and it is possible to roughly locate its singularities. Various parameters of the original grammar can be deduced from those of the function, and even calculated by simple arithmetic.

2.3.1 Definitions for the length generating function

If necessary, the reader should refer back to section 2.2.1 to recall the notation.

Definition

$\Omega_i^{(n)}$ is that subset of Ω_i which contains only those parses with exactly n terminal symbols.

Definition

$$p_i^{(n)} = \sum_{\omega \in \Omega_i^{(n)}} M_i(\omega)$$

The sequence $\overset{\infty}{n=0}\langle p_i^{(n)} \rangle$ is called the length sequence for X_i.

Remark

$p_i^{(n)}$ is the probability that a parse will contain exactly n terminal symbols (and when the infinite parses have zero probability, also the probability that the generated string will have length n). Hence $p_i^{(n)}$ is a non-negative real number and so the sequence $\langle p_i^{(n)} \rangle$ can be used to define a generating function.

Definition

$$f_i(z) = \sum_{n=0}^{\infty} p_i^{(n)} z^n$$

$f_i(z)$ is called the length generating function (for X_i).

2.3.2 The intrinsic equation for the length generating function

Theorem

If the BNF of a grammar is

$$X_i = \sum_j p_{ij} \prod_k X_{ijk}$$

then the length generating functions obey the equations

$$f_i(z) = \sum_j p_{ij} \prod_k f_{ijk}(z)$$

where the second equation means that if one side converges then so does the other, and both then have equal values.

Proof: As usual this is in two parts:

1. $f_i(z) = \sum_j p_{ij} f_{ij}(z)$

If $\omega \in \Omega_i^{(n)}$ then it is a parse with start symbol X_i and n terminal symbols. Some production, P_{ij} say, must have been used first to generate ω and so $\omega \in \Omega_{ij}^{(n)}$ also. Now $\mu_i(\omega) = p_{ij} \mu_{ij}(\omega)$ Hence summing over all ω in $\Omega_i^{(n)}$

$\mu_i(\Omega_i^{(n)}) = \sum_j p_{ij} \mu_{ij}(\Omega_{ij}^{(n)})$, in other words $p_i^{(n)} = \sum_j p_{ij} p_{ij}^{(n)}$.

Hence $\sum_n p_i^{(n)} z^n = \sum_n \sum_j p_{ij} p_{ij}^{(n)} z^n = \sum_j p_{ij} \sum_n p_{ij}^{(n)} z^n$

the last step being permissible because $\sum_n p_{ij}^{(n)} z^n$ converges absolutely. So the result follows from the facts that $f_i(z) = \sum_n p_i^{(n)} z^n$ and similarly $f_{ij}(z) = \sum_n p_{ij}^{(n)} z^n$.

2. $f_{ij}(z) = \prod_k f_{ijk}(z)$

This is more difficult to prove, and some special definitions and notations are needed which are used only in this proof. The sequence of spaces

$$\langle \Theta_k, \mathcal{B}_k, \nu_k \rangle = \prod_{h=1}^{k} \langle \Omega_{ijh}, \mathcal{A}_{ijh}, \mu_{ijh} \rangle \ldots \quad (k=1,\ldots,n_{ij})$$

can easily be inductively defined. If the number $l(\theta)$ of terminals in a k-tuple $\theta = \langle \omega_1, \ldots, \omega_k \rangle$ is defined to be $\sum_{n=1}^{k} l(\omega_n)$, that is the sum of the numbers of terminals in the individual elements of the tuple, then $\theta_k^{(n)}$, $q_k^{(n)}$ and $g_k(z)$ can be defined analagously to $\Omega_i^{(n)}$, $p_i^{(n)}$ and $f_i(z)$.

Thus, $\theta_k^{(n)} = \{\theta : \theta \in \Theta_k \text{ and } l(\theta) = n\}$

$$q_k^{(n)} = \nu_k(\theta_k^{(n)})$$

$$g_k(z) = \sum_{n=0} q_k^{(n)} z^n$$

The proof that $f_{ij}(z) = \prod_k f_{ijk}(z)$ now works by showing that

$$g_{k+1}(z) = f_{ij(k+1)}(z)\, g_k(z)$$

and then using induction.

Any (k+1)-tuple $\theta = \langle \omega_1, \ldots, \omega_k, \omega_{k+1} \rangle \in \Theta_{k+1}$ may be written as a product of the k-tuple $\theta' = \langle \omega_1, \ldots, \omega_k \rangle \in \Theta_k$ and the parse $\omega_{k+1} \in \Omega_{ij(k+1)}$. Clearly $l(\theta) = l(\theta') + l(\omega_{k+1})$. Hence if $\theta' \in \theta_k^{(n-h)}$ and $\omega_{k+1} \in \Omega_{ij(k+1)}^{(h)}$ then $\theta \in \theta_{k+1}^{(n)}$, and clearly every $\theta \in \theta_k^{(n)}$ can be so obtained as h varies from 0 to n. Hence,

$$\theta_{k+1}^{(n)} = \bigcup_{h=0}^{n} \theta_k^{(n-h)} \boxtimes \Omega_{ij(k+1)}^{(h)}$$

Hence by definition of a product measure,

$$\nu_{k+1}(\theta_{k+1}^{(n)}) = \sum_{h=0}^{n} \nu_k(\theta_k^{(n-h)})\, \mu_{ij(k+1)}(\Omega_{ij(k+1)}^{(h)}),$$

so using the definitions of $p_k^{(n)}$ and $q_k^{(n)}$

$$q_{k+1}^{(n)} = \sum_{h=o}^{n} q_k^{(n-h)} \, p_{ij(k+1)}$$

Hence $g_{k+1}(z) \stackrel{def}{=} \sum_{n=o}^{\infty} q_{k+1}^{(n)}$

$$= \sum_{n=o}^{\infty} \sum_{h=o}^{n} q_k^{(n-h)} \, z^{n-h} \, p_{ij(k+1)}^{(h)} \, z^h$$

$$= \sum_{n=o}^{\infty} \sum_{l+h=n} q^{(l)} \, z^l \, p_{ij(k+1)}^{(h)} \, z^h \qquad \text{(by putting } l=n-h\text{)}$$

$$= \sum_{l=o}^{\infty} q_k^{(l)} \, z^l \sum_{h=o}^{\infty} p_{ij(k+1)}^{(h)} \, z^h \qquad \text{(by rearranging the sum which is permissable due to absolute convergence)}$$

$$\stackrel{def}{=} g_k(z) \, f_{ij(k+1)}(z)$$

Clearly $g_1(z) = f_{ij1}(z)$, hence by induction $g_{n_{ij}}(z) = \prod_{k=1}^{n_{ij}} f_{ijk}(z)$.

But theorem 1.5.4 shows that $\langle \Theta_{n_{ij}}, \mathcal{B}_{n_{ij}}, \mathcal{V}_{n_{ij}} \rangle$ is isomorphic to $\langle \Omega_{ij}, \mathcal{A}_{ij}, \mathcal{M}_{ij} \rangle$ and hence

$$f_{ij}(z) = g_{n_{ij}}(z) = \prod_{k=1}^{n_{ij}} f_{ijk}(z).$$

Corollary

$$f_i(z) = F_i(z, f_1(z), \ldots, f_n(z)) \qquad (i=1,\ldots,n)$$

Proof: $F_i(z_o, \ldots, z_n) = \sum_j p_{ij} \prod_k z_{ijk}$.

Remark

$f_o(z) \equiv z$ and $F_o(z_o, \ldots, z_n) \equiv 1$, so the above corollary does not hold for i=0. This is because the left hand side of the above equation is the generating function for the number of terminals appearing at depth 0 or greater in a parse starting from X_i, and the right hand side gives the number appearing at depth 1 or greater.

These numbers are the same except when the start symbol is a terminal, that is i=0.

Theorem

In a reduced grammar, $f_i(1) = T_i$.

Proof: The probability that an infinite parse generates a finite number of terminals is zero in a reduced grammar, so both sides of the above equation give the measure of the set of finite parses.

Corollary

If $r<1$ where r is the dominant eigenvalue of M, the matrix of means, then $f_i(1) = 1$.

2.3.3 The differential coefficients of the length generating function

Definition

An expression $\mathcal{E}$ will be said to have property $\mathcal{P}_N$ if $\mathcal{E}$ is a multinomial in terms $\frac{\partial^s F_k}{\partial z_{i_1} \cdots z_{i_s}}$ and $\frac{d^t f_i}{dz^t}$,

where $1 \leq s,t$; $0 \leq j,k \leq n$; $0 \leq i_u \leq n$ for $1 \leq u \leq s$;

and N-1 is the maximum value of t for the terms $\frac{d^t f_i}{dz^t}$ that occur in $\mathcal{E}$.

Theorem

If $r<1$ and hence all the measure is concentrated on the finite parses then all the differential coefficients $(\frac{d}{dz})^N f_i$ are finite at $z=1$.

Proof: First it will be shown by induction that for $N \geq 1$

$$\frac{d^N f_i}{dz^N} = \sum_{j=1}^{n} \frac{\partial F_i}{\partial z_j} \frac{d^N f_j}{dz^N} + \mathcal{E}_N^i \qquad \text{(where } \mathcal{E}_N^i \text{ has property } \mathcal{P}_N\text{)}$$

By differentiating equation 2.3.2.2 there is obtained

$$\frac{df_i}{dz} = \sum_{j=1}^{n} \frac{\partial F_i}{\partial z_j} \frac{df_i}{dz} + \mathcal{E}_1^i \qquad \left(\text{where } \mathcal{E}_1^i = \frac{\partial F_i}{\partial z_o}\right)$$

Clearly $\frac{\partial F_i}{\partial z_o}$ has property $\mathcal{P}_1$ so the statement holds for N=1.

By differentiating the statement for N is obtained

$$\frac{d^{N+1}f_i}{dz^{N+1}} = \sum_{j=1}^{n} \frac{\partial F_i}{\partial z_j} \frac{d^{N+1}f_j}{dz^{N+1}} + \mathcal{E}_{N+1}^i$$

where $\mathcal{E}_{N+1}^i = \sum_{j=1}^{n} \sum_{k=0}^{n} \frac{\partial^2 F_i}{\partial z_j \partial z_k} \frac{d^N f_j}{dz^N} + \frac{d\mathcal{E}_N^i}{dz}$

The first part of the right hand side of this equation clearly has property $\mathcal{P}_{N+1}$. The second summand can be seen, by using the method of differentiating a function of a function, to be a multinomial in those terms which appear in $\mathcal{E}_N^i$ and also the possibly new terms

$$\frac{\partial^{s+1}F_k}{\partial z_i, \cdots \partial z_{i_{s+1}}} \times \frac{df_{i_{s+1}}}{dz} \quad \text{and} \quad \frac{d^{t+1}f_j}{dz^{t+1}}$$

where $\frac{\partial^s F_k}{z_i, \ldots, z_{i_s}}$ and $\frac{d^t f_j}{dz^t}$ occur in $\mathcal{E}_N^i$.

$\mathcal{E}_N^i$ has property $\mathcal{P}_N$ by the inductive hypothesis, so the maximum value of t is N-1 and hence of t+1 is N, that is $\frac{d\mathcal{E}_N^i}{dz}$ has property $\mathcal{P}_{N+1}$. Hence $\mathcal{E}_{N+1}^i$ also has property $\mathcal{P}_{N+1}$ and equation 2.3.3.2 holds by induction.

Another induction is needed to show that all $\left.\frac{d^N f_i}{dz^N}\right|_{z=1}$ are finite. Assume that all are up to $N=N_o$.

Now $\left.\frac{\partial^s F_k}{\partial z_{i_1} \cdots z_{i_s}}\right|_{z=1}$ is a multinomial in $z_o, \ldots, z_n$ evaluated at $z_o=z=1$ and $z_i=f_i(z)=1$, and so is finite for all s, k and sequences $i_1, \ldots, i_s$. Similarly $\mathcal{E}_{N_o+1}^i\big|_{z=1}$ is a multinomial in the finite numbers $\left.\frac{\partial^s F_k}{\partial z_{i_1} \cdots z_{i_s}}\right|_{z=1}$ and the numbers $\left.\frac{d^N f_i}{dz^N}\right|_{z=1}$ which

are assumed finite for $N \leqslant N_o$ by the induction hypothesis. Hence

$\mathcal{E}^{\,i}_{N_o+1}\Big|_{z=1}$ is also finite. Now equation 2.2.3.2 for $N=N_o+1$

can easily be transformed to

$$\sum_{j=1}^{n} \left(\delta_{ij} - \frac{\partial F_i}{\partial z_j}\right) \frac{d^{N_o+1} f_j}{dz^{N_o+1}} = \mathcal{E}^{\,i}_{N_o+1} \quad (\text{where } \delta_{ij}=1 \text{ if } i=j, \text{ 0 otherwise})$$

And evaluating at $z=1$

$$\sum_{j=1}^{n} \left(\delta_{ij} - M_{ij}\right) \frac{d^{N_o+1} f_j}{dz^{N_o+1}}\Big|_{z=1} = \mathcal{E}^{\,i}_{N_o+1}\Big|_{z=1}$$

where $M_{ij} = \dfrac{\partial F_i}{\partial z_j}\Big|_{z=1}$ is an element of M the matrix of means.

Now the dominant eigenvalue of M is less than 1, so I-M is invertible, and so multiplying both sides of the above equation by the inverse of I-M,

$$\frac{d^{N_o+1} f_j}{dz^{N_o+1}}\Big|_{z=1} \quad \text{is finite.}$$

The base of the second induction is trivial so the proof is complete.

Corollary

The numbers $\left(\frac{d}{dz}\right)^N f_i\Big|_{z=1}$ can be calculated directly without knowing the functions f_i.

Proof: $f_i(1) = 1$ is known, so $\mathcal{E}^{\,i}_{1}$ can be calculated, then because I-M is known $\frac{d}{dz} f_i\Big|_{z=1}$ can be calculated and so on.

Corollary

None of the functions f_i have a pole or an algebraic branch point at $z=1$.

Proof: $f_i(1) = 1$ so there is no pole. If $z=1$ were a branch point then f_i could be expressed in some neighbourhood of $z=1$ as

$$f_i(z) = \sum_{j=\alpha}^{\infty} a_j (z-1)^{j/H}$$

where H is some positive integer, α may be either positive or negative, and there is some j such that H does not divide j and $a_j \neq 0$. [AHLFORS, p.294] Hence some differential coefficient is infinite.

Remark

This second corollary is the main reason for proving the previous theorem. Eventually it will be proved that f_i has no singularities of any kind within some open disk of radius $1+\epsilon$ about the origin, where $\epsilon > 0$.

Theorem

The mean N^{th} power of the number of terminal symbols in a word generated by symbol X_i is finite and finitely calculable for all N and i.

Proof: The sequence of functions h_i^N can be defined inductively by

$$h_i^0(z) \equiv f_i(z)$$

$$h_i^{N+1}(z) \equiv z \frac{dh_i^N}{dz}$$

Using a proof just like that of the previous theorem it can be deduced that all the $h_i^N(1)$ are finite and can be calculated without knowing the functions h_i^N directly. It is also easy to show inductively that

$$h_i^N(z) = \sum_{j=1}^{\infty} j^N p_i^{(j)} z^j \qquad \ldots |z| < 1, N \geq 0, i=1,\ldots,n$$

so that $h_i^N(1) = \sum_{j=1}^{\infty} j^N p_i(j)$, which is just the expression for the mean N^{th} power of the number of terminal letters in a word

generated by X_i.

2.3.4 The Radius of Convergence of the Length Generating Function

Theorem

Each function f_i is algebraic.

Proof: $f_i - F_i(z,f_1,\ldots,f_n) \equiv 0$ for all z, $i=1,\ldots,n$. By corollary 2.2.7.2 , $M-I \neq 0$, but this is the jacobian of the above at $1,\ldots,1$. Hence theorem 2.1.5.4 applies.

Corollary

The only singularities of f_i are poles and branch points.

Proof: Theorem 2.1.5.2.

Theorem

If R_i is the radius of convergence of f_i, then $R_i > 1$.

Proof: First $R_i \geq 1$. This is because for $|z| \leq 1$, the sum $\sum_{n=0}^{\infty} p_i^{(n)} z^n$ is dominated by the sum of positive terms $\sum_{n=0}^{\infty} p_i^{(n)}$, which converges to the probability that a parse generates finitely many terminals, that is somenumber P such that $0 \leq P \leq 1$. Hence $f_i(z)$ is defined and finite for all $z \leq 1$ so $R_i \geq 1$.

Second, $R_i \neq 1$. If $R_i = 1$ then as all the terms $p_i^{(n)}$ are non-negative $z_o=1$ is a singular point of f_i [HILLE 5.7.1, p.133]. As f_i is algebraic z_o can only be a branch point or pole, by corollary 2.3.3.4 it is neither. Hence $R_i \neq 1$.

Thus the only possibility left is that $R_i > 1$.

Corollary

The radius of convergence $R_i = 1+\epsilon$, and for $|z|<1+\epsilon$, z is not a singular point of f_i.

2.3.5 Example - The Length Generating Function for the Language of Assignment Statements

The grammar of this language is

$$S::= \{p_1\}\ L=R$$

$$L::= \{p_{21}\}\ a \mid \{p_{22}\}\ b$$

$$R::= \{p_{31}\}\ (R+R) \mid \{p_{32}\}(R\times R) \mid \{p_{33}\}a \mid \{p_{34}\}b$$

where $p_1 = p_{21}+ p_{22} = p_{31}+ p_{32}+ p_{33}+ p_{34} = 1.$ $p_{31}+p_{32} = q.$

The multinomials F_1, F_2, F_3 may be found by substituting z_o for terminals, z_1 for S, z_2 for L and z_3 for R to give

$$F_1(z_0,z_1,z_2,z_3) \equiv p_1 z_2 z_0 z_3 \equiv z_0 z_2 z_3$$

$$F_2(z_0,z_1,z_2,z_3) \equiv p_{21} z_0 + p_{22} z_0 \equiv z_0$$

$$F_3(z_0,z_1,z_2,z_3) \equiv q(z_0 z_3 z_0 z_3 z_0) + (1-q)z_0 \equiv q z_0^3 z_3^2 + (1-q)z_0$$

The matrix of means can be calculated to be

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 3q+(1-q) & 0 & 0 & 2q \end{pmatrix}$$

This has the eigenvalues $0, 0, 0, 2q$ and eigenvectors

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ -(2q-3) \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 1+2q \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Clearly 2q is the dominant eigenvalue amongst these, and so the measure is concentrated on the finite parses and all the theorems work so long as $q < \frac{1}{2}$.

The equations for the length generating functions may be written

$$f_1(z) = z f_2(z) f_3(z)$$

$$f_2(z) = z$$

$$f_3(z) = q\, z^3 (f_3(z))^2 + (1-q)z.$$

The second and third of these equations give directly that f_2 and f_3 are algebraic, the first gives f_1 as a rational combination of algebraic functions and hence also algebraic.

The third equation can be rearranged as a quadratic in f_3, that is $a(f_3(z))^2 + b\, f_3(z) + c = 0$ where $a = qz^3$, $b = -1$ and $c = (1-q)z$. f_3 can only have a pole when $a=0$, that is when $z=0$. But the branch defined by the series has $f_3(0)=0$ and so no pole there. f_3 has a branch point when the quadratic has a double root, that is when the discriminant $b^2-4ac=0$. Evaluating, when

$$1 - 4.qz^3(1-q)z = 0.$$

So there can only be branch points when $z = \sqrt[4]{(4q(1-q))^{-1}}$. These points are all outside the unit disk so long as $(4q(1-q))^{-1}>1$, in other words so long as $0<(2q-1)^2$, that is so long as $q \neq \frac{1}{2}$. If all the measure is concentrated on the finite parses, then $q<\frac{1}{2}$, so $q \neq \frac{1}{2}$, so the radius of convergence $R_3>1$.

Of the singularities of f_1 and f_2, f_2 has no singularities, and f_1 can only have singularities where f_3 does.

2.4 THE ENTROPY GENERATING FUNCTION

The entropy of a grammar was defined in the footnote on p23, and a method given for calculating it in chapter 1 (theorem 1.5.23). So if the only use of the entropy generating function were to calculate the entropy, then this section would be superfluous. This is not the case, because some properties of the entropy generating function are needed to calculate the information rate of a grammar.

2.4.1 Definitions for the Entropy Generating Function

Definition

$$q_i^{(n)} = -\sum_{\omega \in \Omega_i^{(n)}} \mu_i(\omega) \log \mu_i(\omega)$$

The sequence $\underset{n=0}{\overset{\infty}{}}\langle q_i^{(n)} \rangle$ is called the entropy sequence for X_i.

Remark

$q_i^{(n)}$ is the contribution by words of length n to the total entropy E_i. That is $E_i = \sum_{n=1}^{\infty} q_i^{(n)}$. Similarly $E_{ij} = \sum_{n=1}^{\infty} q_{ij}^{(n)}$ and $E_{ijk} = \sum_{n=1}^{\infty} q_{ijk}^{(n)}$.

Definition

$g_i(z) = \sum_{n=1}^{\infty} q_i^{(n)} z^n$ and is called the entropy generating function for X_i.

Remark

$\mu_i(\omega)$ is positive and $\log \mu_i(\omega)$ negative (because $\mu_i(\omega) < 1$), so every $q_i^{(n)} \geqslant 0$ and hence also $E_i \geqslant 0$. If E_i is finite then the series for $g_i(1)$ converges absolutely, hence also that for $g_i(z)$ for $z \leqslant 1$.

In the case i=0, there is only one ω in $\Omega_i = \Omega_i^{(1)}$, so

$$\sum_{\omega \in \Omega_i^{(n)}} \mu_i(\omega) \log \mu_i(\omega) = 0$$

vacuously for $n > 1$, and for n=1 the sum is also zero but by calculation. Hence $g_0(z) \equiv 0$.

2.4.2 The Intrinsic Equation for the Entropy Generating Function

Theorem

If the BNF of a grammar is

$$X_i = \sum_j p_{ij} \prod_k X_{ijk}$$

then the entropy generating functions g_i, g_{ij}, g_{ijk} obey the equations

$$g_i(z) = -\sum_{j=1}^{n_i} p_{ij}(\log p_{ij})\, f_{ij}(z) + \sum_{j=1}^{n_i} p_{ij}\Big(\sum_{k=1}^{n_{ij}} g_{ijk}(z) \prod_{l=1,\hat{k}}^{n_{ij}} f_{ijl}(z)\Big)$$

where $f_{ij}(z)$, $f_{ijk}(z)$ are length generating functions, and the symbol $\prod_{l=1,\hat{k}}^{n_{ij}}$ means the product where l varies from 1 to n_{ij} missing out k.

Proof: This has the usual two parts:

1) $$g_i(z) = -\sum_{j=1}^{n_i} p_{ij}(\log p_{ij}) f_{ij}(z) + \sum_{j=1}^{n_i} p_{ij} g_{ij}(z).$$

For by expanding the definition of g_i,

$$g_i(z) = -\sum_{n=1}^{\infty}\Big(\sum_{\omega \in \Omega_i^{(n)}} \mu_i(\omega) \log \mu_i(\omega)\, z^n\Big).$$

Now $\Omega_i^{(n)} = \bigcup_{j=1}^{n_i} \Omega_{ij}^{(n)}$ and $\mu_i(\omega) = p_{ij}\mu_{ij}(\omega)$ if $\omega \in \Omega_{ij}$

So $$g_i(z) = -\sum_{n=1}^{\infty} \sum_{j=1}^{n_i} \sum_{\omega \in \Omega_{ij}^{(n)}} \Big(p_{ij}\mu_{ij}(\omega) \log(p_{ij}\mu_{ij}(\omega)) z^n\Big)$$

So using the fact that $\log(p_{ij}\mu_{ij}(\omega)) = \log p_{ij} + \log \mu_{ij}(\omega)$ there follows

$$g_i(z) = -\sum_{n=1}^{\infty} \sum_{j=1}^{n_i} \sum_{\omega \in \Omega_{ij}^{(n)}} p_{ij}(\log p_{ij})\mu_{ij}(\omega) z^n$$

$$- \sum_{n=1}^{\infty} \sum_{j=1}^{n_i} \sum_{\omega \in \Omega_{ij}^{(n)}} p_{ij}\mu_{ij}(\omega) \log \mu_{ij}(\omega) z^n .$$

Part 1 of the theorem now follows by rearranging the $\sum$'s and using the facts that

$$f_{ij}(z) = \sum_n \sum_\omega \mu_{ij}(\omega) z^n \quad \text{and that} \quad g_{ij}(z) = -\sum_n \sum_\omega \mu_{ij}(\omega) \log \mu_{ij}(\omega) z^n.$$

2) $$g_{ij}(z) = \sum_{k=1}^{n_{ij}} g_{ijk}(z) \sum_{=1,\hat{k}}^{n_{ij}} f_{ijk}(z)$$

The proof of this second part requires an induction rather like that in the second part of theorem 2.3.2.1 . This induction requires a lot of machinery, so to save the effort in setting it up, the second part will only here be proved for the case $n_{ij} = 2$. This case is more general than it appears at first sight because it is also the central step in the proof of the general induction.

In the case $n_{ij} = 2$, the equation to be proved may be simply written

$$g_{ij}(z) = g_{ij1}(z) f_{ij2}(z) + f_{ij1}(z)\, g_{ij2}(z)$$

If $\omega \in \Omega_{ij}$ and $n_{ij} = 2$ then the parse ω must stem from the production $\langle X_i \rightarrow X_{ij1} X_{ij2} \rangle$ and the symbols X_{ij1} and X_{ij2} must give rise to subparses $\omega_1 \in \Omega_{ij1}$ and $\omega_2 \in \Omega_{ij2}$ respectively. If $\omega \in \Omega_{ij}^{(n)}$ so that ω has n terminals, then ω_1 and ω_2 have n between them, so if $\omega_1 \in \Omega_{ij1}^{(m)}$ then $\omega_2 \in \Omega_{ij2}^{(n-m)}$. Finally $\mu_{ij}(\omega) = \mu_{ij1}(\omega_1)\, \mu_{ij2}(\omega_2)$. So

$$q_{ij}^{(n)} = - \sum_{\omega \in \Omega_{ij}^{(n)}} \mu_{ij}(\omega) \log \mu_{ij}(\omega)$$

$$= - \sum_{m=1}^{n} \sum_{\substack{\omega_1 \in \Omega_{ij1}^{(m)} \\ \omega_2 \in \Omega_{ij2}^{(n-m)}}} \mu_{ij1}(\omega_1) \mu_{ij2}(\omega_2) \log \left(\mu_{ij1}(\omega_1) \mu_{ij2}(\omega_2) \right)$$

The log in the RHS can be split in the usual way to give

$$q_{ij}^{(n)} = - \sum_{m=0}^{n} \left(\sum_{\omega_1 \in \Omega_{ij1}^{(m)}} \mu_{ij1}(\omega_1) \right) \times \left(\sum_{\omega_2 \in \Omega_{ij2}^{(n-m)}} \mu_{ij2}(\omega_2) \log \mu_{ij2}(\omega_2) \right)$$

$$- \sum_{m=0}^{n} \left(\sum_{\omega_2 \in \Omega_{ij2}^{(n-m)}} \mu_{ij2}(\omega_2) \right) \times \left(\sum_{\omega_1 \in \Omega_{ij1}^{(m)}} \mu_{ij1}(\omega_1) \log \mu_{ij1}(\omega_1) \right)$$

That is

$$q_{ij}^{(n)} = \sum_{m=0}^{n} p_{ij1}^{(m)} \; q_{ij2}^{(n-m)} + \sum_{m=0}^{n} p_{ij2}^{(n-m)} \; q_{ij1}^{(m)}$$

So multiplying both sides of the above by z^n and summing over n

$$\sum_{n=1}^{\infty} q_{ij}^{(n)} \, z^n = \sum_{n=1} \sum_{m=0}^{n} p_{ij1}^{(m)} \, z^m \, q_{ij2}^{(n-m)} \, z^{n-m} + \sum_{n=1} \sum_{m=0}^{n} p_{ij2}^{(n-m)} z^{n-m} q_{ij1}^{(n)} \, z^n$$

The right hand side of this equation is the sum of two convolutions, when rewritten as functions the expressions become

$$g_{ij}(z) = f_{ij1}(z) \; g_{ij2}(z) + f_{ij2}(z) \; g_{ij1}(z),$$

that is the simplified form of equation 2.

The proof is terminated by substituting the value obtained for $g_{ij}(z)$ from equation 2 into equation 1.

Corollary

The result of the previous theorem may be rewritten

$$g_i(z) = - \sum_{j=1}^{n_i} p_{ij} \log p_{ij} \; f_{ij}(z) + \sum_{k=1}^{n} \frac{\partial F_i}{\partial z_k} \; x \; g_k(z)$$

Proof: This may be seen by inspecting the form of the multinomial $F_i(z_o,\ldots,z_n)$.

2.4.3 Properties of the Entropy Generating Function

Theorem

All the functions g_i are algebraic.

Proof: The dominant eigenvalue of $\left.\frac{\partial F_i}{\partial z_j}\right|_{\langle 1,\ldots 1\rangle} < 1$ by proviso.

So the matrix $\left(\delta_{ij} - \left.\frac{\partial F_i}{\partial z_j}\right|_{\langle 1,\ldots,1\rangle}\right)$ is non-singular and invertible.

$\mathrm{Det}\left(\delta_{ij} - \left.\frac{\partial F_i}{\partial z_j}\right|_{\langle z,f_1(z),\ldots,f_n(z)\rangle}\right)$ is a continuous function of z

and so not zero in an open neighbourhood of $z=1$. An invertible matrix can be inverted by rational operations alone, so

$$\left(\delta_{ij} - \frac{\partial F_i}{\partial z_j}\bigg|_{\langle z, f_1(z) \ldots f_n(z)\rangle} \right)$$

can be inverted to give a proper inverse matrix R_{ij} whose terms are all rational complexes of the f_i. Rearranging the equation of corollary 2.4.2.2 and multiplying both sides by R_{ik} yields

$$g_i(z) = \sum_{k=1}^{n} R_{ik} \left(-\sum_{j=1}^{n_k} p_{kj} \log p_{kj} f_{kj}(z) \right).$$

The above gives g_i as a rational complex of the algebraic functions f_i, so g_i is algebraic.

Corollary

E_i, the entropy of the language stemming from X_i, is finite.

Proof: As g_i is a rational complex of the f_i, it can only have a branch point where one at least of them has one. So it has none inside or on the unit circle. It can only have a pole where some $\sum p_{kj} \log p_{kj} f_{kj}(z)$ has a pole, again only outside the unit circle, or where

$$\det \left(\delta_{ij} - \frac{\partial F_i}{\partial z_j}\bigg|_{\langle z, f_1(z) \ldots f_n(z)\rangle} \right) = 0.$$

Now each element of

$$\frac{\partial F_i}{\partial z_j}\bigg|_{\langle z, \ldots, f_n(z)\rangle}$$

is zero at $z=0$ and steadily increasing as z increases to 1. They are also positive. Hence the dominant eigenvalue of $\frac{\partial F_i}{\partial z_j}$ is steadily increasing also. Hence $\det \left(\delta_{ij} - \frac{\partial F_i}{\partial z_j} \right) \neq 0$ for $0 \leqslant z \leqslant 1$, so g_i has no poles inside or on the unit circle, so the sum for g_i converges for $z=1$, and its value is the finite number

given by

$$g_i(1) = \sum_{k=1}^{n} R_{ik}\Big|_{z=1} (- \sum_{j=1}^{n_k} p_{kj} \log p_{kj} f_{kj}(z)).$$

2.4.4 Example of the Entropy Generating Function

Once again the language of assignment statements will be used as an example (see section 2.3.5). f_i will be written as short for $f_i(z)$, g_i for $g_i(z)$ and so on, also q_{ij} instead of $p_{ij} \log p_{ij}$.

The equations for the entropy generating functions are:

$$g_1 = - q_1 f_{11} + p_1(g_1 f_0 f_3 + f_1 g_0 f_3 + f_1 f_0 g_3)$$

$$g_2 = - q_{21} f_{21} - q_{22} f_{22} + p_{21} g_0 + p_{22} g_0$$

$$g_3 = - q_{31} f_{31} - q_{32} f_{32} - q_{33} f_{33} - q_{34} f_{34}$$
$$+ (p_{31} + p_{32})(g_0 f_3 f_0 f_3 f_0 + f_0 g_3 f_0 f_3 f_0 + f_0 f_3 g_0 f_3 f_0$$
$$+ f_0 f_3 f_0 g_3 f_0 + f_0 f_3 f_0 f_3 g_0)$$
$$+ (p_{33} + p_{34}) g_0$$

The above can be simplified using the general facts true for all grammars:- $f_{ij} = \prod f_{ijk}$, $f_0(z) = z$, $g_0(z) = 0$; and the particular facts true for this grammar:- $p_1 = 1$, $f_1 = z^3 f_3$, $f_2 = z$, and $q_4 = 0$; to give

$$g_1 = g_1 z f_3 + z^3 f_3 g_3$$

$$g_2 = (q_{21} + q_{22}) z$$

$$g_3 = (q_{31} + q_{32}) z^3 f_3^2 - (q_{33} + q_{34}) z + (p_{31} + p_{32})\, 2 z^3 f_3 g_3$$

which can be solved to give

$$g_3 = -[(q_{31} + q_{32}) z^2 f_3^2 + (q_{33} + q_{34}) z] / [1 - 2(p_{31} + p_{32}) z^3 f_3]$$

$$g_1 = z^3 f_3 g_3 / [1 - z f_3].$$

Chapter 3

THE INFORMATION RATE OF A CONTEXT FREE GRAMMAR

Communication engineers talk about the rate of transfer of information by languages, but can calculate this parameter only for very simple languages, which they call stochastic and which are in effect the Chomsky type 3 or linear languages. Linguists are interested in more complicated languages, context-free languages being among the simplest. This section achieves one of the main goals of this thesis by showing how to calculate the rate of context-free languages.

There are two pieces of earlier work along these lines. Firstly a paper by K. WALK [34] discussed the general problem of finding the rate and showed how the results of information engineers allowed it to be calculated for linear grammars. Secondly KUICH [22] found a single rate for a context-free language. In effect he allowed one probability distribution, that in which all strings of length N are equally likely.

This section shows how to define and calculate the rate for any distribution generated by a preprobability. Kuich's distribution is often one of the many which cannot be so generated.

3.1 FINITE AND INFINITE CLOSURES OF A GRAMMAR

3.1.1 Definition

If L is any language then L^+, the set of all finite concatenation of one or more elements of L, is called the finite closure of L.

3.1.2 Remark

The empty string ϵ is only in L^+ if it is also in L. So the

closures considered here will not contain the empty string.

3.1.3 Definition

If L is any language, then L^c, the set of all infinite strings consisting of concatenations of elements of L, is called the infinite closure of L.

3.1.4 Theorem

If L is context free then L^+ is also context free. L^c is not context free.

Proof: If the grammar for L is $G = \langle N,T,P,S\rangle$ and S^+ is a new non-terminal symbol not contained in N or T, then a grammar for L^+ is $G^+ = \langle Nu\{S^+\},T,Pu\{\langle S^+\rightarrow S\rangle, \langle S^+\rightarrow SS^+\rangle\},S^+\rangle$. L^c is not context free because by definition context free languages contain only finite strings.

3.1.5 Remark

Although L^c is not context free it may be thought of as having the improper grammar $G^c = \langle Nu\{S^c\},T,Pu\{\langle S^c\rightarrow SS^c\rangle\},S^c\rangle$. Clearly this grammar can produce (amongst others) all non-terminal strings of the form $w_1 w_2 \ldots w_n S^c$ where each $w_i \in L$.*

3.1.6 Definition

A grammar G is said to have unambiguous closure if G^+ is unambiguous (where G^+ is as defined in 3.1.10)

3.1.7 Remark

It is possible for a grammar to be itself unambiguous but have ambiguous closure. A trivial example is the grammar with BNF $S\rightarrow a|Sa$ which unambiguously generates strings of the form a^m. Its closure ambiguously generates the same set because it can generate a^m as any product $a^{m1}\, a^{m2} \ldots a^{mn}$ where $\Sigma m_i = m$.

* Note. In what follows we are concerned with the infinite strings generated by grammars as well as with the finite strings normally thought of as constituting their language.

3.1.8 Proviso

In what follows it will be assumed that all grammars as well as being themselves unambiguous also have unambiguous closure.

3.1.9 Remark

The above standard definitions can easily be extended to cover probabilistic context-free grammars.

3.1.10 Definition

If G is a probabilistic context-free grammar with start symbol S, then a finite closure G^+ is any probabilistic grammar with a new start symbol S^+ and the two additional probabilistic productions

$$S^+ \rightarrow \{p_1\}\ SS^+$$

$$S^+ \rightarrow \{p_2\}\ S \qquad \text{where } p_1 + p_2 = 1,\ 0 < p_1 < 1.$$

3.1.11 Definition

If G is as above, then its infinite closure G^c is the improper grammar with start symbol S^c and the two extra productions

$$S^c \rightarrow \{1\}\ SS^c$$

$$S^c \rightarrow \{0\}\ S$$

Note G^+ and G^c generate the same (finite & infinite) strings but with different measures.

3.1.12 Notation

If X is some notation referring to a grammar then X^+ will stand for the corresponding concept for its finite closure and X^c for that of the infinite closure. There is in fact an infinite spectrum of finite closures, and this will always be parametrised by p_1. Care must be exercised because the concepts named by X^+ and X^c may be exceptional or non-existent. For instance the finite closure of a finite closure is always an ambiguous grammar and the length generating function of the infinite closure would have all zero terms.

3.1.13 Remark

The probability distribution generated by a proper probabilistic context-free grammar could have been defined in two ways. Either bottom up, that is by first defining the probability of each terminal string and then defining the measures of sets to be the sums of the probabilities of their elements. Or as in this thesis, top down, that is by starting by defining the probabilities of sets stemming from partial parses. The bottom up method will not work for sets containing infinite strings because in general an individual infinite string has probability zero. There is no difficulty about the top down method.

Thus in contradistinction to non-probabilistic grammars, probabilistic grammars have been allowed to generate both finite and infinite strings indiscriminately. In particular if G is any grammar then both G^+ and G^c have their measures defined on exactly the same terminal set, they are only distinguished by their different probability measures. (This is why the production $\langle S^c \rightarrow \{0\}\ S\rangle$ was added to the infinite closure.)

The infinite closure language is the one of interest for information theory, but it is often difficult to directly calculate its properties. One way to find them is to take the limit as $p_1 \rightarrow 1$ of the corresponding property of the finite closure. But care is needed, it is only sometimes the case that

$$\lim_{p_1 \rightarrow 1} X^+ = X^c$$

(where X stands for the property in question).

Particular examples, both positive and negative, follow.

3.1.14 Theorem

If $r<1$ and hence
$\therefore$ G generates finite parses with probability one then so does G^+.

Proof: The set of eigenvalues of the matrix of means of G^+ is the same as that of G with the addition of p_1. $p_1 < 1$ so the dominant eigenvalue of G^+ is $\lessgtr 1$ simultaneously with that of G.

3.1.15 Corollary

If Ω^f is the set of finite strings generable by G^+, then

$$\lim_{p_1 \to 1} \mu^+(\Omega^f) \neq \mu^c(\Omega^f).$$

Proof: $\mu^c(\Omega^f) = 0$.

3.1.16 Theorem

If Ω^p is the set of words stemming from a single partial parse p then

$$\lim_{p_1 \to 1} \mu^+(\Omega^p) = \mu^c(\Omega^p).$$

Proof: The measure is a finite product of elements of the pre-probability function.

3.1.17 Corollary

If Ω^1 is a finite disjoint union of sets of words stemming from partial parses then

$$\lim_{p_1 \to 1} \mu^+(\Omega^1) = \mu^c(\Omega^c).$$

3.1.18 Corollary

If ω is any sequence of terminals of G, and Ω^ω is the set of all terminal strings generable by G^+ which have ω as initial part, then

$$\lim_{p_1 \to 1} \mu^+(\Omega^\omega) = \mu^c(\Omega^\omega).$$

Proof: Ω^ω can be broken down to a finite disjoint sum of sets generable by partial parses.

3.2 THE GENERATING FUNCTIONS OF THE FINITE CLOSURE GRAMMAR

3.2.1 Remark

At a later stage in this section the limiting behaviours of the series used to define the generating functions of the finite closure grammar are needed. The limiting behaviours can be found by using theorem 2.1.4.1. This theorem can only be applied when its three preliminary conditions hold. The aim of this subsection is to show that for many closure functions the point z_o mentioned in the condition is real and near the unit point $\langle 1,0 \rangle$, and then to confirm that the three conditions do hold.

3.2.2 Notation

The length and entropy generating functions of the finite closure of a context-free grammar are written f^+ and g^+ respectively.

3.2.3 Theorem

If the length and entropy generating functions of a context-free grammar are f and g respectively, then the finite closure generating functions are given by

$$f^+ = p_2 f/(1-p_1 f)$$

and

$$g^+ = -p_2 \log p_2 f/(1-p_1 f) - p_2(p_1 \log p_1 f^2 - g)/(1-p_1 f)^2$$

Proof: The only productions involving S^+ in the closure grammar are $\langle S^+ \rightarrow \{p_1\} S^+ S \rangle$ and $\langle S^+ \rightarrow \{p_2\} S \rangle$. Theorems 2.3.2.1 and 2.4.2.1 tell that f^+ and g^+ obey the intrinsic equations

$$f^+ = p_1 f^+ f + p_2 f$$

$$g^+ = -p_1 \log p_1 f^+ f - p_2 \log p_2 f + p_1 f g^+ + p_1 g f^+ + p_2 g.$$

The theorem can be obtained by a straightforward algebraic manipulation of the above formula. The only step which might present

difficulty is that f^+ is eliminated from the second equation by using its value as given by the first.

3.2.4 Remark

The above theorem gives f^+ and g^+ as (sums of) fractions with $(1-p_1 f(z))$ in the denominator, so both f^+ and g^+ have poles at any point z where $(1-p_1 f(z)) = 0$ unless the corresponding numerators are zero there too. It will be shown in the sequel that neither numerator is zero at a particular such point z_o.

3.2.5 Lemma

There is some real number $u<1$ and some (small) positive ϵ, such that for all p_1 such that $u<p_1<1$ there is exactly one real z_o such that $0 \leq z_o \leq 1+\epsilon$ and $1-p_1 f(z_o) = 0$.

Proof: This can be derived from the facts that $R>1$ (where R is the radius of convergence of f), theorem 2.3.4.3, that f is given by a real positive termed series and so increasing with z for real positive $z<R$, that $f(1) = 1$, corollary 2.3.2.5 and that $p_1<1$.

3.2.6 Remark

Although z_o is the only number which obeys the above condition such that $0<z_o<1+\epsilon$, it is easy to see that in fact $1<z_o$ always. From any given p_1 which obeys the condition the above theorem defines a unique z_o, that is z_o is a function of p_1. p_1 and z_o are related by the equation $1-p_1 f(z_o) = 0$ or in other words $p_1 = 1/f(z_o)$. So a converse holds: p_1 is a function of z_o.

3.2.7 Lemma

The point z_o found above is a zero of $1-p_1 f(z)$ nearest the origin.

Proof: If z_1 is another zero such that $|z_1|<z_o$ then because f is given by a real positive termed series, $f(z_o)>|f(z_1)|$. This contradicts that $f(z_o) = 1/p_1 = f(z_1)$.

3.2.8 Remark

There may be one (or more) other zeros z_i of $(1-p_1f(z))$ such that $|z_i| = z_o$. It can be shown that this can only happen when all the words generated by the original grammar have lengths a multiple of some positive factor $h>2$. From G can be constructed a new probabilistic grammar G' which is in Greibach normal form and generates exactly the same language with the same probability measure as G. From G can be constructed a new grammar G'/h which uses a terminal alphabet of all the n^h h-tuples of terminal symbols of G, and generates with the same probabilities words corresponding in the obvious way with those generated by G. The only common factor of the lengths of the words generated by G'/h is one, and the new expression $1-p_1f(z)$ for G'/h has a single real root nearest the origin. The various parameters for G can be calculated from those for G'/h.

3.2.9 Proviso

Because of the above remark there is no loss in generality in assuming that the real point z_o found above is a single zero of $1-p_1f(z)$, and the zero strictly nearest the origin.

3.2.10 Theorem

There is some (small) positive ϵ' (such that $\epsilon'\leq\epsilon$) and some $u<1$, such that for all p_1 where $u<p_1<1$, the exactly one real $z_o(>1)$ such that $0\leq z_o\leq 1+\epsilon'$ and $1-p_1f(z_o) = 0$ has the properties that z_o is a single pole of f^+ and a double pole of g^+. In both cases z_o

is the pole strictly nearest the origin.

Proof: The part about f^+ is true because $f^+ = p_2 f/(1-p_1 f)$. The numerator is bounded but positive with f inside the circle of convergence of f, and the circle of convergence has radius $R > 1 + \epsilon$.

Similarly the first fraction $-p_2 \log p_2 f/(1-p_1 f)$ in the sum for g^+ has a single pole at z_o, and no others closer to the origin. Only the second fraction remains.

The second fraction $-p_2(p_1 \log p_1 f^2 + g)/(1-p_1 f)^2$ has no poles at any points z, $|z| < z_o$. Firstly because f(z) and g(z) are finite for $z < z_o$ hence the numerator $-p_2(p_1 \log p_1 f^2(z) + g(z))$ is finite. Secondly because $1-p_1 f(z) \neq 0$. (Lemma 3.2.7)

z_o is a zero of $1-p_1 f(z)$ by definition, so the second fraction has a double pole at z_o so long as the numerator is non-zero.

p_1 is a function of z_o (remark 3.2.6) and $p_1 = 1/f(z_o)$. Therefore the denominator may be rewritten

$$-p_2 \left(\frac{1}{f(z_o)} \log \frac{1}{f(z_o)} f^2(z_o) + g(z_o)\right)$$

$$= -p_2 \left(-f(z_o) \log f(z_o) + g(z_o)\right)$$

Hence the denominator is non-zero so long as

$g(z_o) \neq f(z_o) \log f(z_o)$ (p_2 is non-zero by definition).

There are now two possibilities:

1) There is an $\epsilon' \leq \epsilon$ such that for $1 < z_o < 1+\epsilon'$

$g(z_o) \neq f(z_o) \log f(z_o)$.

In which case the lemma is proved.

2) There is no such ϵ'.

In this case the points z_o such that $g(z_o) = f(z_o) \log f(z_o)$ are dense near 1. But as f, g and log f are all analytic functions, this implies that $g(z) = f(z) \log f(z)$ for all z.

Rearranging gives

$$g(z)/f(z) \equiv \log f(z).$$

f is algebraic (theorem 2.3.4.1), g is algebraic (theorem 2.4.3.1), so the left hand side of the above is algebraic. $f(z) \not\equiv 0$ so the right hand side is not algebraic. Contradiction. Hence possibility 2 does not arise.

3.2.11 Theorem

If p_n^+ is the n^{th} term in the length generating series for G^+, then

$$p_n^+ \xrightarrow[n\to\infty]{} \frac{-p_2 f(z_o)}{z_o^{n+1}(-p_1)f'(z_o)}$$

Proof: The above is just a specific example of theorem 2.1.4.1 with f^+ for A, p_i^+ for a_i, $p_2 f$ for U and $1-p_2 f$ for V. Condition 1 holds (proof of theorem 3.2.10), condition 2 holds (lemma 3.2.7), z_o is of multiplicity 1. The formula is just an example of that of corollary 2.1.4.3.

3.2.12 Theorem

If q_n^+ is the n^{th} term in the entropy generating series for G^+, then

$$q_n^+ \xrightarrow[n\to\infty]{} \frac{-(n+1)p_2[p_1 \log p_1 f^2(z_o) - g(z_o)]}{z_o^{n+2}\, p_1^{\,2}(f'(z_o))^2}$$

Proof: If A, B are functions with generating series a_i, b_i respectively, both A and B have (multiple) poles at z_o such that z_o is the closest pole to the origin for both, but A's pole has higher multiplicity than B's; then the series for A and B is a_i+b_i, z_o is the pole of A + B closest to the origin and its multiplicity is the same as that of z_o for A, and Lim a_i+b_i = Lim a_i. In particular the second fraction in the sum for g^+ has a double pole at z_o, but

the first a single, so the second fraction can be used to give the behaviour of q_n^+. So putting $U(z) = -p_2(p_1 \log p_1 f^2(z) - g(z))$ and $V(z) = (1-p_1 f(z))^2$, the three conditions of theorem 2.1.4.1 have been shown to hold. So using corollary 2.1.4.4

$$q_n^+ \xrightarrow[n\to\infty]{} \frac{-(n+1)2p_2[p_1 \log p_1 f^2(z_o) - g(z_o)]}{z_o^{n+2}\, 2\, p_1[p_1(f'(z_o))^2 - (1-p_1 f(z_o))f''(z_o)]}$$

This simplifies to the expression in the statement of the theorem by using that $1-p_1 f(z_o) = 0$.

3.3 STANDARD DEFINITIONS OF THE RATE

The rate of a language was first defined by SHANNON in [33] and gives a very useful categorisation of them. He gives two different definitions for discrete languages.

3.3.1 Definition

If $\{\omega_i : i=1,\ldots,n\}$ is a finite set of words, the probability of ω_i is p_i, and all the words ω_i have the same length N, then the rate is defined to be the value of

$$(-\sum_{i=1}^{n} p_i \log p_i)/N$$

3.3.2 Definition

If $\{\omega : \omega \in \Omega\}$ is a set of infinite strings, for any N, $\omega|N$ is the finite word containing the first N letters of ω, $\Omega|N$ (assumed finite) the set of all such initial strings, and $p(\omega|N)$ the probability that the initial string $\omega|N$ is generated. Then in this case the rate is defined to be:

$$\lim_{N\to\infty} \left\{ \left(-\sum_{\omega|N \in \Omega \in N} p(\omega|N) \log p(\omega|N) \right) /N \right\}$$

3.3.3 Remark

These two definitions are connected because if R_1 is the first rate for a finite set Ω, and R_2 is the second set for Ω^c, the infinite closure of Ω, then $R_1 = R_2$.

3.4 FIVE RATES FOR A CONTEXT-FREE LANGUAGE

3.4.1 Remark

In general neither of Shannon's definitions apply directly to a context-free language (although it is possible to construct a language to fit the conditions of the first). It is necessary to broaden the definitions. Five different generalizations will be considered, and the values of the 'rate' as defined by each written as $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ and $\mathcal{R}_5$ respectively.

3.4.2 Notation

In order to make formulas more readable by reducing the number of brackets they contain, μ_ω will sometimes be written instead of $\mu(\omega)$ (the probability that ω is generated). N_ω ,$(N(\omega))$ is the length of (number of terminals in) ω .

3.4.3 Definition

If Ω is the set of words generated by a context-free grammar

$$\mathcal{R}_1 = \Big(-\sum_{\omega \in \Omega} \mu_\omega \log \mu_\omega \Big) \Big/ \Big(\sum_{\omega \in \Omega} \mu_\omega N_\omega \Big)$$

3.4.4 Definition

If Ω is as above

$$\mathcal{R}_2 = -\sum_{\omega \in \Omega} \left(\mu_\omega \log \mu_\omega / N_\omega \right)$$

3.4.5 Remark

Both these definitions reduce to Shannon's first definition if Ω is finite, and all N_i are the same and equal to N. The next three definitions are related to Shannon's second definition. Loosely

speaking $\mathcal{R}_3$ is obtained by applying Shannon's definition to the original language and $\mathcal{R}_4$ by applying it to its infinite closure. $\mathcal{R}_5$ is slightly different but analogous to the definition that Kuich uses.

3.4.6 Notation

Much of what follows is made confusing because of the many slightly different sets and measures which are being used simultaneously. To help reduce this confusion the complete pattern of super and subscripts needed is presented here.

The three basic underlying sets used are Ω, ambiguously the set of all parses/set of all strings generated by a grammar (this is alright because by the assumption of unambiguity, strings and parses are in bijective correspondence), $\Omega^+ = \Omega^c$, $(but\ M' \neq M^c)$. the set containing all finite and infinite concatenations of elements of Ω, and Ω', a set consisting of the (finite) strings of padded out with an infinite number of a new 'null' symbol.

In addition to the superscripts '+', 'c', ',', there are the subscripts '(N)', which when added to a sign X means that set which contains just those strings of length N which belong to X, and '(|N)' which means those strings which are an initial string of length N of some string in X. In general $X^{(|N)}$ is not a subset of X, but $X^{(N)}$ is always a subset of both $X^{(|N)}$ and X.

Subscripts may be added to the set symbols to give a variable subset of the set denoted by the superscript.

Variables are based on the symbol ω and have the same pattern of superscripts to show which sets they vary over. An exception is that a variable with no superscript may if required be used to vary over the set of all strings from the terminal alphabet, not just those in Ω. Subscripts are used to distinguish one variable from another.

The pattern of variables and sets is

$$\omega \in \Omega \qquad \omega^{+} \in \Omega^{+} \qquad \omega^{c} \in \Omega^{c} \qquad \omega' \in \Omega'$$

$$\omega^{(N)} \in \Omega^{(N)} \qquad \omega^{+(N)} \in \Omega^{+(N)} \qquad \omega^{c(N)} \in \Omega^{c(N)} \qquad \omega'^{(N)} \in \Omega'^{(N)}$$

$$\omega^{(|N)} \in \Omega^{(|N)} \qquad \omega^{+(|N)} \in \Omega^{+(|N)} \qquad \omega^{c(|N)} \in \Omega^{c(|N)} \qquad \omega'^{(|N)} \in \Omega'^{(|N)}$$

Examples of subscripts are $\Omega_1^{+(|N)} \subseteq \Omega^{+(|N)}$, and $\omega_1^{+(N)} \in \Omega^{+(N)}$

Although Ω^{+} and Ω^{c} are the same set, the measure on Ω^{+} is concentrated on finite strings and that on Ω^{c} on infinite. So ω^{+} will always be assumed finite, but ω^{c} infinite. Both $\Omega^{+(N)} = \Omega^{c(N)}$ and $\Omega^{+(|N)} = \Omega^{c(|N)}$ are finite sets, so this convention is not relevant to $\omega^{+(|N)}$ or $\omega^{+(N)}$.

The measures μ have the same pattern of superscripts as the sets and variables. Their definitions are derived in the obvious way but they contain a couple of catches and so are listed here.

μ, μ^{+}, μ^{c} are already known [3.1.12]

$\mu'(\omega') = \mu(\omega)$ where ω' = ω followed by an infinite string of nulls.

$$\mu'(\Omega_1') = \sum_{\omega_1' \in \Omega_1'} \mu'(\omega_1').$$

If $\mu(\Omega^{(N)}) \neq 0$ then $\mu^{(N)}(\Omega^{(N)}) = \mu(\Omega^{(N)}) / \mu(\Omega^{(N)})$ else $\Omega^{(N)}$ is empty and $\mu^{(N)}$ is undefined.

A similar definition to that for $\mu^{(N)}$ holds for $\mu^{+(N)}$ and $\mu'^{(N)}$ (but note that $\Omega'^{(N)}$ is always empty so $\mu'^{(N)}$ is undefined for all N). Because $\mu^{c}(\Omega^{c(N)}) = 0$ the above definition will not work so the more general $\mu^{c(N)}(\Omega_1^{c(N)}) = \lim_{p_1 \to 1} \mu^{+(N)}(\Omega_1^{c(N)})$ is used instead. Finally $\mu^{(|N)}(\Omega_1^{(|N)}) = \mu(\{\omega \in \Omega : \omega = \omega_1 \omega_2$ and $\omega_1 \in \Omega^{(|N)}\})$, and $\mu^{+(|N)}$, $\mu^{c(|N)}$ and $\mu'^{(|N)}$ can be defined similarly.

3.4.7 Remark

It is now possible to make the last three definitions of rates.

3.4.8 Definition

$$\mathcal{R}_3 = \lim_{N\to\infty} \left(-\sum_{\omega\in\Omega^{(1N)}} \mu'^{(1N)}(\omega) \log \mu'^{(1N)}(\omega)/N \right)$$

3.4.9 Definition

$$\mathcal{R}_4 = \lim_{N\to\infty} \left(-\sum_{\omega\in\Omega^{c(1N)}} \mu^{c(1N)}(\omega) \log \mu^{c(1N)}(\omega)/N \right)$$

3.4.10 Definition

$$\mathcal{R}_5 = \limsup_{N\to\infty} \left(-\sum_{\omega\in\Omega^{(N)}} \mu^{(N)}(\omega) \log \mu^{(N)}(\omega)/N \right)$$

3.4.11 Remark

$\mathcal{R}_3$ is that Shannon rate for a communication channel which starting from time zero transmits a single terminal string from the original context-free grammar and then remains silent for ever after.

$\mathcal{R}_4$ is the Shannon rate for a channel which starts from time zero by transmitting a terminal string and immediately it finishes one string starts transmitting another.

$\mathcal{R}_5$ has no such interpretation. It has the advantage that it refers to a language itself, not some construction on it, but the disadvantage that, in a loose sense, it only takes account of the very long (and hence very seldom occurring) strings. It is defined here because it plays an important intermediary role in the following theorems. It also corresponds to the definition given in Kuich.

$\mathcal{R}_5$ is defined here using the 'Lim sup' because in languages in which all words are of length a multiple of N, only every N^{th} term in its series will be non-zero. In general these non-zero terms have a non-zero limit. For $\mathcal{R}_3$ and $\mathcal{R}_4$ the 'Lim' and 'Lim sup' give the same value.

3.5 CALCULATING THE RATES (Part 1)

In general none of the rates can be calculated directly because they have definitions which involve infinite sums and limits. The theorems in this and the next subsection give finite explicit formulas for three of them.

3.5.1 Theorem

$\mathcal{R}_1 = g(1)/f'(1)$ and can be calculated with finite effort.

Proof: $g(1) = -\sum_{\omega\in\Omega} \mu_\omega \log \mu_\omega$ and $f'(1) = \sum_{\omega\in\Omega} \mu_\omega N_\omega$.
The proof of theorem 2.4.3.1 gives $g(1)$ as a rational compound of $f(1)$'s and hence finitely calculable, and $f'(1)$ is finitely calculable by corollary 2.3.3.3.

3.5.2 Theorem

$$\mathcal{R}_2 = \int_0^1 \frac{g(z)}{z}\, dz$$

Proof: $$\frac{g(z)}{z} = -\sum_{\omega\in\Omega} \mu_\omega \log \mu_\omega z^{(N_\omega - 1)}$$

So $$\int_0^z \frac{g(z)}{z}\, dz = -\sum_{\omega\in\Omega} \mu_\omega \log \mu_\omega z^{N_\omega}/N_\omega$$

and $$\int_0^1 \frac{g(z)}{z}\, dz = -\sum_{\omega\in\Omega} \mu_\omega \log \mu_\omega/N_\omega$$

The right hand side of this final equation is the definition of $\mathcal{R}_2$.

3.5.3 Remark

The above integration can be solved if g can be displayed explicitly. In general, however, g is only known in terms of the f_i which are in turn only known implicitly as the solutions of polynomial equations $P_i(z,f_i) \equiv 0$. P_i may have terms of degree five or greater in f_i, so in general f_i cannot be found explicitly.

3.5.4 Theorem

$$\mathcal{R}_3 = 0.$$

Proof: Let $E = -\sum_{\omega \in \Omega} \mu_\omega \log \mu_\omega$ be the entropy of the original language (known finite by corollary 2.4.3.2). Let $E_N = -\sum_{\omega \in \Omega^{(|N)}} \mu^{(|N)}(\omega) \log \mu^{(|N)}(\omega)$ be the entropy of the set of initial words of length N. (So that $\mathcal{R}_3 = \lim_{N \to \infty} (E_N/N)$.) Let $E_{(\omega|N)}$ be the entropy of the set of words starting with the string $\omega|N$. It is well known that

$$E = E_N + \sum_{\omega|N \in \Omega^{(|N)}} \mu^{(|N)}(\omega|N)\, E_{(\omega|N)} \qquad \text{(for all N)},$$

and that E, E_N and $\sum \mu^{(|N)}(\omega|N)\, E_{(\omega|N)}$ are all non-negative, (Khinchin [ ] p.6, equation 3).

Hence $E \geq E_N$ for all N and so

$$\mathcal{R}_3 = \lim_{N \to \infty} E_N/N \leq \lim_{N \to \infty} E/N = 0.$$

3.5.5 Remark

The above theorem shows that $\mathcal{R}_3$ does not discriminate between grammars. So although its definition might have originally appeared promising, it turns out to be useless. $\mathcal{R}_3$ will therefore not be mentioned again. In physical terms the theorem shows that even the information contained in the occasional very long string is never enough to compensate for the infinite time during which the channel is in effect switched off.

3.5.6 Remark

Note that the next two theorems are done in reverse order, so that $\mathcal{R}_5$ is found before $\mathcal{R}_4$.

3.5.7 Theorem

$$\mathcal{R}_5 = \operatorname*{Lim\,sup}_{N\to\infty} \left(\frac{q_N}{Np_N} + \frac{\log p_N}{N} \right)$$

where p_N and q_N are the N^{th} terms of the length and entropy series respectively.

Proof:

$$\mathcal{R}_5 \underset{\text{def}}{=} \operatorname*{Limsup}_{N\to\infty} \left(-\sum_{\omega\in\Omega^{(N)}} \mu^{(N)}(\omega) \log \mu^{(N)}(\omega)/N \right)$$

So using the definition $\mu^{(N)}(\omega) = \mu(\omega)/p_N$

$$\mathcal{R}_5 = \operatorname*{Lim\,sup}_{N\to\infty} \left(-\sum_{\omega\in\Omega^{(N)}} \frac{\mu\omega(\log\mu\omega - \log p_N)}{Np_N} \right)$$

$$= \operatorname*{Lim\,sup}_{N\to\infty} \left(\frac{1}{Np_N} (-\Sigma \mu\omega \log\mu\omega) + \frac{\log p_N}{Np_N} (\Sigma \mu\omega) \right)$$

but $-\sum_{\omega\in\Omega^{(N)}} \mu\omega\log\mu\omega = q_N$ by definition and $\sum_{\omega\in\Omega^{(N)}} \mu\omega = p_N$

so

$$\mathcal{R}_5 = \operatorname*{Lim\,sup}_{N\to\infty} \left(\frac{q_N}{Np_N} + \frac{\log p_N}{N} \right) .$$

3.6 A STRENGTHENING OF MACMILLAN'S THEOREM

3.6.1 Motivation

At the last stage in the writing of this thesis it became apparent that one of the proofs in it did not work because none of the versions of Macmillan's theorem which it might use gave close enough bounds for the convergence of a certain sequence. This section remedies the lack by proving a new stronger version.

Macmillan's theorem is about the convergence of the n-step entropy of a stochastic process as n gets large. There are two versions, one, the mean ergodic theorem, says that the entropy converges in the L_1 mean. The other, the individual ergodic theorem, that it converges almost everywhere. (Different variants of both these theorems give different restrictions on the class of the stochastic processes to which the theorems apply.) The convergence in the L_1 mean is analogous to the weak law of large numbers and the almost everywhere convergence to the strong law, the difference and reason why they are not examples of the corresponding large number laws is that in Macmillan's theorem the partial sums are not (in general) independent variables.

This section has as its main result a new version of Macmillan's theorem which is analogous to the central limit theorem. The class of stochastic processes for which the result is proved is very small, but just large enough to cover the processes which are of interest here, that is those which are obtained from context free grammars. The theorem surely holds more generally, but the proof here cannot be extended much because it uses special methods which can only be applied to these processes.

3.6.2 Plan of this section

There are two distinct kinds of structure involved in context free grammar theory: the top-to-bottom branching tree structure of the parses; and the left-to-right linear structure of the generated language. The difficulty of the theory lies in that although each of the above structures is reasonably simple and well understood on its own, the two together do not interact in any particularly regular way. Corresponding to the two structures there

are two ways of handling the theory: either by basing the exposition on the tree structure and treating the linear structure as a complicating factor; or by basing it on the linear structure and treating the tree structure as the complication. The bulk of this thesis is designed according to the first approach, but unfortunately this section seems to need the second.

So this section is designed as follows. First some background is covered. In particular stochastic processes and a few of their properties are mentioned, including ergodicity and the uniform mixing property. Next it is shown how grammars may generate their languages as stochastic processes, and in particular some special grammars, those in Greibach normal form. A new normal form, Greibach supernormal form, is defined, and this with some additional lemmas allows a theorem of Ibragimov and Linnik [17] to be used to obtain the central limit convergence.

The above theorem requires as an additional assumption that its stochastic process be uniformly mixing, and the next part of this section is devoted to showing that this assumption is unnecessary. The uniform mixing property is non-constructive, so firstly a finitely decidable property of a grammar is found which holds if and only if its process is uniformly mixing. Then it is shown how to map any grammar to another which has the same limit properties but which is also uniformly mixing. So the central limit convergence theorem extends to all context-free grammar processes whether uniformly mixing or not.

Finally two lemmas are deduced from the central limit convergence; these give the results which are actually required in the next section.

3.6.3 Acknowledgement

I could not have proved this theorem without the assistance of Professor Bill Parry. He told me about [17] and suggested a way to prove my theorem. The proof here up to the central limit theorem with the uniform mixing assumption is similar to his, but altered to allow the remainder of the proofs to be carried out. He has not yet (at the time of writing) seen the new proof, which is therefore entirely my responsibility.

3.6.4 Notation

There has not been time to completely unify the notation of this section. Some of it is compatible with that of the rest of the thesis and some has been taken from other books, in particular [20].

3.6.5 Definitions

If Ω is a set of sequences $x = \langle \ldots, x_{-1}, x_0, x_1, x_2, \ldots \rangle$ then a subset of Ω is a cylinder if it can be defined by the properties of the values of a finite number of indices.

M_t^s where $t \leqslant s$ is the class of all cylinders which may be defined using only properties of the values of the indices from t to s. (t and s may be integers, $+\infty$ or $-\infty$.)

A stochastic process is a probability measure space $\langle \Omega, \mathcal{A}, \mu \rangle$ where the underlying set Ω is the set of all two-way infinite sequences $x = \langle \ldots, x_{-1}, x_0, x_1, \ldots \rangle$ where each x_i is taken from some set A called the alphabet, and the σ-field $\mathcal{A}$ is the minimal σ-field containing all the cylinders. (A more general definition exists with a more complicated $\mathcal{A}$.)

In this thesis, stochastic processes are restricted in that either A is finite, or A is countable.

The operator T is the function which maps a sequence to the same sequence shifted by one position, that is

$$T\langle \ldots,x_{-1},x_0,x_1,\ldots\rangle = \langle \ldots,x'_{-2},x'_{-1},x'_0,\ldots\rangle$$

where $x'_i = x_{i+1}$

T extends to sets of sequences by the rule

$$T(S) = \{Tx : x \in S\} .$$

A set $S \subseteq \Omega$ is <u>invariant</u> if $T(S) = S$.

A process is <u>ergodic</u> if every invariant set has measure either 0 or 1.

A process is <u>stationary</u> if $\mu(T(S)) = \mu(S)$ for any set S .

In a stationary process, the <u>n-step dependent probability</u> of a symbol $a_0 \in A$ given $a_{-n},\ldots,a_{-1}$, written $\mu(a_0 | a_{-n},\ldots,a_{-1})$ is defined as:

$$\mu(a_0 | a_{-n},\ldots,a_{-1}) = \frac{\mu\{x : x_{-n} = a_{-n},\ldots,x_0 = a_0\}}{\mu\{x : x_{-n} = a_{-n},\ldots,x_{-1} = a_{-1}\}}$$

A <u>markov process</u> is a stationary process where the n-step dependent probability of a symbol is independent of all but the last symbol, that is

$$\mu(a_0 | a_{-n},\ldots,a_{-1}) = \mu(a_0 | a_{-1})$$

whatever $a_{-n},\ldots,a_{-2}$ may be.

Two states a_1, a_2 from the alphabet of a process are said to <u>communicate</u> if there are indices i, j such that $\mu\{x : x_0 = a_1,\ x_i = a_2\} > 0$ and $\mu\{x : x_0 = a_2,\ x_j = a_1\} > 0$.

3.6.6 <u>Proposition</u>

If all states in a countable or finite markov process communicate with each other then that process is ergodic [25 , p.423].

3.6.7 Definition

If $\langle \Omega, \mathcal{A}, \mu \rangle$ is a stationary process then its r^{th} mixing coefficient is

$$\phi(r) = \sup_{A \in M_{-\infty}^{t}} \sup_{B \in M_{t+r}^{\infty}} \frac{|\mu(A \cap B) - \mu(A)\mu(B)|}{\mu(A)}$$

A stationary process is said to obey the uniform mixing condition if

$$\lim_{r \to \infty} \phi(r) = 0.$$

3.6.8 Proposition

Let A_1 and A_2 be any two alphabets and f any function such that $f: A_1 \xrightarrow{\text{onto}} A_2$. Then f induces unique maps from $\Omega = \{\langle \ldots, x_0, x_1 \ldots \rangle\}$ to the sequences $f(\Omega) = \{\langle \ldots, fx_0, fx_1, \ldots \rangle\}$, and from the σ-field $\mathcal{A}$ to the well-defined σ-field $f\mathcal{A}$ such that $f^{-1}A$ is measurable if and only if A is measurable & $\mu(f^{-1}A) = \mu(A)$.

3.6.9 Theorem

If Ω is ergodic so is the process $f(\Omega)$.

Proof: If A is an invariant set of $f(\Omega)$ then $f^{-1}(A)$ is an invariant set of Ω with the same measure.

3.6.10 Theorem

If Ω obeys the uniform mixing condition then so does $f(\Omega)$.

Proof: If A and B are in $f(\Omega)$ and if $A \in f(M_{-\infty}^{t})$ and $B \in f(M_{t+r}^{\infty})$ then $f^{-1}(A) \in M_{-\infty}^{t}$ and $f^{-1}(B) \in M_{t+r}^{\infty}$. So the set of values over which the supremum is taken to obtain $f\phi(r)$ is a subset of the set which is used to yield $\phi(r)$, so $f\phi(r) \leqslant \phi(r)$.

3.6.11 Notation (after Macmillan, Khinchin)

If a is the sequence $\langle \ldots, a_{-1}, a_0, a_1, \ldots \rangle$ then

$$p_n(a) = \frac{\mu\{x : x_0 = a_0, x_{-1} = a_{-1}, \ldots, x_{-n} = a_{-n}\}}{\mu\{x : x_{-1} = a_{-1}, \ldots, x_{-n} = a_{-n}\}}$$

or in other words,

$$p_n(a) = \mu(a_0 \mid a_{-n}, \ldots, a_{-1}).$$

3.6.12 Remark

The next part shows how a grammar may be turned into a stochastic process by using a stack.

3.6.13 Definition

The process related to a grammar G is the markov process with the following properties.

The states of the process are finite strings of terminals and non-terminals of the grammar G, the states are also called stacks.

The active symbol is the symbol at the left hand end of the stack.

The transitions are:

The successor of the empty stack is the empty stack, the output string is the empty string, the probability of the transition is one.

The successor of a stack where the active symbol is a terminal t is the same string with that terminal t deleted, the output symbol is t, the probability of the transition is one.

There are several possible successors of a stack Xu whose active symbol is the non-terminal X, one possible successor corresponding to each production $\langle X \rightarrow v \rangle$ with X as its left-hand side. The successor corresponding to $\langle X \rightarrow v \rangle$ is the stack vu, the output

string of this transition is the empty string, its probability is the preprobability of $\langle X \to v \rangle$.

3.6.14 Remark

In general the output string will be shorter than the sequence of stacks. And worse still, the ratio of the lengths of the output string and the sequence of stacks will be variable. However for a special class of grammars, those in Greibach normal form, the ratio is constant and so the rate of the generated language is simply related to the information rate of the Markov process of its generating stacks.

3.6.15 Definition

A grammar is in Greibach normal form if the right hand of every production is of the form tu, where t is a terminal symbol and u is a possibly empty string containing only non-terminal symbols.

3.6.16 Lemma

A terminal is output on every second transition of the process of a grammar in Greibach normal form. The stack alternates between containing all non-terminal symbols and containing just one terminal symbol, the active symbol.

Proof: By induction. If only the active symbol is terminal then it is output and the next stack contains no terminals. If the stack contains no terminals, then the empty string is output and the active symbol is replaced by a right-hand side tu of one of its productions, so only the new active symbol is terminal.

3.6.17 Remark

Because of the above lemma it is possible to alter the definition of the stack process of a Greibach normal form grammar so that the new process makes two at a time of the transitions of the old process.

3.6.18 Definition

The Greibach process related to a Greibach normal form grammar G is the Markov process with the following properties.

The states are finite strings of non-terminals of G, called stacks.

The active symbol of a state is the symbol at the left-hand end of the stack.

There are several possible successors of a stack Xu whose active symbol is X, one for each of the productions $\langle X \rightarrow tv \rangle$. The successor corresponding to $\langle X \rightarrow tv \rangle$ is vu, the output is t, the probability of the transition is the preprobability of $\langle X \rightarrow tv \rangle$.

The grammar can be adjusted to prevent the empty stack occurring, in an infinite closure grammar it cannot anyway; in a finite grammar the productions involving the start symbol S can easily be adjusted to yield a new grammar which produces an infinite tail of dummy symbols after the output string.

3.6.19 Proposition

For any grammar there is a grammar in Greibach normal form which generates the same language.

Proof: A proof was originally presented by Greibach herself in [12]. A more constructive proof is given in Hopcroft and Ullman [15].

3.6.20 Remark

The above theorem is only stated for nonprobabilistic grammars. But Hopcroft and Ullman's proof depends on two key lemmas which can easily be extended to the following probabilistic form.

3.6.21 Lemma

Let G be a grammar,

$\{P_{ij} : P_{ij} = \langle X_i \rightarrow \prod_k X_{ijk} \rangle \; ; \; j=1,\ldots,n_i\}$ the set of all its productions with X_i on the left-hand side, p_{ij} the preprobability of P_{ij}, $P_{lm} = \langle X_l \rightarrow \alpha X_i \beta \rangle$ any production containing X_i on its right-hand side, where the preprobability of P_{lm} is p_{lm}. Let G' be obtained from G by deleting the production P_{lm} and adding the n_i productions $P_{lj} = \langle X_l \rightarrow \alpha \prod_k X_{ijk} \beta \rangle$ with the preprobabilities $p_{lj} = p_{lm}p_{ij}$. If all the above is so then G' and G generate the same language with the same probability structure.

Proof: This is lemma 4.2 on p.53 of [15], restated in the notation of this thesis with the probability structure added. The basic idea is that whenever the production P_{lm} is used in the old grammar G then the symbol X_i is produced, and X_i must later be used in one of the productions P_{ij}. G achieves the same terminal string by using the single production P'_{lj}.

3.6.22 Definition

A production $P_{ij} = \langle X_i \rightarrow \alpha_{ij} \rangle$ is called left recursive if X_i is the first symbol of α_{ij}.

3.6.23 Lemma

Let G be a grammar, $\{P_{ij} : P_{ij} = \langle X_i \rightarrow \alpha_{ij} \rangle \; ; \; j=1,\ldots,n_i\}$ the set of all the productions with X_i on the left. These productions may be renumbered if necessary so that the left recursive productions are the first h ($0 \leq h \leq n_i$), and the remainder are not left recursive. $\{\beta_{ij} : j=1,\ldots,h\}$ is defined so that the first h productions $\langle X_i \rightarrow \alpha_{ij} \rangle$ are also $\langle X_i \rightarrow X_i \beta_{ij} \rangle$. Let the preprobability of P_{ij} be p_{ij} and define

$$p_1 = \sum_{j=1}^{h} p_{ij} \quad \text{and} \quad p_2 = \sum_{j=h+1}^{n_i} p_{ij}$$

that is p_1 is the probability that a left recursive production be chosen and $p_2 = 1-p_1$ the remaining probability.

If G'' is obtained from G by deleting all the productions with X_i on the left and adding instead the new productions

$$\left.\begin{array}{l} X_i \to \{p_{ij}\}\ \beta_{ij} \\ X_i \to \left\{\frac{p_2 p_{ij}}{p_1}\right\}\ \beta_{ij}\, Z \end{array}\right\} j=1,\ldots,h$$

$$\left.\begin{array}{l} Z \to \{p_{ij}\}\ \alpha_{ij} \\ Z \to \left\{\frac{p_2 p_{ij}}{p_2}\right\}\ \alpha_{ij}\, Z \end{array}\right\} j=h+1,\ldots,n_i$$

(where Z is a new non-terminal symbol), then G'' generates the same language as G with the same probability structure.

Proof: This is lemma 4.3 of [15 , p.53] (extended to include the information about the probabilities). The idea is that a left recursive sequence of productions of G

$$X_i \Rightarrow X_i \beta_{ij(1)} \Rightarrow X_i \beta_{ij(2)} \beta_{ij(1)} \Rightarrow \cdots$$

$$\cdots \Rightarrow X_i \beta_{ij(m)} \beta_{ij(m-1)} \cdots \beta_{ij(1)} \Rightarrow$$

$$\Rightarrow \alpha_{ij(m+1)} \beta_{ij(m)} \cdots \beta_{ij(1)}$$

which yields the probability $\prod_{l=1}^{m+1} p_{ij(l)}$, corresponds to the right recursive sequence of productions of G''

$$X_i \Rightarrow \alpha_{ij(m+1)}\ Z \Rightarrow \alpha_{ij(m+1)}\ \beta_{ij(m)}\ Z \Rightarrow \cdots$$

$$\cdots \Rightarrow \alpha_{ij(m+1)} \beta_{ij(m)} \cdots \beta_{ij(2)} Z$$

$$\Rightarrow \alpha_{ij(m+1)} \beta_{ij(m)} \cdots \beta_{ij(2)} \beta_{ij(1)}\ Z$$

which yields the probability $\frac{p_2}{p_1}\left(\prod_{l=1}^{m+1} p_{ij(l)}\right)\frac{p_1}{p_2}$.

Clearly the two generated probabilities arethe same.

3.6.24 Theorem

For any grammar G there is another G' in Greibach normal form which generates the same language with the same probability structure.

Proof: The proof in [15] of proposition 3.6.19 constructs a series of grammars $G = G_1, G_2, \ldots, G_k = G'$, where each grammar is obtained from the previous one by the transformation in one of the previous two lemmas. The extension of the lemmas tells that the probability structure remains the same along the sequence.

3.6.25 Remark

The next step is to show that every language can be generated by an even more restricted type of grammar, one in what will here be called Greibach supernormal form. The reason for constructing such a grammar is purely technical: it is to allow proposition 3.6.8 and theorems 3.6.9 and 3.6.10 to be used.

3.6.26 Definition

A grammar is in Greibach semisupernormal form if it is in Greibach normal form and has the additional property that if $\langle X \to t_1 \alpha_1 \rangle$, $\langle X \to t_2 \alpha_2 \rangle$ are two productions with the same left-hand symbol X, (where t_1, t_2 are terminals, α_1, α_2 possibly empty strings of non-terminals), then $t_1 = t_2$.

3.6.27 Lemma

If G is a probabilistic grammar in Greibach normal form, then there is another G' in semisupernormal form which generates the same language with the same probability structure.

Proof: Call a non-terminal initial variable (i.v.) if it can generate terminal strings starting with different initial symbols, or equivalently (for Greibach normal form grammars) if it can appear on left of two or more productions $\langle X \to t_i \alpha_i \rangle$ with different symbols t_i. Clearly if a Greibach grammar can be changed to another with one less i.v. non-terminal, then by induction it can also be converted to a Greibach grammar with no i.v. non-terminals; in other words a grammar in semisimple form.

The induction step can be done by choosing an i.v. non-terminal, and replacing it with several non-terminals X_i, one for each possible initial symbol t_i. Every production $\langle X \to t_i \rangle$ is replaced by $\langle X_i \to t_i \rangle$, and then any production $\langle Y \to \alpha X \beta \rangle$ is replaced by the several productions $\{\langle Y \to \alpha X_i \beta \rangle$: where i is such that t_i exists$\}$. (And similarly $\langle Y \to \alpha X \beta X \gamma \rangle$ is replaced by all possible combinations $\langle Y \to \alpha X_i \beta X_j \gamma \rangle$ and so on for more occurrences of X in the right-hand side.) The preprobabilities can also easily be adjusted.

Clearly none of the new symbols X_i is i.v. The i.v. non-terminal X has been deleted, and the other non-terminals generate the same terminal strings as before, so the induction step is proved.

3.6.28 Definition

A grammar G is in supernormal form if it is in semisupernormal form and in addition the probability of a stack transition can be determined from the active symbol after the transition has taken place.

3.6.29 Lemma

For any semisupernormal form grammar there is a supernormal form grammar which generates the same language with the same

probability structure.

Proof: If any non-terminal X is the left-hand side of both productions $\langle X \to t \rangle$ and productions $\langle X \to t\alpha \rangle$ (where α is non empty), then it may be replaced by two non-terminals, X_1 which only appears in the production $\langle X_1 \to t \rangle$, and X_2 which appears on the left of the remainder. Every production with a single occurrence of X on the right is split into one with X_1 and one with X_2, every production with two X's is split into four, and so on as in the last proof. The probabilities can also be adjusted, and $\langle X_1 \to t \rangle$ must have preprobability one.

The productions of the form $\langle X \to t\alpha \rangle = \langle X \to tY\beta \rangle$ (β possibly empty) may be numbered from 1 to h say. If $\langle X \to tY\beta \rangle$ is the j^{th} production of the old grammar then the new grammar contains the h new productions $\langle X_i \to tY_j\beta \rangle$ and also $\langle X \to tY_j\beta \rangle$ instead. The preprobability of all these is the same as the preprobability of $\langle X \to tY\beta \rangle$. A final version of the grammar may be obtained by deleting redundant productions.

This grammar is in supernormal form. Clearly it is in semi-supernormal form because each new symbol X_i is i.v. if and only if the symbol X which gave size to it was i.v. The stack property also holds because if the active symbol after a transition is Y_j then transition must have been one of those corresponding to a production of the form $\langle X_i \to tY_j\beta \rangle$, and all these have the same preprobability. Alternatively if the symbol is Y then the transition must have been one of the form $\langle X \to t \rangle$ with preprobability one.

3.6.30 Theorem

Every probabilistic context-free language may be generated by a grammar in supernormal form.

Proof: Consequence of lemmas 3.6.24, 3.6.27, and 3.6.29.

3.6.31 Remark

The next stage is to prove a limited central limit theorem.

3.6.32 Proposition

Let the uniform mixing sequence (of numbers) X_j satisfy $\mathcal{E}|X_j|^{2+\delta} < \infty$ for some $\delta > 0$. If

$$\sigma_n^2 = \mathcal{E}(X_1 + \ldots + X_n)^2 \to \infty$$

as $n \to \infty$, then X_j satisfies the central limit theorem.

Proof: This is theorem 18.5.1 of Ibragimov and Linnik [ ].

($\mathcal{E}$ means expectation.)

3.6.33 Remark

The above proposition can be applied to the random sequence of real numbers $(\log p_j) + H$, where $p_j = p_j(\omega)$ is the probability of the j^{th} transition, and H is the entropy of the process.

3.6.34 Lemma

The process consisting of the sequence of real numbers $\log p_j$ is ergodic.

Proof: The original Markov process is ergodic and by the construction of the supernormal form grammar, $\log p_j$ is a function of its states. Hence the conclusion follows from lemma 3.6.9.

3.6.35 Lemma

The entropy H of the supernormal form process exists.

Proof: Consequence of above.

3.6.36 Lemma

If the supernormal form process is uniform mixing then so is the sequence $(\log p_j + H)$.

Proof: $\log p_j + H$ is a function of the states of the supernormal form process, so the conclusion is a special case of theorem 3.6.10.

3.6.37 Theorem

If 1) the supernormal form process is uniform mixing and

2) $\sigma_n^2 = \mathcal{E}(\log p_1(\omega) + \ldots + \log p_n(\omega) + nH)^2 \to \infty$ as $n \to \infty$

then $\log p_j + H$ satisfies the central limit theorem.

Proof: This is an instance of 3.6.32. All that is necessary is to show that $(\log p_j + H)^{2+\delta} < \infty$ for some δ. This is easy because the supernormal form process is Markov so the distribution of $(\log p_j + H)$ is independent of j. Its value may be found from the finite sum

$$\sum_{i,j} p_i(\log p_{ij} + H)^{2+\delta}$$

where p_i is the probability that the non-terminal X_i is at the top of the stack, and p_{ij} the preprobability of the production P_{ij}. This value is clearly finite so long as each $\log p_{ij}$ is finite; in other words every transition has probability > 0.

3.6.38 Remark

The second assumption of the above theorem will cause no trouble. If it does not hold then the partial sums converge to their limit even faster than in the central limit case.

3.6.39 Remark

The next step is to work on assumption one. It will be shown that the definition of uniform mixing can be simplified.

3.6.40 Theorem

For a countable Markov process, the supremum $\phi(r)$ in the definition of uniform mixing is the same, even when A is restricted to vary only over coordinate sets.

Proof: Let $A \in M_{-\infty}^{t}$, $\mu(A) > 0$. Then A can be split into the union of its cross-sections on coordinate t, that is

$$A = \sum_i A^i A_i \quad \text{where } A^i \in M_{-\infty}^{t-1} \text{ and } A_i \in M_t^t$$

and in addition each A_i completely determines the state at time t, and only A^i such that $\mu(A^i) > 0$ are considered. (There must be at least one because $\mu(A) > 0$ and the process is countable.)

Then given $B \in M_{t+r}^{\infty}$

$$\frac{|\mu(A \cap B) - \mu(A)\mu(B)|}{\mu(A)}$$

$$= \frac{|\mu(\sum A^i A_i B) - \mu(\sum A^i A_i)\mu(B)|}{\mu(\sum A^i A_i)}$$

$$= \frac{|\sum_i (\mu(A^i A_i B) - \mu(A^i A_i)\mu(B))|}{\sum_i \mu(A^i A_i)}$$

(because $\sum A^i A_i$ is a disjoint countable sum).

$$\leq \frac{\sum_i |\mu(A^i A_i B) - \mu(A^i A_i)\mu(B)|}{\sum \mu(A^i A_i)}$$

(because moduli of sums are less than sums of moduli).

Now by the definition of conditional probability (3.6.5)

$$\mu(A^i A_i B) = \mu(A^i A_i)\mu(B | A^i A_i)$$

so the above fraction may be expressed

$$= \frac{\sum_i \mu(A^i A_i) \left| \mu(B\, A^i A_i) - \mu(B) \right|}{\mu(A^i A_i)}$$

The above expression is a weighted mean of terms $\left| \mu(B|A^i A_i) - \mu(B) \right|$. So either all such terms have the same value or else there is at least one index I for which the term $\left| \mu(B|A^I A_I) - \mu(B) \right|$ has value greater than the mean. In either case, for some I the value of the above weighted mean

$$\leq \left| \mu(B|A^I A_I) - \mu(B) \right|$$

Finally by the assumption that the process is Markov

$$\mu(B|A^I A_I) = \mu(B|A_I)$$

and so multiplying top and bottom by A_I, the above term

$$= \frac{\left| \mu(A_I B) - \mu(A_I)\mu(B) \right|}{\mu(A_I)}$$

All in all the above reasoning easily yields

$$\phi(r) \leq \sup_{A_I \in M_t^t, B \in M_{t+r}^\infty} \frac{\left| \mu(A_I B) - \mu(A_I)\mu(B) \right|}{\mu(A_I)}$$

The converse inequality is obvious because $M_t^t \subset M_{-\infty}^t$ so the above inequality is in fact an equality.

3.6.41 Proposition

For a homogeneous Markov chain Ω, if (1) there is a probability measure μ on Ω, an integer r and a positive real such that for all measurable sets A in the coordinate space,

$$\frac{\mu\{x : x_t = \xi,\ x_{t+r} \in A\}}{\mu\{x : x_t = \xi\}} \geq 1 - \epsilon$$

whenever $\mu(A) \leq \epsilon$

and (2) there is only one ergodic set, then the process is uniformly mixing.

Proof: This is stated on pp.367 and 368 of [17] in a slightly different notation.

3.6.42 Theorem

Either an ergodic process is uniform mixing or else for every measure μ, for every integer r, for every real ϵ there exists a measurable coordinate set A such that

$$\frac{\mu\{x : x_t = \xi, x_{t+r} \in A\}}{\mu\{x : x_t = \xi\}} \geq 1 - \epsilon$$

Proof: This can be derived by predicate calculus from proposition 3.6.41.

3.6.43 Theorem

Either an ergodic Markov process is uniform mixing or else all of the mixing coefficients $\phi(r) = 1$.

Proof: If it isn't uniform mixing then the previous theorem shows that $\phi(r) \geq 1 - \epsilon$ for any ϵ.

3.6.44 Remark

So for the processes derived from grammars, either there are for any given r, stacks which completely determine the next r transitions, or else the process is uniform mixing. The next stage of this proof is to find which stacks determine their transitions.

3.6.45 Definition

A non-terminal is deterministic if it can generate exactly one terminal string; otherwise it is non-deterministic.

3.6.46 Lemma

If a deterministic symbol is the active symbol of a stack, then only one sequence of transformations is possible until that symbol has been removed. This sequence has probability one.

Proof: The only sequence is that which generates the sole terminal string.

3.6.47 Lemma

If there are r deterministic symbols at the head of the stack then at least the next r transitions are determined.

Proof: Each symbol needs at least one transition to delete it.

3.6.48 Lemma

If a non-deterministic symbol is the active symbol, then the probability of any sequence of transitions which removes it is less than one.

Proof: It is the same as the probability of the string it generates.

3.6.49 Theorem

The process obtained from a grammar is uniform mixing if and only if there is a finite maximum of the number of deterministic symbols which can appear adjacent to each other.

Proof: This is an obvious consequence of 3.6.43, 3.6.47 and 3.6.48.

3.6.50 Lemma

The right-hand side of a production stemming from a deterministic symbol contains only deterministic symbols.

Proof: Only one terminal sequence can be produced.

3.6.51 Definition

A type 1 production is one whose right-hand side contains only deterministic symbols.

A type 2 production is one where the right-hand side ends with a non-deterministic symbol.

A type 3 production is one which contains a non-deterministic symbol but whose final right-hand symbol is deterministic.

3.6.52 Remark

The three types are disjoint but include all productions between them.

3.6.53 Definition

The score of a deterministic symbol is the length of the sole terminal string it can generate.

The score of an adjacent string of deterministic symbols is the sum of the scores of its components.

The score of a string containing one non-deterministic symbol followed by adjacent deterministic symbols is the sum of the scores of the deterministic symbols plus the number one greater than the maximum of the scores of the deterministic symbols in the right-hand side of any production of the grammar.

3.6.54 Theorem

Only a type 3 production can result in an increase in the score of the string of symbols at the top of a stack.

Proof: Otherwise, if a type 1 production is used, either the active symbol was deterministic, in which case the sum of the scores of the new deterministic symbols on the top of the stack is one less than before, as a terminal has been output, or the active symbol was non-deterministic in which case the new string has a lower score by the

definition of the score.

If a type 2 production was used a non-deterministic symbol gets replaced with another so the score stays the same.

3.6.55 Theorem

A sufficient condition for a grammar to have a uniformly mixing process is that it has no type 3 productions.

Proof: By the above theorem the set of scores of adjacent strings of deterministic symbols is bounded. As the score of any deterministic symbol is non-zero, the set of numbers of adjacent deterministic symbols is also bounded. So the conclusion follows from 3.6.49.

3.6.56 Theorem

Every language is isomorphic to a language generated by a uniform mixing grammar.

Proof: Given a supernormal form grammar G let G' be the grammar with the same structure of non-terminals as G, but each terminal in each production changed to a new symbol which uniquely identifies that production. There is clearly a bijective correspondence between the sets of parses of the two languages and hence also between their languages because both are unambiguous.

Let G'' be obtained from G' by rearranging the order of the non-terminals of the type 3 productions of G' so that a non-deterministic symbol comes last. Thus G'' has no type 3 productions. There is a bijective correspondence between the parses of G' and G'' and both are unambiguous so there is also a bijective correspondence between their languages.

$\mathcal{L}(G)$ is isomorphic to $\mathcal{L}(G')$, $\mathcal{L}(G')$ to $\mathcal{L}(G'')$ so $\mathcal{L}(G)$ is isomorphic to $\mathcal{L}(G'')$ which is uniformly mixing.

3.6.57 Theorem

If Ω is a process derived from a Greibach grammar such that

$$\sigma_N^2 = \mathcal{E}(\log p_1(\omega) + \ldots + \log p_N(\omega) + NH)^2 \xrightarrow[N\to\infty]{} \infty$$

then $\log p_i + H$ satisfies the central limit theorem.

Proof: This is theorem 3.6.37 with the uniform mixing condition removed because of the previous theorem.

3.6.58 Corollary

If Ω is any Greibach process then given $\epsilon > 0$ there is an $X < 1$ such that the sequence of initial pieces $\Omega^{c(N)}$ partitions into two sequences of sets, a sequence of high probability sets E_N such that $\left|\log \mu^{c(N)}(\omega)/N + H\right| < \epsilon$ for every $\omega \in E_N$ and the remainder $\bar{E}_N = \Omega^{c(N)} - E_N$, such that

$$-\sum_{\omega \in E_N} \log \mu^{c(N)}(\omega)\, \mu^{c(N)}(\omega)/N < X^N$$

and

$$\mu^{c(N)}(\bar{E}_N) < X^N.$$

Proof: Because the process is Markov each $p_i(\omega)$ has the same probability distribution, so

$$\mathcal{E}(\log p_1(\omega) + \ldots + \log p_N(\omega) + NH)^2 = N^2\mathcal{E}(\log p_1(\omega) + H)^2$$

so if σ_N^2 is bounded, $\sigma_N^2 = 0$.

If $\sigma_N^2 = 0$ then $\log \mu^{c(N)}(\omega) = \sum_{i=1}^{N} \log p_i(\omega) = NH$ for all ω

and so any positive ϵ and $X < 1$ will do.

Otherwise $\sigma_N^2 \to \infty$, the central limit theorem 3.6.57 applies, so writing $\eta(X)$ for the normal function and using that $\mu^{c(N)}(\omega) = \sum_{i=1}^{N} p_i(\omega)$

$$\mu\left\{\omega : \frac{\log \mu^{c(N)}(\omega) + NH}{\sqrt{N}} > a\right\} \xrightarrow[N\to\infty]{} \eta(a)$$

Hence substituting $\sqrt{N}\epsilon$ for a,

$$\mu\left\{\omega : \log \mu^{c(N)}(\omega)/N + H > \epsilon\right\} \xrightarrow[N\to\infty]{} \eta(\sqrt{N}\epsilon)$$

Now $\eta(\sqrt{N}\epsilon) = \int_{\sqrt{N}\epsilon}^{\infty} e^{-y^2}\, dy < \int_{\sqrt{N}\epsilon}^{\infty} 2y\, e^{-y^2}\, dy = e^{-\epsilon^2 N}$

so long as $\sqrt{N}\epsilon > 1$ which it eventually becomes as $N \to \infty$. Hence taking $X = e^{-\epsilon^2}$ the result $\mu^{c(N)}(\bar{E}_N) < X^N$ follows. Yet another similar integration yields the other result.

3.7 CALCULATING THE RATES (PART 2)

This subsection shows how to calculate $\mathcal{R}_4$, which is physically the most important kind of rate. It does this by showing that $\mathcal{R}_4 = \lim_{p_1 \to 1} \mathcal{R}_5^+$, where $\mathcal{R}_5^+$ is the fifth kind of rate for the closure grammar, and then calculating $\mathcal{R}_5^+$. The final result, that $\mathcal{R}_4$ is the entropy of the language divided by the average of the lengths of its sentences, is neat and straightforward, but its derivation here is extremely tortuous and requires some powerful mathematics. Perhaps a simpler proof can be found?

3.7.1 Lemma

$$\mathcal{R}_4 = \limsup_{N \to \infty} - \sum_{\omega \in \Omega^{c(N)}} \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N$$

Proof:

By definition $\mathcal{R}_4 = \lim_{N \to \infty} - \sum_{\omega \in \Omega^{c(|N)}} \mu^{c(|N)}(\omega) \log \mu^{c(|N)}(\omega)/N$

Now any word $\omega \in \Omega^{c(|N)}$ is the first N letters of a concatenation of words from Ω, so it divides into two parts, $\omega_1 \in \Omega^{c(N-a)}$ which consists of complete words, and $\omega_2 \in \Omega^{(|a)}$ which is the final truncated word (of length a). Hence

$$\mu^{c(|N)}(\omega) = \mu^{c(N-a)}(\omega_1)\, \mu^{(|a)}(\omega_2)\, p_N^a$$

where p_N^a is the probability that the final truncated word is of length a in a string of total length N.

This expansion for $\mu^{c(|N)}(\omega)$ can be substituted into the definition of $\mathcal{R}_4$ and the log of a product split in the usual way to yield

$$\mathcal{R}_4 = \lim_{N \to \infty} S_1 + \lim_{N \to \infty} S_2 + \lim_{N \to \infty} S_3$$

where $S_1 = - \sum_{a=0}^{N} p_N^a \sum_{\omega_1 \in \Omega^{c(N-a)}} \mu^{c(N-a)}(\omega_1) \log \mu^{c(N-a)}(\omega_1)/N$

$$S_2 = -\sum_{a=0}^{N} p_N^a \sum_{\omega_2 \in \Omega^{(|a)}} \mu^{(|a)}(\omega_2) \log \mu^{(|a)}(\omega_2)/N$$

$$S_3 = -\sum_{a=0}^{N} p_N^a \log p_N^a/N$$

Now $-\sum_{\omega_2 \in \Omega^{(|a)}} \mu^{(|a)}(\omega_2) \log \mu^{(|a)}(\omega_2) \leqslant -\sum_{\omega \in \Omega} \mu(\omega) \log \mu(\omega) = E$

because the partition defined by the second sum is a refinement of that defined by the first.

Hence $S_2 \leqslant \sum_{a=0}^{N} p_N^a E/N = E/N$

and so $\lim_{N \to \infty} S_2 = 0$.

The next step is to show that $\lim_{N \to \infty} S_3 = 0$ also.

As N gets large the probabilities p_N^a tend to the limit p^a, where p^a is the probability that a random cut cuts off an initial part of a sentence of length a in an infinite concatenation of words from Ω.

So $$S_3 \xrightarrow[N \to \infty]{} \sum_{a=0}^{\infty} p^a \log p^a / N.$$

It is well known that ([7], p.333)

$$p^a = \sum_{i=a}^{\infty} p_i/M$$

where p_i is a term of the length generating series of Ω and M is the mean length of word in Ω.

If f(z) is the length generating function for Ω and f(z) has a minimal simple pole at z_o, then by (by corollary 2.1.43)

$$p_i \xrightarrow[i \to \infty]{} b/z_o^{i+1} = B \alpha^i$$

where $B = b/z_o$, $\alpha = 1/z_o$, and b is some constant.

Now by remark 3.2.6 $z_o > 1$ so $\alpha < 1$. Hence

$$p^a \xrightarrow[a\to\infty]{} \frac{B\alpha^a}{M(1-\alpha)}$$

The sum $-\sum_{a=0}^{\infty} p^a \log p^a$ may now be split into two parts, the first few terms $S_4 = -\sum_{a=0}^{A} p^a \log p^a$ where the approximation $p^a \simeq \frac{B\alpha^a}{M(1-\alpha)}$ is not very close, and the rest $S_5 = -\sum_{a=A} p^a \log p^a$.

S_4 is a finite constant sum so $\lim_{N\to\infty} S_4/N = 0$.

$$S_5 \simeq -\sum_{a=A}^{\infty} \frac{B\alpha^a}{M(1-\alpha)} \log \frac{B\alpha^a}{M(1-\alpha)}$$

$$= -\left(\frac{B}{M(1-\alpha)} \log \frac{B}{M(1-\alpha)} + \sum_{a=A}^{\infty} \alpha^a\right) - \left(\frac{B}{M(1-\alpha)} \log\alpha \sum_{a=A}^{\infty} a\alpha^a\right).$$

As $\alpha < 1$ both these component sums are finite, so S_5 is finite and hence

$$\lim_{N\to\infty} S_5/N = 0.$$

As $S_3 = S_4 + S_5$, $\lim_{N\to\infty} S_3/N = 0$ also.

The proofs that $\lim_{N\to\infty} S_3/N = 0$ for the other cases when $f(z)$ has a multiple pole nearest the origin, or a branch point, or several equidistant singularities are similar.

Because both S_2 and S_3 tend to 0 as N tends to infinity, the expression for $\mathcal{R}_4$ reduces to

$$\mathcal{R}_4 = \lim_{N\to\infty} S_1$$

This can be rewritten by expanding S_1 and using the fact that

$$q^{c(N-a)} = -\sum_{\omega_1 \in \Omega^{c(N-a)}} \mu^{c(N-a)}(\omega_1) \log \mu^{c(N-a)}(\omega_1)$$

as

$$\mathcal{R}_4 = \lim_{N\to\infty} \sum_{a=0}^{N} p_N^a \, q^{c(N-a)}/N$$

which can be algebraically rearranged to give

$$\mathcal{R}_4 = \lim_{N\to\infty} \left(\sum_{a=0}^{N} p_N^a \, q^{c(N-a)}/(N-a) \right) - \lim_{N} \left(\sum_{a=0}^{N} \frac{a}{N} \left(p_N^a \, q^{c(N-a)}/(N-a) \right) \right)$$

There are now two possibilities. Either the sequence $q^{(N)}/N$ tends to a limit. Or (in the case of a language all of whose words have length a multiple of some common factor) only some of the terms $q^{(N)}/N$ are non-zero, but these non-zero terms tend to a limit, and furthermore, if $q^{c(N-a)}/(N-a)$ is zero, then so is p_N^a. In both cases p_N^a (which tends to the limit distribution p^a) is mainly concentrated on those terms with small a, hence large (N-a), $q^{c(N-a)}/(N-a)$ close to the limit and small a/N. Hence,

$$\lim_{N\to\infty} \sum_{a=0}^{N} p_N^a \, q^{c(N-a)}/(N-a) = \limsup_{N\to\infty} q^{c(N)}/N$$

and

$$\lim_{N\to\infty} \sum_{a=0}^{N} a/N \left(p_N^a \, q^{c(N-a)}/(N-a) \right) = 0$$

So expanding $q^{c(N)}$

$$\mathcal{R}_4 = \limsup_{N\to\infty} - \sum_{\omega\in\Omega^{c(N)}} \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N$$

which proves this lemma.

3.7.2 Remark

The crux of the proof of the next main lemma [3.7.6] is the change from a sum weighted by one measure $\mu^{*(N)}$ to the same sum but weighted by the measure $\mu^{c(N)}$. In fact this is intuitively plausible because as N gets large, $\mu^{c(N)}$ and $\mu^{*(N)}$ are like each other except on sets of small measure.

3.7.3 Lemma

For all ω (of length N)

$$\mu^{+(N)}(\omega) = P\, p_1^{M}\, p_2\, \mu^{c(N)}(\omega)$$

where M+1 is the number of complete words generated from the original (non closure) grammar in ω , and P is a normalising factor.

Proof: $\mu^{+(N)}$ and $\mu^{c(N)}$ are the same set and their words are generated in the same way except that when one of the closure productions is used the preprobability of $\Omega^{+(N)}$ is p_1 and for $\Omega^{c(N)}$ is 1.

3.7.4 Lemma

$$P = \frac{1}{\sum_{\Omega^{c(N)}} p_1^{M}\, p_2\, \mu^{c(N)}(\omega)}$$

Proof: By summing both sides in the expression of lemma [3.7.3] and algebraic manipulation. (Using the fact that $\mu^{+(N)}(\Omega^{+(N)}) = 1.$)

3.7.5 Lemma

$$P \leqslant \frac{1}{p_2\, p_1^{N}}$$

Proof: As M is the number of complete subwords in ω and as each subword is of length $\geqslant 1$, $M \leqslant N$. Hence as $p_1 < 1$, $p_1^{N} \leqslant p_1^{M}$

$$\text{so} \quad P = \frac{1}{\sum_{\Omega^{c(N)}} p_2 p_1^{M} \mu^{c(N)}(\omega)} \leqslant \frac{1}{\sum_{\Omega^{c(N)}} p_2 p_1^{N}\, c(N)(\omega)} = \frac{1}{p_2 p_1^{N}}$$

3.7.6 Lemma

$$\lim_{p_2 \to 1} \mathcal{H}_5^{+} = \limsup_{N \to \infty} - \sum_{\omega \in \Omega^{c(N)}} \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega) \,/\, N$$

Proof: By definition

$$\mathcal{K}_5^* = \operatorname*{Lim\ sup}_{N\to\infty} -\sum_{\omega\in\Omega^{*(N)}} \mu^{*(N)}(\omega)\log \mu^{*(N)}(\omega)/N$$

So using lemma [3.7.3] to substitute for the second occurrence of $\mu^{*(N)}(\omega)$ and splitting the resultant log of a product to a sum of logs, there is obtained the equation:

$$\mathcal{K}_5^* = \operatorname*{Lim\ sup}_{N\to\infty} T_1 + \operatorname*{Lim\ sup}_{N\to\infty} T_2 + \operatorname*{Lim\ sup}_{N\to\infty} T_3$$

where $T_1 = -\sum_{\omega\in\Omega^{*(N)}} (\mu^{*(N)}(\omega)\log \mu^{c(N)}(\omega))/N$

$$T_2 = -\sum_{\omega\in\Omega^{*(N)}} (\mu^{*(N)}(\omega)\log (Pp_2))/N$$

and $$T_3 = -\sum_{\omega\in\Omega^{*(N)}} (\mu^{*(N)}(\omega)\log (p_1)\ M)/N$$

Each of the above expressions can be further reduced (T_2 and T_3 first because they are easiest).

For $T_2 = -\log Pp_2/N \sum_{\omega\in\Omega^{*(N)}} \mu^{*(N)}(\omega) = -\log Pp_2/N.$

Similarly,

$$T_3 \leqslant -\log p_1 \sum_{\omega\in\Omega^{*(N)}} \mu^{*(N)}(\omega)\,(N/N) = -\log p_1$$

So $T_3 = k_1 \log p_1$ where k_1 is some parameter $|k_1| \leqslant 1$.

Finally T_1 will be reduced. By the extension [3.6.57, 3.6.58] of Macmillan's theorem, given $\epsilon > 0$ there is an $X < 1$ such that for all sufficiently large N, the set $\Omega^{c(N)}$ partitions into two sequences of sets, a sequence of high probability (with respect to the measure $\mu^{c(N)}$) sets E_N, such that $|\log \mu^{c(N)}(\omega)/N + H| < \epsilon$ for every one of its sequences, and the sequence of low probability

remainders $\bar{E}_N = \Omega^{c(N)} - E_N$, such that

$$-\sum_{\omega \in \bar{E}_N} \log \mu^{c(N)}(\omega)\ \mu^{c(N)}(\omega)/N < x^N$$

and

$$\mu^{c(N)}(\bar{E}_N) < x^N.$$

In the above

$$H = \operatorname*{Lim\ sup}_{N \to \infty} -\sum_{\omega \in \Omega^{c(N)}} \mu^{c(N)} \log \mu^{c(N)}(\omega)/N$$

is the entropy of the process.

By the ordinary version of Macmillan's theorem,

$$-\log \mu^{c(N)}(\omega) = H + \alpha\epsilon \qquad \text{on } E_N$$

(where α is a function of ω but $|\alpha| \leqslant 1$ for all $\omega \in E_N$).

The expression for T_1 can therefore be split into two parts

$$T_1 = T_4 + T_5$$

where $T_4 = \sum_{E_N} \mu^{+(N)}(\omega)\ (H + \alpha\epsilon)$

$$T_5 = -\sum_{\bar{E}_N} (\mu^{+(N)}(\omega) \log \mu^{c(N)}(\omega))/N$$

Now because $E_N \cup \bar{E}_N = \Omega^{+(N)}$ and $\mu^{+(N)}(\Omega^{+(N)}) = 1$,

$$T_4 = H - H\sum_{\bar{E}_N} \mu^{+(N)}(\omega) + \epsilon \sum_{E_N} \mu^{+(N)}(\omega)\alpha$$

The middle term in the above sum has the property that

$$\left| H \sum_{\bar{E}_N} P\ p_1^M\ p_2\ \mu^{c(N)}(\omega) \right| \leqslant H \frac{1}{p_1^N} \sum_{\bar{E}_N} \mu^{c(N)}(\omega)$$ (expanding P by lemma [3.7.4])

$$\leqslant H \frac{x^N}{p_1^N}$$ (by the extension of Macmillan's theorem)

So the middle term may be written as

$$k_2 \, H \frac{X^N}{p_1^N} \qquad \text{where} \quad |k_2| \leqslant 1.$$

The final term in the sum for T_4 is a product of ϵ and a sum of terms less than one, so may be written $k_3\epsilon$ where $|k_3| \leqslant 1$.

Finally T_5 can be reduced by substituting μ^+ for μ^c to yield

$$T_5 = -\sum_{\overline{E}} P \, p_1^M \, p_2 \, \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N$$

so expanding P

$$T_5 \leqslant -\sum_{\overline{E}} \frac{p_1^M}{p_1^N} \, \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N$$

$$\leqslant \frac{X^N}{p_1^N} \quad \text{by the extension of Macmillan's theorem.}$$

So $T_4 = k_4 \frac{X^N}{p_1^N}$ where $|k_4| \leqslant 1$.

So gathering together all the terms,

$$\mathcal{R}_5^+ = \operatorname*{Lim\,sup}_{N\to\infty} \left(H + k_2 H \frac{X^N}{p_1^N} + k_3\epsilon + k_4 \frac{X^N}{p_1^N}\right)$$

$$+ \operatorname*{Lim\,sup}_{N\to\infty} \left(-\log \frac{Pp_2}{N}\right)$$

$$+ \operatorname*{Lim\,sup}_{N\to\infty} (k_1 \log p_1)$$

So $\mathcal{R}_5^+ = H + \epsilon \left(\operatorname*{Lim\,sup}_{N\to\infty} k_3\right) + 0 + \log p_1 \left(\operatorname*{Lim\,sup}_{N\to\infty} k_1\right)$

so long as $p_1 > X$.

As $\operatorname*{Lim}_{p_1\to 1} (\log p_1) = 0$, and ϵ may be chosen to be arbitrarily small,

$$\operatorname*{Lim}_{p_1\to 1} \mathcal{R}_5^+ = H.$$

Or in other words using the definition of H

$$\lim_{p_1 \to 1} \mathcal{R}_5^+ = \limsup_{N \to \infty} - \sum_{\omega \in \Omega^{c(N)}} \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N.$$

3.7.7 Theorem

$$\mathcal{R}_4 = \lim_{p_1 \to 1} \mathcal{R}_5^+$$

Proof: By lemmas 3.7.1 and 3.7.6 both sides are equal to

$$\limsup_{N \to \infty} - \sum_{\omega \in \Omega^{c(N)}} \mu^{c(N)}(\omega) \log \mu^{c(N)}(\omega)/N.$$

3.7.8 Theorem

$$\mathcal{R}_4 = \frac{g(1)}{f'(1)}$$

Proof: By the above theorem and the theorem 3.5.7

$$\mathcal{R}_4 = \lim_{p_1 \to 1} \mathcal{R}_5^+ = \lim_{p_1 \to 1} \limsup_{N \to \infty} \left(\frac{q_N^+}{N p_N^+} + \frac{\log p_N^+}{N} \right)$$

By theorems 3.2.11 and 3.2.12,

$$\frac{q_N^+}{N p_N^+} = \frac{-(N+1)p_2(p_2 \log p_1 f^2(z_o) - g(z_o)) z_o^{N+1}(-p_1)f'(z_o)}{z_o^{N+2} p_1^2 (f'(z_o))^2 N (-p_2 f(z_o))}$$

$$= \frac{(N+1)(g(z_o) - p_2 \log p_1 f^2(z_o))}{N \cdot z_o \cdot p_1 f'(z_o)}$$

$$\text{So } \limsup_{N \to \infty} \frac{q_N^+}{N p_N^+} = \frac{g(z_o) - p_2 \log p_1 f^2(z_o)}{z_o p_1 f'(z_o)}$$

$$\frac{\log p_N^{\dagger}}{N} = \frac{\log (p_2 f(z_o))}{N} - \frac{N \log z_o}{N} - \frac{\log z_o p_1 f'(z_o)}{N}$$

So $\underset{N \to \infty}{\text{Lim sup}} \frac{\log p_N^{\dagger}}{N} = - \log z_o .$

Now, as $p_1 \to 1$, then $z_o \to 1$ and $p_2 \to 0$ so

$$\underset{p_1 \to 1}{\text{Lim}} \; \underset{N \to \infty}{\text{Lim sup}} \left(\frac{q_N^{\dagger}}{N p_N^{\dagger}} + \frac{\log p_N^{\dagger}}{N} \right) = \frac{g(1)-0}{f'(1)} - 0 = \frac{g(1)}{f'(1)}$$

Hence using the first equation in this proof

$$\mathcal{R}_4 = \frac{g(1)}{f'(1)}$$

3.7.8 Remark

Much of this chapter has been devoted to obtaining the previous result, and its proof has involved a tortuous route and a good deal of difficult mathematics. But now that the result has been justified it is very simple to use, for theorems 2.3.3.2 and 2.4.3.2 show how $g(1)$ and $f'(1)$ can be calculated merely by solving linear equations with real coefficients.

Chapter 4

PARSING

It is a simple matter to generate a terminal string from a parse [chapter 1]. The more difficult parsing problem is the inverse of this, that is, given a string, construct a parse which will generate it. This problem is important because any computer programme written in a high level language must first be translated into a machine language before it can be run, and one of the steps of the translation process is to produce some representation of the parse of the input programme. The parsing problem is also intrinsically interesting theoretically.

For the above two reasons the parsing problem has been intensively studied, and many different parsing algorithms have been found [6,8,21,35]. As time has passed, the algorithms have become more general and more powerful, and their descriptions more simple, but most are clearly variants of one of the three basic parsing algorithms which are studied here: top-down; bottom-up; or precedence. Versions of these three basic parsing methods are also used in compilers, although sometimes two are used at a time, and they are often supplemented by special shortcuts which can be used because the compiler only needs to work on one grammar, that of the language being compiled.

Once the problem of finding one solution to the parsing problem is solved, another problem can be tackled, that of measuring (absolutely) the behaviour of an algorithm, or of comparing (relatively) the effects of two or more parsing methods.

On the theoretical side the usual measurement is a worst case

analysis which shows what is the maximum amount of time and/or space which a parser needs to analyse any string. Most parsing methods only work on some grammars, and simple tests may be presented to decide whether a particular method will work on a particular grammar. Different parsing methods may be compared in respect of the size of the set of grammars they work on.

On the practical side the effects of a parser are difficult to disentangle from those of the rest of the compiler in which it is embedded. But there are three important ones. First, the orders in which the input string is scanned and in which the parse is generated. If these correspond with the order in which the input string is read by the compiler, and the order in which the parts of the parse are needed by the next stage of the compiler after the parser, then only a small part of the input string and of the parse need be stored by the parser at any one time, and so a lot of space is saved. However, whether or not such a one pass compiler can be built is often more a function of the semantics of the input and output languages than of the choice of parser.

The second effect of the choice of parser is relatively independent of the rest of the compiler. That is, the average or typical amount of space or time needed to analyse a string. The third effect is that some parsing methods allow better error detection than others.

This chapter is intended to help the designer of a compiler to calculate the second of the above effects before writing his compiler. Unfortunately the results have turned out to be rather negative. In a reasonable sense, all deterministic parsers require the same amount of time and space. Any backtracking parser requires more, but it is not possible to give a general algorithm which

calculates how much more. For restricted classes of grammars and parsers it is possible to give an algorithm, but the restrictions are of a theoretical kind and the calculations are very complicated.

4.1 PARSING METHODS: THE DOMINO GAME

The parsing methods will be described within a uniform framework which allows them to be easily compared. They are described in informal physical terms as a solitaire version of a type of game of dominoes. This (hopefully) makes them easy to understand, but does not result in a loss of rigour, because the description could, with effort but not intellectual difficulty, be translated into graph theory.

The general idea of the game is to start with the start symbol at the top of the board and symbols representing the input string at the bottom. An attempt is made to build a bridge of dominoes from one to the other. If it succeeds the final bridge represents the parse.

4.1.1 The Physical Apparatus for the Domino Game

The game is specified relative to a particular grammar (N,T,P,S), so different grammars correspond to different versions of the game.

The board

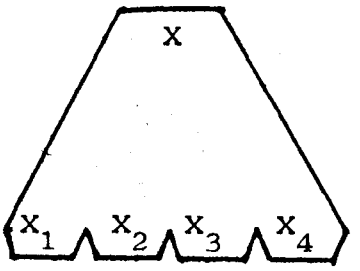
The board is a two-dimensional flat area with a top, bottom, left and right sides.

The pieces

There are three kinds of piece. They are all stretchable as is necessary, but may not be rotated, folded nor reflected. When placed on the board each occupies a definite area of space, none of which can be shared with another piece.

Domino

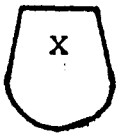
Corresponding to each production $\langle X \rightarrow X_1 X_2 \ldots X_n \rangle$ there is an as large as needed set of identical dominoes. Each domino has one top edge marked with an X, and n bottom edges marked with X_1, X_2, $\ldots, X_n$ in left to right order (see diagram below). For some parsing strategies the dominoes may have a finite number of spaces into which a finite amount of erasable information may be written.



A domino corresponding to the production $(X \rightarrow X_1 X_2 X_3 X_4)$

Input piece

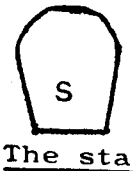
Corresponding to each terminal symbol $X \in T$ there is an as large as needed set of identical input pieces. Each input is marked on the top edge with an X. The bottom edge is ignored. Again there may be space for a finite amount of erasable information.



An input piece for the terminal X

Start piece

There is one start piece. It has the start symbol S on its bottom edge. The top edge is ignored. There may be space for a finite amount of erasable information.

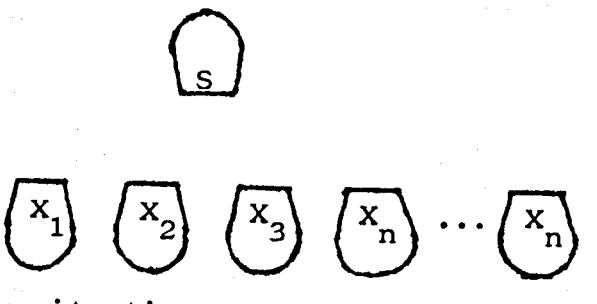


The start piece

4.1.2 The Aim of the Game

The aim of the game is to change a pattern of pieces on the board which represent an input string to another pattern which represents a parse of that string, using only allowable rules to change from one pattern to the next.

The board is set up to represent a particular input string which is to be parsed. If $X_1X_2...X_n$ is the string to be parsed then the board is set up with the start piece at the top, and input pieces corresponding to $X_1, X_2, ...X_n$ in left to right order across the bottom.



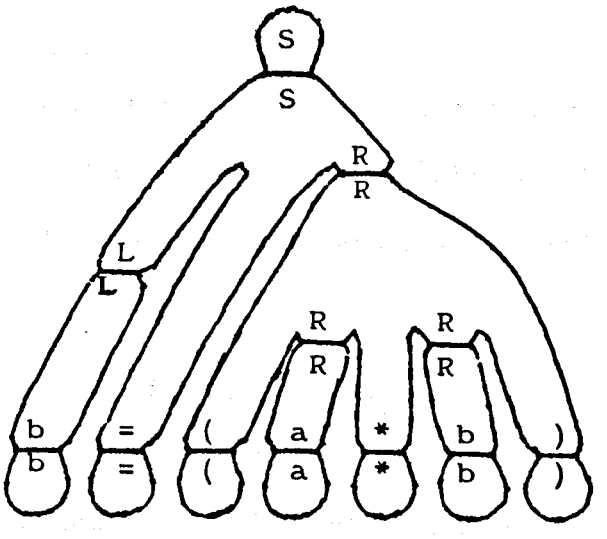
Starting situation corresponding to the input $X_1X_2...X_n$

Finish of play

There are three different ways that play can finish.

1. Successful finish

Every top edge is matched against a bottom edge and every bottom edge against a top edge. In particular the bottom edge of the start piece and all the top edges of the input pieces are matched. The parse of the input string may easily be obtained from the pattern of dominoes (see diagram).



Example of a successful finish for the game corresponding to the language of assignment statements and input string 'b=(a * b)'

2. Unsuccessful finish

The parser decides that it cannot find a parse and so stops. This can happen for one of two reasons.

There is no parse, that is the input is not in the language generated by the grammar G.

There is a parse, but the parser has failed to find it.

3. Non-termination

It is also possible that neither of the above situations is reached. In this case play goes on for ever.

4.1.3 Rules of Play

There are three types of move.

Placing a domino

Put a new domino on the board, matching one or more of its edges against edges of pieces already on the board. Top edges must be abutted to bottom edges and bottom edges to top, and of course matching symbols must be marked with the same symbol.

Joining two dominoes

This is a variant of the above. Stretch two dominoes already on the board so that the top edge of one abuts a bottom edge of the other, where both edges have the same marking symbol.

Backtracking

Take some dominoes off the board. There are often restrictions as to exactly which dominoes may be removed. These restrictions will be explained in the course of the explanation of particular parsing methods. The general rule is that the pattern of dominoes and their connections should be the same after a backtracking move as it was at some earlier stage of the parsing.

Examples

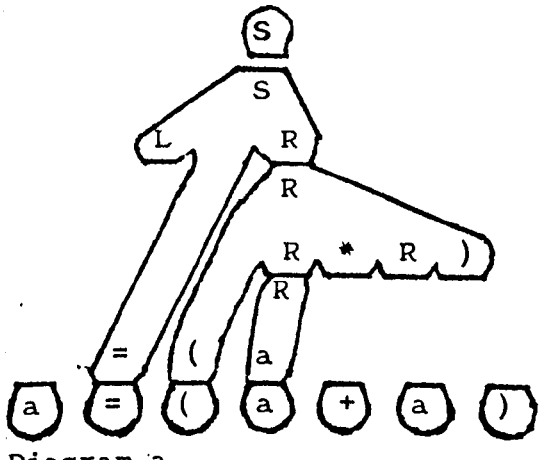


Diagram a

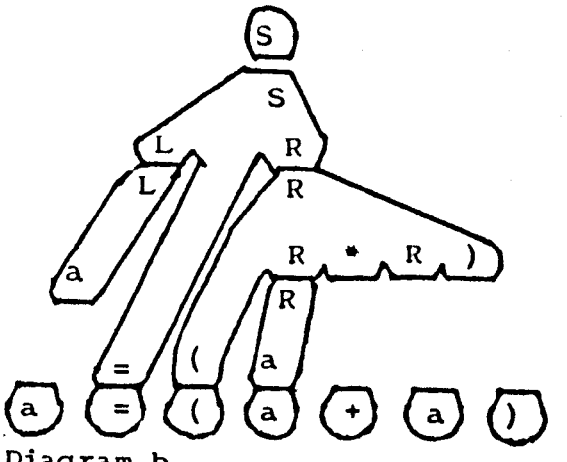


Diagram b

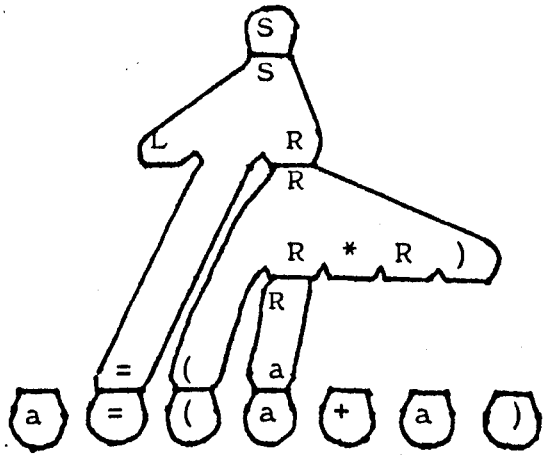


Diagram c

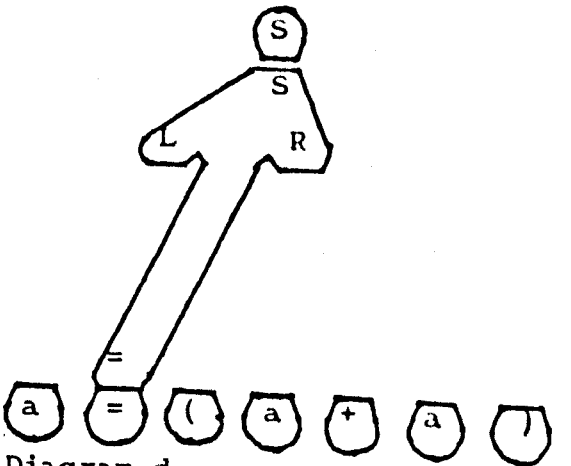


Diagram d

The above four diagrams show situations which might arise during parsing. b, c and d are alternative situations which might follow situation a. b is obtained from a by the first type of move, that is, the domino corresponding to ⟨L→a⟩ is abutted to the bottom L of the domino corresponding to ⟨S→L=R⟩. c is obtained from a by the second type of move, that is the domino corresponding to ⟨S→L=R⟩ is stretched so that its top edge S abutts the start piece. d is obtained from a by the third type of move, backtracking, that is dominoes corresponding to ⟨R→(R*R)⟩ and ⟨R→a⟩ are deleted.

Remark: Other types of move

Other moves are conceivable. For instance, two are: place a domino on the table but do not immediately abutt any of its edges to any piece already on the table; adjust two adjoining dominoes so that they no longer abutt each other, but leave both on the table. Only the three named types of move are used in the three types of parsing algorithm considered here.

4.1.4 Definitions for Describing Situations

All parsing methods progress from situation to situation in a discrete manner. So to analyse them it is necessary to consider the situations in detail. The following definitions give names to particular parts of situations and also help to illuminate their structures.

Definition

A situation is a pattern of pieces and dominoes on the table.

A start situation is one where there are just the input pieces and start piece on the board. The start piece is at the top, and the input pieces in left to right order across the bottom.

A final situation is one where no piece on the table has an unmatched edge.

Definition

A domino or piece is directly connected to another domino or piece if it abutts it.

A domino or piece A_1 is indirectly connected to another A_2 if either $A_1 = A_2$ or A_1 is directly connected to A_2, or there is a chain $A_1 = X_1, X_2, \ldots, X_n = A_2$ such that each X_i is directly connected to X_{i+1}.

Remarks

Input pieces are never directly connected to each other. In the start situation they are not indirectly connected either, but if a final situation is reached every pair of pieces is indirectly connected.

The relation of being indirectly connected to is an equivalence relation.

Lemma

In any situation obtained while parsing by any algorithm using just the three rules (placing, joining, and backtracking), every domino is either indirectly connected to the start piece, or to an input piece (or both).

Proof: By induction. The above statement is clearly true of a starting situation. If it is true in a situation before a placing or joining move, then it is true after. The situation after a backtracking move has the same pattern of connections as some earlier situation, and so has the above property by complete induction.

Definition

An edge is said to be free if it does not abutt another. Otherwise it is matched.

Definitions

A piece or domino is said to be ceilinged if it is indirectly connected to the start piece.

An edge is said to be ceilinged if its piece or domino is ceilinged.

Definition

A ceilinged free bottom edge (cfbe) is a bottom edge which is free and ceilinged.

Lemma

The left to right order of the (cfbe)s is a total order.

Proof: The set of all ceilinged dominoes forms a tree with the domino abutting the start piece as root. Any pair of (cfbe)s can be compared in left to right order by whether one's branch is to the left of the other's or not. If two different (cfbe)s were equal in the ordering, then they would share the same branch and so one could not be free.

Definition

An input piece covers itself. A domino covers all those input pieces which are covered by pieces or dominoes abutted to its bottom edges.

A top edge covers those pieces which are covered by its domino.

Remark

Thus a domino (domino's top edge) covers all those input pieces

which can be reached from itself through a descending chain of dominoes.

Definition

A piece or domino is fully grounded if either it is an input piece or else it is a domino, all its bottom edges abutt fully grounded pieces or dominoes; and also the set of all the input pieces which it covers is a gap-free subsequence of all the input pieces in left to right order.

A top edge is fully grounded if its domino is fully grounded.

Definition

A fully grounded free top edge (fgfte) is a top edge which is free and fully grounded.

Lemma

The left to right order of the (fgfte)s is a total order.

Proof: The set of dominoes from an (fgfte) form a tree. It is impossible for an input piece to be covered by two different free top edges because it is impossible for two trees from different roots to share a descendant. Hence the gap-free subsequences covered by (fgfte)s are disjoint. The (fgfte)s inherit the total order of the subsequences.

Remark

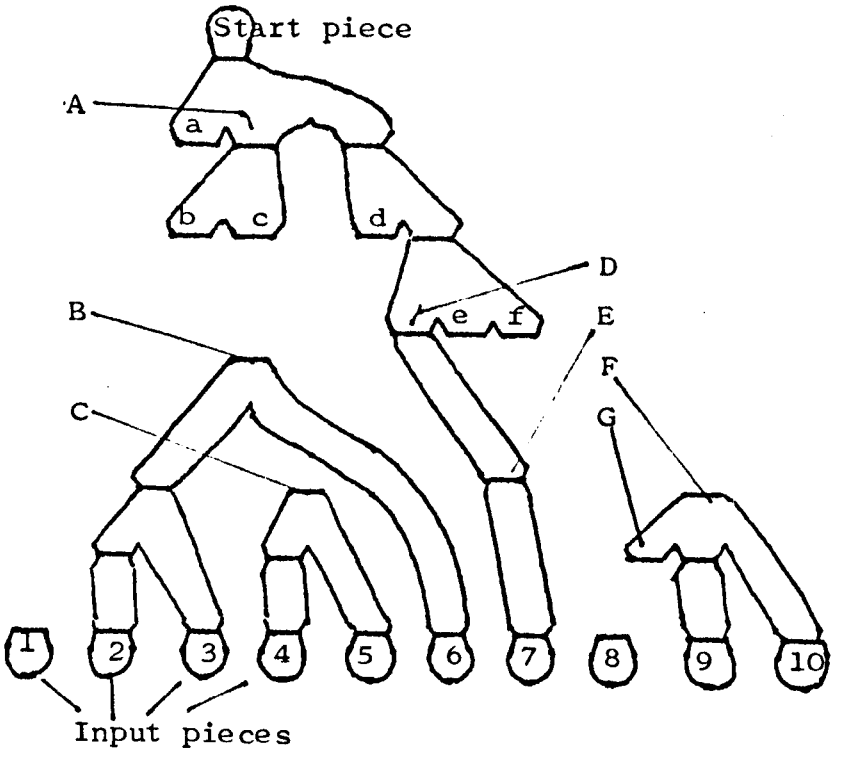
The above two lemmas about left to right orders are not as obvious as they may seem. The next diagram shows some of the odd patterns which have been excluded by the definitions.

The (cfbe)s in order are a,b,c,d,e,f. Two ceilinged bottom edges may only be on the same branch to the root if at least one is matched. For example A,b and D,E are two such pairs. It is often

difficult to compare the position of non-ceilinged bottom edges, for instance to decide whether F is to the left of f or not.

The (fgfte)s in left to right order are 1, C, 8. Not all input pieces are covered by (fgfte)s. B is not an (fgfbe) because the sequence of input pieces covered by it contains a gap. Luckily a relationship like that between the dominoes containing B and C, where C is a free top edge but has no bottom edge to abutt against, can never occur during any of the three standard kinds of parsing algorithm.

Diagram



Definition

The above two lemmas show that it makes sense to talk about the leftmost and rightmost cfbe and fgfte.

Definition

The effective input of a situation is the sequence of (fgfte)s in left to right order.

Remark

The grammars dealt with here are unambiguous, so there is a unique pattern of dominoes which represents the only parse of any correct input string. In order for the next definition to make sense it must be imagined that this unique finishing pattern of dominoes is drawn on the table. The parsing strategies try to build this final parse by following strict rules of what to do next. As the rules do not take into account what is drawn on the table, the order of the situation through which a strategy goes is unaffected by this pattern. But an observer is able to take both the final pattern and the pattern reached at some point in time into account together and he can in particular compare them.

Definition

A domino is correct if when it is placed on the table, it and all the dominoes and pieces to which it is directly or indirectly connected form a subpattern of the final pattern.

Otherwise a domino is incorrect.

In particular, if a domino is incorrect, but when it was placed all the dominoes it abutted were correct, then that domino is first incorrect.

A domino remains correct, incorrect or first incorrect according to what it was at the instant it was placed on the table.

Remark

The above definitions may seem a little odd for it is not possible to know whether or not a particular domino is correct without

first constructing a complete parse of the input string. But once this parse has been built, why use any parsing method at all?

They are less odd than they appear. Because the language of all input strings is a recursive set and because the function from input strings to their parses is constructive (using British Museum Algorithm as a last resort [11]), it is always decidable in a finite time whether or not a particular domino is correct or not. In practice, it is easy to see with hindsight which dominoes were correct after a parsing algorithm has terminated. Finally, in general it is of no interest which particular dominoes are correct; what is wanted is a way of distinguishing different situations which arise during parsing.

But the definitions are odd. They only make sense for strings which can be generated by the grammar to be parsed. It is possible to consider what would happen if a parser is given as input a string outside the language to be parsed. An analysis of a parsing algorithm in terms of correctness will not extend to this case because such a string has no correct final parse.

The intention behind the definitions is that if all the dominoes on the board are correct then it is possible to build up to a complete parse by using placing moves only; it is not necessary to use back-tracking or adjoining. There are situations in which this situation is not exemplified: for instance after two correct dominoes have been abutted by incorrect edges. These nasty situations do not make the analysis of parsing methods as difficult as they might because once one arises during an attempt at parsing by any of the three types of algorithm, the attempt is doomed to failure.

4.2 PARSING STRATEGIES IN DETAIL

The domino game tightly restricts the manner in which parsing may be done. The parsing strategies complete the specifications by saying exactly what must be done to progress from one situation to the next. The three strategies have in common that they decide the next move by the answers to three questions:

(1) Which free edge(s) should be dealt with next?

(2) Which kind of move should be used on that edge (those edges)?

(3) Which of the examples of the type of move selected in question 2 should be used?

The three strategies differ firstly in the answers they give to the three questions. Second. A reason for describing parsing as a game of dominoes is to make the strategies as similar (and hence as easily comparable) as possible. But most of the strategies require extra information besides just the pattern of the dominoes on the board. In general only two kinds of extra information are needed. Firstly a table containing permanent information which can be worked out once for all before any attempts to parse any input strings are made. And secondly, small amounts of updatable information attached to individual pieces and dominoes. The tables and updatable information are different for different parsing methods.

Remark

A couple of definitions will now be made which refer to and make sense for all parsing methods.

Definition

A parser is universally successful on a grammar if it succeeds in parsing all strings of its language, otherwise it is partially successful.

Definition

A parser is deterministic on a grammar if it never uses a type 3 (backtracking) move when parsing a string of its language.

Remark

A new deterministic parsing algorithm can be obtained from any given algorithm simply by disallowing backtracking, but in general the new algorithm will succeed in parsing relatively few strings.

4.2.1 Left to Right Top Down Parsing

There is an infinite sequence of variants of this basic parsing strategy. They are called LR(k) top down parsing, where k is a non-negative integer. LR(o) parsing is also called pure LR parsing.

This method of parsing is widely used in compilers, but often with a stack. (The method of recursive descent is this method using an implicit stack.) In general LR(o) and LR(1) parsing is used. The key initial paper which originally analysed and laid out these methods is [21]

The table for top-down parsing

A table is a function with finite domain. In top-down parsing the table is a function

$$t: \{\langle N,\alpha\rangle\} \rightarrow P^*$$

where N is a non-terminal, α a string of k terminals, and P* the set of finite sequences of productions (including the empty sequence). In practice only non-recurrent sequences of productions are needed. In words, t maps a pair consisting of a non-terminal and a string of k terminals to a finite (non-recurrent) sequence of productions. $t(N,\alpha)$ is the string corresponding to $\langle N,\alpha\rangle$, and $t(N,\alpha)_i$ the i^{th} production in the sequence so long as $t(N,\alpha)$ has i

or more productions. If it does not then $t(N,\alpha)_i$ is without denotation.

Defining property of the table

Let p be a production. Then $p \in t(N,\alpha)$ if and only if there is a generation of the form

either $S \overset{*}{\Rightarrow} xNz \overset{*}{\Rightarrow} x\alpha\beta z$

or $S \overset{*}{\Rightarrow} xNyz \overset{*}{\Rightarrow} x\omega yz$ where $\omega y = \alpha$

and in both cases N is expanded by p.

In words, $p \in t(N,\alpha)$ iff it can (in context) produce a terminal string starting with α .

The property is sufficient to specify the set of the productions in $t(N,\alpha)$ for any N and α , and this set can be calculated with finite effort (see [21]), but the order of the productions within this set is not specified and can be freely chosen. Different orders give algorithms with (sometimes) different properties. The algorithms for the same grammar but with different values of k can be related by the following theorem and definition.

Theorem

If $k_1 \leq k_2$ and α_i is a terminal string of length $k_i (i=1,2)$ and α_1 is an initial subsequence of α_2, and N is any non-terminal, then

$$t(N,\alpha_1) \supseteq t(N,\alpha_2)$$

(where $t(N,\alpha_i)$ is considered as a set).

Proof: Follows easily from the defining property of $t(N,\alpha_i)$.

Definition

If in addition to the above the order of the productions within $t(N,\alpha_2)$ is a suborder of the productions within $t(N,\alpha_1)$ for all

possible pairs $\langle N, \alpha_2 \rangle$, then the two parsing algorithms are said to be compatible.

Updatable information on the dominoes

Every bottom edge marked with a non-terminal, including that of the start piece, is associated with a space which can hold a non-negative integer.

The parsing strategy

In outline, an attempt is made to abutt something to the leftmost ceilinged free bottom edge. This edge will therefore be called the active edge. If the attempt fails then the algorithm backtracks.

In detail. If the active edge is marked by a terminal and the leftmost fully grounded free top edge (fgfte) is marked by the same terminal then the two edges are abutted using rule 2. If the (fgfte) is marked with a different symbol then backtracking occurs.

If the active edge is marked by a non-terminal N then its associated integer is increased by 1 to yield i. The symbols marking the leftmost k(fgfte)s in order yield the sequence α . If $t(N, \alpha)_i$ exists then the domino corresponding to that production is abutted to the active edge. All the integers associated with non-terminal bottom edges of the new domino are initially set to zero. If $t(N, \alpha)_i$ does not exist then the algorithm backtracks.

When the algorithm has to backtrack the domino containing the active edge is removed, and in addition any dominoes which directly or indirectly hang beneath it.

(Technical problems are that there may be no (fgfte) in a situation when rule 2 is attempted; in this case the match fails. There may be too few (less than k) (fgfte)s when rule 1 is attempted;

in this case the terminals on the (fgfte)s which do exist form the initial substring of α and the rest is padded out with a dummy symbol. The table function $t(N, \alpha)$ has to allow for these dummies.)

The strategy succeeds if there are no unmatched edges. It can fail in two different ways. There are unmatched top edges but no (cfbe) to be chosen as the next active edge. Alternatively backtracking fails. This happens when the active edge which ought to be removed is the bottom edge of the start piece. A final possibility is that a parsing strategy neither succeeds nor fails but goes on for ever.

Properties of the LR top-down parsing strategy

(1) Backtracking does return the pattern of connections between dominoes to that of an earlier situation, the only change is that the integer associated with some non-terminal edge is increased.

(2) In any situation reached during parsing, all the top edges of input pieces to the left of some position are matched, all those to the right are not matched, and these unmatched edges of input pieces are exactly the set of all the (fgfte)s.

(3) An LR(k) top-down parsing strategy is deterministic and universally successful if and only if every sequence $t(N, \alpha)$ contains at most one element.

Proofs:

The first property can be deduced because once a domino A is placed on the board, all subsequent dominoes are placed directly or indirectly beneath A until A is fully grounded.

The second can be proved by induction.

The third also by induction. If $t(N, \alpha)$ contains only one production then there is only one domino which may be placed against

the active edge in each situation. This domino must be correct because a final parse exists. If $t(N, \alpha)$ contains two productions then sometimes the wrong domino must be chosen first. If this incorrect domino is not removed the parsing strategy cannot succeed; if it is removed the strategy must have backtracked.

Remark

Whether or not all the sequences $t(N, \alpha)$ contain exactly one production is only a function of the grammar and the length of α. So the following definition makes sense: a grammar is LR(k) if it is deterministically parsable by a top-down LR(k) parser, but not deterministically parsable by any LR(k') parser for $k' < k$.

4.2.2 Left to Right Bottom-Up Parsing

This is similar to top-down parsing. I do not know that it has been used in practice. It was first described in INGERMAN [18] but with a few loose ends. The reason for both these facts should be clear by the end of this description.

The table for bottom-up parsing

This is similar to the function for top-down parsing and is a function

$$t: \{\langle N, \alpha \rangle\} \rightarrow P^*$$

where N, and P^* are as before, but this time α varies over sequences of length k, where the first symbol may now be a non-terminal. $t(N, \alpha)_i$ has its previous meaning.

Defining property of the table

If p is a production then $p \in t(N, \alpha)$ if and only if

either $S \overset{*}{\Rightarrow} xNy \overset{*}{\Rightarrow} x\alpha\beta y$

or $S \overset{*}{\Rightarrow} xNyz \overset{*}{\Rightarrow} xwyz$ (where $wy = \alpha$)

and the leftmost symbol of the right hand side of p is that which gives rise to the first symbol of α .

Theorem

If $k_1 \leq k_2$ and α_i is a string of symbols of length $k_i (i=1,2)$ whose first symbol may be terminal or non-terminal but the remainder of which is terminal, and α_1 is an initial subsequence of α_2, and N is any non-terminal, then

$$t(N,\alpha_1) \supseteq t(N,\alpha_2).$$

Proof: From defining property of $t(N,\alpha)$.

Definition

If in addition to the above the order of the productions within $p(N,\alpha_2)$ is a suborder of the productions within $p(N,\alpha_1)$ for all possible pairs $\langle N,\alpha_2 \rangle$ then the two parsing algorithms are said to be compatible.

Remark

The defining property of the bottom-up table seems similar to that of the top-down table. An important difference is that in the bottom-up definition α varies over a larger set and so the domain of its function t is greater.

Updatable information on the dominoes

Every top edge marked with any symbol (including those of input pieces) has associated with it a space into which a non-negative integer may be written.

The Parsing Strategy

Definition

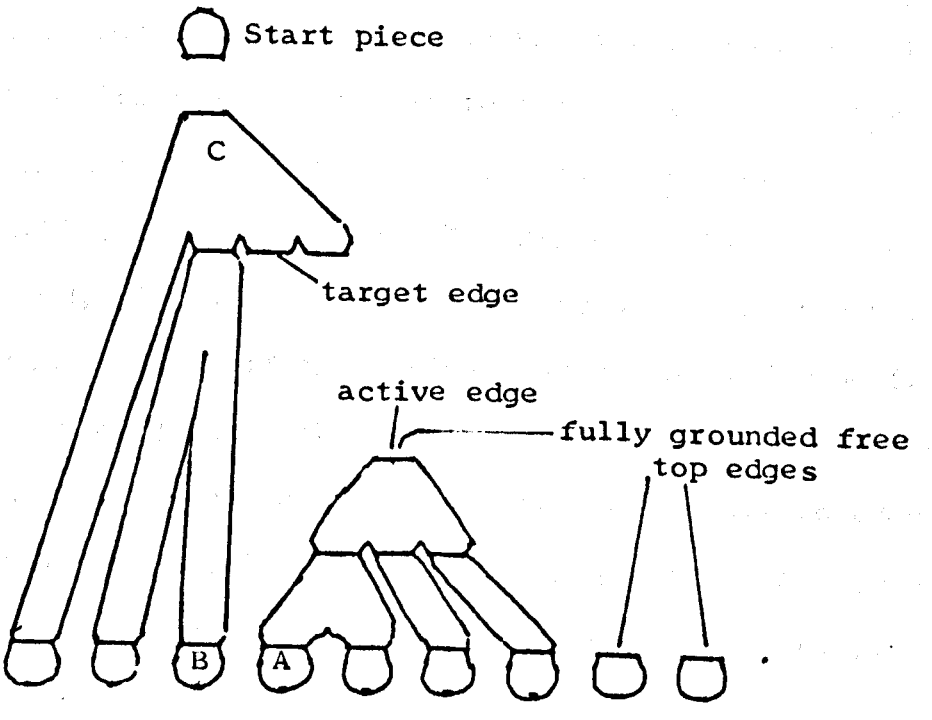
The active edge is the leftmost fully grounded free top edge.

Definition

The target edge is obtained from the active edge in the following way. Let A be the leftmost input piece covered by the active edge. If A is the leftmost of all the input pieces then the target edge is the bottom of the start piece. Otherwise there is another input piece B immediately to the left of A. The dominoes which cover B form an ascending chain. Let C be the lowest of these dominoes which is not fully grounded. The bottom edge of C which is immediately to the right of the edge on the chain from B is the target edge.

If all the dominoes which cover B are fully grounded or alternatively C exists but the bottom edge which covers B is the rightmost edge of C, then no target edge exists.

Diagram



The letters A,B,C mark the pieces used as auxiliaries in the definition of the target edge.

Remark

The definition of the target edge seems very complicated. But once the dominoes are laid out the target edge is easy to find. The rule is: start from the active edge, go down (keeping to the left) until an input piece is reached, go one piece left, finally up and as soon as possible one step right (see diagram).

The algorithm

In outline. An attempt is made to abutt something against the active edge. If the attempt fails the algorithm backtracks.

In detail. If the active edge and the target edge are marked with the same symbol then they are abutted. (A type 2 move is made.)

Otherwise the integer associated with the active edge is increased by 1 to yield i. The symbol marking the target edge is N. The symbols marking the leftmost k (fgfte)s in order yield the string α. If $t(N,\alpha)_i$ exists then a corresponding domino is laid on the table, its leftmost bottom edge is abutted to the active edge, and the integer marking its top edge is set to zero. (A type 1 move is made.) If $t(N,\alpha)_i$ does not exist, backtracking takes place.

Backtracking is complicated. The domino containing the target edge is removed, plus any dominoes directly or indirectly below any but its leftmost edge. If the active edge was on a domino (rather than an input piece), then that domino and all dominoes beneath it are removed. The integer associated with the new active edge (whether or not on an input piece) is left as it was, but the integers associated with all the (other) unmatched edges of input pieces are reset to zero.

(Technical problems are that there may be no free top edges but still unmatched bottom edges remaining; in this case a backtrack

is tried. The input may have to be padded out with dummies as in the top-down strategy.)

The strategy succeeds if there are no unmatched edges. It can fail in two different ways. There is an active edge but no target edge (this happens when all dominoes are fully grounded and ceilinged, but some input pieces remain). Alternatively, backtracking fails. This happens when the target edge which should be removed is part of the start piece. A final possibility is that the strategy neither succeeds nor fails but goes on for ever.

Properties of the LR bottom-up parsing strategy

Backtracking does return the pattern of connections between dominoes to that of an earlier situation, the only change is that the integer associated with the active edge is increased.

In any situation reached during parsing all the fully grounded free top edges except possibly the leftmost are the top edges of input pieces. The top edges of input pieces fall into three groups. Firstly to the left, matched edges which are not covered by the active edge. Secondly in the middle, edges covered by the active edge. Thirdly to the right, free edges. (Some of the groups may be empty.)

An LR(k) bottom-up parsing strategy is both deterministic and universally successful if and only if every sequence $t(N, \alpha)$ contains at most one element.

Proofs:

The first property can be deduced by defining the position of a domino to be the position of the rightmost input piece covered by its leftmost bottom edge. Dominoes are then never placed in a position to the left (using this definition) of a domino already

on the board. Backtracking removed the target domino A and all those dominoes to the right of A, and so returns the pattern of connections to that holding immediately before A was put on the board.

The second property can be proved by induction.

The third in the same way as for top-down parsing.

Remark

It should be clear why top-down parsing is more popular in compiler design than bottom-up.

4.2.3 Precedence Parsing

This is the second main kind of parsing method used in practice. It was developed from operator precedence parsing initially described in [8], was itself first described in [35] and has since been generalised.

Precedence parsing is different from top-down and bottom-up parsing because backtracking is not allowed. Because of this precedence parsing is only applicable to some grammars, called precedence grammars.

4.2.3.1 The table for precedence parsing

This is a function t from ordered pairs of symbols of the grammar to one of the four new symbols $\lessdot$, $\gtrdot$, $\doteq$, blank. In symbols,

$$t : (N \cup T) \times (N \cup T) \longrightarrow \{\lessdot, \gtrdot, \doteq, \ \}$$

4.2.3.2 Defining properties of the table

Some auxiliary definitions are needed.

Given a non-terminal X, its leftmost symbols $\mathcal{L}(X)$ are those symbols which can start a string generated from X.

$x \in \mathcal{L}(X)$ iff $X \overset{+}{\Rightarrow} x\alpha$ for some string α where x is a symbol. The rightmost symbols $\mathcal{R}(X)$ as similarly those which can terminate such a string.

$x \in \mathcal{R}(X)$ iff $X \overset{+}{\Rightarrow} \alpha x$ for some string α where x is a symbol.

These prerequisite definitions are used to help define the table t. In the statement of its properties α and β are possibly empty strings, x and y any two symbols (terminal or non-terminal), u and v any two non-terminals.

1) If there is a production $\langle X \rightarrow \alpha xy \beta \rangle$ then $t(x,y) = \doteq$.

2) If there is a production $\langle X \rightarrow \alpha xv \beta \rangle$ and $y \in \mathcal{L}(v)$ then $t(x,y) = \lessdot$.

3) If there is a production $\langle X \rightarrow \alpha uy \beta \rangle$ and $x \in \mathcal{R}(u)$ then $t(x,y) = \gtrdot$.

4) If there is a production $\langle X \rightarrow \alpha uv \beta \rangle$ and $x \in \mathcal{R}(u)$ and $y \in \mathcal{L}(v)$ then $t(x,y) = \gtrdot$.

5) If none of the above hold, then $t(x,y)$ = blank.

4.2.3.3 Definition

Depending on the grammar it is perfectly possible for two or more of the above properties to hold simultaneously and so contradict each other. For instance there may be two productions

$\langle X_1 \rightarrow \alpha_1 \, xy \, \beta_1 \rangle$ and $\langle X_2 \rightarrow \alpha_2 \, xv \, \beta_2 \rangle$ where $y \in \mathcal{L}(v)$,

so that by properties 1 and 2 above, $t(x,y) = \doteq$ but $t(x,y) = \lessdot$. The occurrence of this situation is said to be a precedence clash.

4.2.3.4 Definition

A grammar is a precedence grammar if it has the following two properties:

1) It has no precedence clashes so a precedence table exists.

2) No two right-hand sides of different productions are identical.

4.2.3.5 Remark

The above two conditions force there to be just one branch

at any point where the parsing algorithm might diverge and so prevent backtracking being necessary.

4.2.3.6 Updatable information on the dominoes

Every top edge (both of dominoes and of input pieces) has associated with it a space which can be blank or hold one of the two symbols $\lessdot$, $\doteq$. This symbol gives the relation between the symbol marking its top edge and the symbol immediately to the left of it in the effective input string.

The Parsing Strategy

4.2.3.7 Definition

A substring of the effective input is a phrase iff a correct domino may be abutted against its top edges, so that the bottom edges of the domino and the top edges of the substring match each other without gaps.

4.2.3.8 Definition

The handle of the effective string is the leftmost phrase.

4.2.3.9 Propositions

Any effective input (except that consisting of just the start symbol) has at least one phrase.

No two phrases overlap.

Any effective input has a handle.

Proofs: Any effective input may be obtained by generating a partial parse from the root. Then by induction, every node is either a tip, gives rise to a phrase directly or gives rise to a phrase indirectly.

This proves the first statement, the other two are easy consequences.

4.2.3.10 The algorithm

In outline. The handle of the effective input is located by using the precedence relations between its adjacent symbols. (The relation between the leftmost symbol of a phrase and the preceding symbol is $\lessdot$, between adjacent symbols of a phrase $\doteq$, and between the last symbol and the succeeding symbol $\gtrdot$. The only possible domino is matched against the handle. The process is repeated with the new effective input.

In detail. To start with, all the input pieces have their updatable information set to blank. The active piece is the leftmost input piece.

One step of the process. The relation between the active edge and the preceding top edge is looked up in the precedence table.

If the relation is $\doteq$ or $\lessdot$ then that symbol is written into the updatable space of the active edge. The next active edge is one place to the right of the old.

If the relation is blank an error has occurred, the input string is not in the language.

If the relation is $\gtrdot$ a reduction occurs. The parser leaves the symbol marking the active edge blank. The handle is now the sequence of top edges to the left of (and not including) the active edge, consisting of edges marked with $\doteq$ until the leftmost (included in the handle) is marked with $\lessdot$. There is only one domino which will fit this handle and this is abutted. The next active edge is the top edge of this new domino.

The procedure stops when the effective input consists solely of the start symbol.

Technical points are. If the active piece is the first in the input (there is no top edge to its left), then it is marked with a $\lessdot$

and the procedure continues as usual. If the end of the input is reached the parser behaves as if there were another dummy symbol to the right of the input and the relation between this dummy and the last symbol of the input was always $\gtrdot$.

4.2.3.11 Remark

For proofs that precedence parsing does work see [35].

4.2.3.12 Property of the parsing process

A small point that might cause some worry is that when the precedence relation is $\gtrdot$ the relevant updatable space is left blank. The reason for this is that information only needs to be stored if it is to be used again in the future. When the precedence relation is $\gtrdot$ this fact is used immediately to steer the algorithm but never again, hence it need not be noted down.

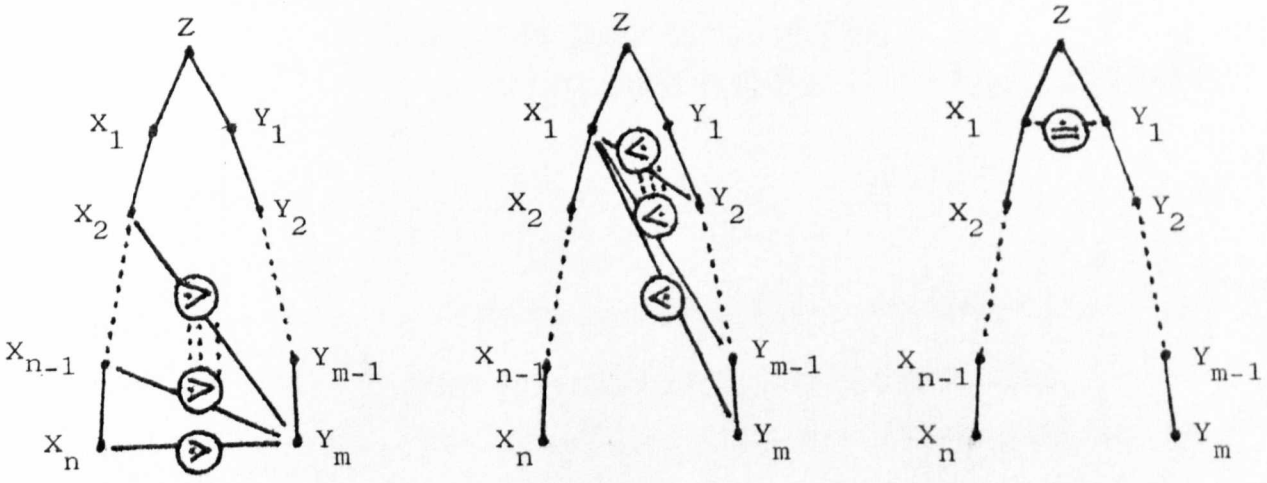
If one of the nodes of a parse is the symbol Z, if it generates its adjacent direct descendants X_1, Y_1 by the production $\langle Z \rightarrow \alpha_1 X_1 Y_2 \beta_2 \rangle$, if each X_i and Y_i generates its direct descendants by the productions $\langle X_i \rightarrow \alpha_{i+1} X_{i+1} \rangle$ $\langle Y_i \rightarrow Y_{i+1} \beta_{i+1} \rangle$ until the final terminal symbols X_n, Y_m are reached, then the order in which the symbols X_i and Y_j are found and matched against each other in the precedence table is

$$X_n \gtrdot Y_m, \quad X_{n-1} \gtrdot Y_m, \quad \ldots, \quad X_2 \gtrdot Y_m,$$

$$X_1 \lessdot Y_m, \quad X_1 \lessdot Y_{m-1}, \quad \ldots, \quad X_1 \lessdot Y_2, \quad X_1 \doteq Y_1$$

Finally, both X_1 and Y_1 are part of a handle which gets reduced to Z.

4.2.3.13 Diagram



First this group of relations is found from bottom to top.

Then this group from bottom to top.

Finally this relation.

Proof: This can be shown by induction by considering the details of the parsing algorithm and the construction of the precedence table.

4.2.3.14 Remark

Notice that the above theorem does not state that the order is gap free. In fact it is not, for the active node being used by the parser zigzags back and forth across the branches of the parse tree. Between occasions when the parser comes across an adjacent pair of symbols X, Y from any particular pair of branches it will in general work on pairs to the left or the right of these pairs.

4.2.3.15 Corollary

If a parse P is produced by the first production $\langle X \rightarrow x_1 x_2 \ \ldots \ x_n \rangle$ and then by expanding each of the x_i to produce a complete subparse p_i, and the number of times a precedence parser finds occurrences of the relation $\lessdot$, $\gtrdot$, $\doteq$ in a parse p is given by $r^{\lessdot}(p)$, $r^{\gtrdot}(p)$, $r^{\doteq}(p)$ respectively, then

$$r^{\lessdot}(P) = 1 + \sum_{i=1}^{n} r^{\lessdot}(p_i)$$

$$r^{\gtrdot}(P) = 1 + \sum_{i=1}^{n} r^{\gtrdot}(p_i)$$

$$r^{\doteq}(P) = n-1 + \sum_{i=1}^{n} r^{\doteq}(p_i)$$

Proof: Relations which hold when p_i is free standing between leftmost symbols of p_i and the starting dummy get changed to relations between leftmost symbols of p_i and x_{i-1}, the start symbol of p_{i-1}. Relations between rightmost symbols and the rightmost dummy get changed into relations between rightmost symbols and the leftmost terminal of p_{i+1}. The extra relations hold between the x_i and x_{i+1} and also between the left dummy and x_1 and x_n and the right dummy.

4.2.3.16 Remark

A similar theorem but more complicated holds for the number of occurrences between a particular pair of symbols.

4.2.3.17 Corollary

A relation $t(X,Y) = \gtrdot$ is never used unless Y is terminal.

Proof: Examination of form of sequence in last property.

4.2.3.18 Remark

Precedence parsers stand up to errors in the input string better than the other two types because in effect they restart their parsing from scratch every time they use some new input. Hence errors in the input string yield more localised effects in precedence parsing than in the other two. A powerful generalisation would be a backtracking precedence parser.

The difficulty in making precedence parsers backtrack seems to be that whereas in the case of top-down and bottom-up parsing the activity at any instant is centred on a single node, and it is therefore possible to store information associated with a node so that a parser can reorientate itself after backtracing, in the case of precedence parsing the activity is centred on a pair of nodes. Information for backtracking must not only be stored with pairs which do occur together, but also with pairs which might occur together. For instance in diagram 4.2.3.13 X_{n-1} and Y_{m-1} form such a pair. In short, to allow for backtracking a very complicated data structure seems necessary.

4.3 DECIDABILITY AND POST'S PROBLEM

A continuous concern of this thesis has been to calculate various parameters. Ideally the solution is given in closed form (e.g. results in chapter 1); as a second best a method for calculating a solution is given (e.g. in 2.3.3). A third and worst possibility is not to present an algorithm, but at least prove that none exists.

To prove that an algorithm does exist is in essence easy; all that has to be done is to present the algorithm. To prove that one does not exist is more difficult; there are an infinite number of algorithms so it is impossible to present them all one by one as non-algorithms. So algorithms must be excluded in blocks by their properties. Luckily properties of algorithms have already been extensively studied [3, 28], and particular problems shown to have no solving algorithms. This gives rise to a relatively simple method to show that a new problem has no algorithm to solve it. The method is to present a way to convert any algorithm solving the new problem into an algorithm solving an old problem. So if the

new problem had an algorithm solving it then the algorithm could be converted into one solving the old problem. As the old problem has no solution *neither has the new.*

In this section some standard definitions will be given and the old problem described.

4.3.1 Definitions

A property is semi-decidable if there is an algorithm which when given an object to which the property might apply returns the answer 'yes' in a finite time if the property does in fact hold.

A property is decidable if both it and its converse are semi-decidable, equivalently if there is an algorithm which when given an object to which the property might apply returns with either the answer 'yes' or else the answer 'no' in a finite amount of time, depending on whether or not the property holds.

A set is recursively enumerable if it can be defined by a semi-decidable property. Equivalently if there is an algorithm which prints out all its elements.

A set is recursive if its property is decidable.

4.3.2 Proposition (Post's Theorem)

Consider the property which is true of a pair of ordered tuples of words $\langle \alpha_1, \ldots, \alpha_j \rangle$ and $\langle \beta_1, \ldots, \beta_j \rangle$ if and only if there exists a finite sequence of integers $i_1, \ldots, i_n$ (where for all k, $1 \leq i_k \leq j$) such that the corresponding concatenations $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_n}$ and $\beta_{i_1} \beta_{i_2} \ldots \beta_{i_n}$ are the same word. This property is semi-decidable but not decidable.

Proof: That it is semi-decidable is easy to prove, the algorithm is to try all sequences $i_1, \ldots, i_n$ in order and stop with the answer 'yes'

When the corresponding concatenations turn out to be the same.

To prove that the property is not decidable is very difficult. It was originally done by Post, and is more accessible in [28 chs 12 & 13].

4.3.3 Remark

The previous result can be used in context-free grammar theory via the intermediary of a Post grammar.

4.3.4 Definition

Given a Post problem on the words $\langle \alpha_1, \ldots, \alpha_j \rangle$ and $\langle \beta_1, \ldots, \beta_j \rangle$, the corresponding pair of Post grammars G, H have the following forms.

The non-terminal alphabet of each contains a single symbol, A, B respectively.

The terminal alphabet is common to both. It consists of all the symbols used in forming any of the words α_i or β_i $(i=1,\ldots,j)$ in addition j other distinct new symbols n_i $(i=1,\ldots,j)$ and finally a 'middle' symbol a.

The productions are j+1 in number. j are of the form $\langle A \to n_i A \alpha_i \rangle$, or $\langle B \to n_i B \beta_i \rangle$ respectively, the final production is $\langle A \to a \rangle$ or $\langle B \to a \rangle$.

The start symbols of G and H are determined to be A and B because each grammar has that symbol as its sole non-terminal.

4.3.5 Theorem

The pair of ordered tuples $\langle \alpha_1, \ldots, \alpha_j \rangle$, $\langle \beta_1, \ldots, \beta_j \rangle$ has Post's property if and only if the languages of the corresponding grammars G and H have non empty intersection.

Proof: If $\alpha_{i_1} \ldots \alpha_{i_k}$ is the same word as $\beta_{i_1} \ldots \beta_{i_k}$ then $n_{i_k} n_{i_{k-1}} \ldots n_{i_1} a \alpha_{i_1} \ldots \alpha_{i_{k-1}} \alpha_{i_k}$ is the same word as

$n_{i_k} n_{i_{k-1}} \cdots n_{i_1} a \beta_{i_1} \cdots \beta_{i_{k-1}} \beta_{i_k}$ so the intersection of the languages of G and H is non empty.

Conversely, if G and H have non empty intersection then any word in that intersection must have the form $n_{i_k} \ldots n_{i_1} a \alpha_{i_1} \ldots \alpha_{i_k}$ and also $n_{i_k} \ldots n_{i_1} a \beta_{i_1} \ldots \beta_{i_k}$ for some sequence of integers $i_1, \ldots, i_k$. This sequence is that required to demonstrate that Post's property holds.

4.3.6 Remark

There is a single plan for all the undecidability results obtained from Post grammars. First the grammars G, H are modified in some way to produce sometimes a new pair of grammars G', H', sometimes just a single grammar G'. It is then shown that the new (pair of) grammar(s) has some property if and only if G and H have non empty intersection. The new property is then only semi-decidable, for if it were decidable then the intersection problem for G and H would be also decidable.

As a simple example it can be shown that ambiguity is not a decidable property [11 section 4.5].

4.3.7 Theorem

The set of ambiguous grammars is not recursive.

Proof: From any pair of Post grammars $G = \langle \{A\}, T, P, A \rangle$, $H = \langle \{B\}, T, Q, B \rangle$ it is possible to form a new grammar $G' = \langle \{A, B, S\}, T, P \cup Q \quad S \to A \;,\; S \to B \;,\; S \rangle$. Parses of G' have their roots expanded by either the production $\langle S \to A \rangle$ or else $\langle S \to B \rangle$ and the remainder of their productions taken from G or H respectively. Hence G' is ambiguous if and only if G and H have non empty intersection.

4.3.8 Remark

The set of ambiguous grammars is however recursively enumerable. This is best shown by presenting a semi-decision algorithm.

4.4 THE NON-PROBABILISTIC EFFECTIVENESS OF PARSERS

One of the choices a software programmer has to make when writing a compiler is which parsing method to use. Various factors influence his decision. Probably the most important in practice is his state of knowledge, i.e. he chooses the method he knows best. Less accidental criteria are: how well a parsing strategy interfaces with the remainder of a compiler; how robust the parsing method is under the influence of errors in the input; how large a piece of code and how much data structure is needed to implement the method; and in particular the two factors to be dealt with here, how large a subset of the input language is successfully parsed and how quickly (in how many steps) the parser does its job.

There are two methods to obtain values for these factors: the first is by measurement, the second by calculation. Factors can only be measured for a compiler already in existence, so direct measurement is out of the question for predicting the behaviour of a compiler at design stage. Indirect measurement, that is measuring already existing compilers and extrapolating the results to the new compiler, is also difficult. This is because there are at least four inter-dependent factors which effect the measured behaviour of a compiler; they are the parsing strategy, the design of the rest of the compiler, the grammar of the language which is to be parsed, the machine on which the compiler programme is running. As yet these factors have not been completely disentangled and as compilers are very complicated they are unlikely to be so. There remains what is attempted in the next two sections, to obtain the behaviour of parsers by calculation.

In this section information about the probability distribution is ignored. First some results about the absolute size of the set successfully parsable by particular methods is given, then some comparative results; that is, does the subset parsable by one method contain that parsable by another? Next some absolute and relative results are given for the speed of a parser. If one parser takes less moves than another whatever the input string, then it is faster in a very strong sense.

In the next section the same four problems are again attacked, but this time in terms of the absolute and relative measures of sets and the absolute and relative average speeds of parsers. Unfortunately the results are not of much practical use. Firstly they are undecidability results. Secondly, in real life there is certainly more than zero probability that an incorrect input is met. So more realistic results would not only use a different measure from any generated by a pre-probability, but also contain some estimate of the time required to reject an incorrect input.

4.4.1 Remark

The first question to be dealt with concerns the absolute size of the parsable subset. There is only one absolute set-theoretic measure of a subset, is it or is it not the complete set, so the question becomes: give criteria on grammars and parsing methods which distinguish those which succeed on all their inputs from the others. This will be done by examining ways in which parsers can fail.

4.4.2 Definition

A grammar is left recursive if it contains a non-terminal A such that $A \overset{+}{\Rightarrow} A\alpha$ where α is a possibly empty string containing possibly both terminals and non-terminals.

4.4.3 Theorem

There is a finite test for whether or not a grammar is left recursive.

Proof: The relation L which holds between two non-terminals A, B iff there is a production $\langle A \rightarrow B\alpha \rangle$ (where α possibly empty) is a finite relation. Hence its transitive closure L^+ is finite and obtainable with finite effort. It is a finite task to test all non-terminals A to discover whether or not AL^+A. But a grammar is left recursive if and only if there is such an A.

4.4.4 Theorem

For any left recursive grammar G and any top-down LR(k) parser of G there is a word from the language of G for which the parser fails to find a parse.

Proof: Let $A \overset{+}{\Rightarrow} A\alpha$ be a generation which makes G left recursive where without loss of generality α is a terminal string. Let l be such that the length of α^l is greater than k. Let $S \overset{+}{\Rightarrow} \beta A\gamma \overset{+}{\Rightarrow} \beta A\alpha^l\gamma \overset{+}{\Rightarrow} \beta\delta\alpha^l\gamma$ be the skeleton of the generation of the terminal word $\beta\delta\alpha^l\gamma$. Then the parser will fail on this word. If it does not succeed in parsing the initial string β then it fails. If it does and reaches a state where only correct dominoes are on the board, the active edge is marked with an A, and the input is $\delta\alpha^l\gamma$, then it will also fail. For to succeed in parsing it must next abutt some correct dominoes under A until the new active edge is A again. But if the parser succeeds in doing this then the new A is faced with exactly the same context as the original A and so the procedure will loop indefinitely, growing a longer and longer chain of dominoes on the board as it goes. If the new active edge A is not placed on the board then a correct domino has not been placed so

the procedure still fails.

4.4.5 Remark

The above theorem only states that the parser of a left recursive grammar will fail, not that it will fail by looping. There is however a converse.

4.4.6 Theorem

Top-down parsers for non left recursive grammars always terminate in finite time (although not necessarily with a parse).

Proof: Let α be a finite string of length n on which the parser continues indefinitely. Consider the dominoes which may be placed under the start piece. One must have been used last, because there are only a finite number of possibilities, and if all are removed the parsing terminates. Consider the bottom edges of this last domino, one of these must be abutted against last for a similar reason. There is similarly a last option for the domino against this edge, and so continuing this procedure, a well-defined infinite chain of last dominoes and last edges used is obtained. In order for top-down parsing to go from abutting below the leftmost bottom edge of a domino to the next bottom edge at least one input must be matched against, so as there is only a finite number of input symbols, the final limit chain of dominoes must, beyond a certain point, consist only of dominoes abutted to the leftmost bottom edge of the domino above. There is an infinite number of occurrences of non-terminals marking these bottom edges; as there are only finitely many different non-terminals, one must occur twice. But then this non-terminal is left recursive.

4.4.7 Theorem

Any bottom-up parser of any non-ambiguous grammar always terminates on all inputs.

Proof: An indefinitely long ascending chain of dominoes cannot be formed by abutting to target edges marked with some symbol Y dominoes corresponding to productions of the form $\langle X \rightarrow Y\alpha \rangle$ where α is non empty, because before X can be used as the active edge at least one input symbol must be used to ground α . On the other hand, if there are indefinitely long chains containing dominoes corresponding to productions of the form $\langle X \rightarrow Y \rangle$ then the grammar is ambiguous.

4.4.8 Remark

Precedence parsers always terminate but the following result seems similar to theorem 4.4.4.

4.4.9 Theorem

If a grammar G contains some left recursive symbol A which in addition occurs in the second or later position of the right-hand side of some production of G, then the precedence table of G suffers from clashes.

Proof: Let $\langle X \rightarrow \alpha BA\beta \rangle$ be the production, where α and β are possibly empty strings. Then $B \doteq A$ by the precedence table defining rule 1 and $B \lessdot A$ by rule 2 [4.2.3.2].

4.4.10 Remark

The previous theorems allow attention to be concentrated on those parsers which terminate on all inputs. The exposition continues by looking in detail into why terminating parsers succeed and fail.

4.4.11 Lemma

Any terminating parsing method will fail to find a parse if a

correct domino is ever removed from the board.

Proof: The parsing methods have been designed never to repeat themselves. Hence if a correct domino is removed it cannot be replaced. Every correct domino has to be in place for a parsing method to succeed.

4.4.12 Lemma

When a terminating parser stops there are no incorrect dominoes on the table.

Proof: If it succeeds there are only correct ones; if it fails none at all.

4.4.13 Theorem

A terminating top-down parser will find a correct parse for a word if and only if no first incorrect dominoes ground.

Proof: Every incorrect domino is eventually removed including every first incorrect domino. If a first incorrect domino does ground it can only be removed along with some domino above it, but that domino must be correct, and hence by the previous lemma the parser fails.

On the other hand, if a first incorrect domino fails to ground it and all the incorrect dominoes beneath it will eventually be removed leaving the pattern of dominoes on the table the same as just before it was placed. So if there were only correct dominoes on the table before the first incorrect domino was placed there are still the same number of first incorrect dominoes after it is removed. By induction the number of correct dominoes can only increase so when the parser terminates it must succeed.

4.4.14 Definition

A terminating top-down parser is prematurely successful on a particular input if some first incorrect domino grounds on that input;

it is in short just prematurely successful if such an input exists.

4.4.15 Theorem

If a top-down parser is neither left recursive nor prematurely successful, then it succeeds on all words in the language.

Proof: This is theorem 4.4.13 restated using the definition of premature success.

4.4.16 Definitions

Given a grammar G with start symbol S, if $P_1 = \langle X \rightarrow \alpha_1 \rangle$ and $P_2 = \langle X \rightarrow \alpha_2 \rangle$ are two of its productions then P_1 will be said to initialise P_2 if $S \overset{*}{\Rightarrow} \beta_1 X \beta_2$ and $\alpha_2 \beta_2 \overset{*}{\Rightarrow} \gamma$ such that α_1 can generate an initial substring of γ .

X will be said to generate α_1 in top-down context.

4.4.17 Theorem

A terminating top-down parser suffers from premature success if and only if there is a pair of productions P_1, P_2, such that P_1 occurs before P_2 in an entry of the parsing table and P_1 initialises P_2.

Proof: P_1, the first first-incorrect domino to ground can do so only if it generates in top-down context an initial substring of the string generated by P_2, the correct domino at that point. That is, a parser fails by premature success only if at least one pair of productions as in the statement of the theorem exists.

If P_1 initialises P_2 then by definition there must be a string γ generable in top-down context by P_2 such that P_1 can generate an initial substring δ . If P_1 succeeds in grounding when presented with δ then it is prematurely successful when presented with γ ; if it does not then some error has occurred somewhere and by theorem

4.4.15 it must be premature success.

4.4.18 Remark

The relation which holds between productions $P_1 = \langle X \rightarrow \alpha_1 \rangle$ $P_2 = \langle X \rightarrow \alpha_2 \rangle$ when P_1 initialises P_2 is not intrinsically symmetric (although it may be so for some grammars), and premature success can only happen when a false production is chosen first. Thus it can happen that an LR(k) parser suffers from premature success but another with a different ordering of the productions in its table does not.

It can also happen that a long enough look ahead can resolve the choice; an LR(k) parser may suffer from premature success although none of the compatible LR(k+1) parsers do so. On the other hand, an LR(K+1) parser is always at least as good as a compatible LR(k) parser, its parsable set always contains at least as much of the language to be parsed.

4.4.19 Theorem

The set of prematurely successful top-down terminating parsers is recursively enumerable but not recursive.

Proof: To show that it is recursively enumerable is easy, all parsers can be run on all their possible inputs. This can be done as a sequential algorithm by the standard diagonalisation. If a parser is prematurely successful the string on which it is so must eventually be tried on it. If it is not the process never stops.

The non-recursiveness can now be proved with the aid of Post grammars. Every top-down parser for a Post grammar always succeeds on every input because the right-hand side of each production starts with a different terminal. Similarly if a grammar G' contains a Post grammar G with start symbol A as a subgrammar, and a parser for

G' starts trying to parse a subword generated by G with A as the active symbol then it will succeed in grounding A. Consider the grammar

$$G' = \langle \{S,X,A,B\}, T \cup \{c,d,e\}, P \cup Q \cup \{\langle S \rightarrow Xc\rangle, \langle X \rightarrow Ad\rangle, \langle X \rightarrow Bde\rangle\}, S\rangle$$

where $G = \langle \{A\}, T, P, A\rangle$ and $H = \langle \{B\}, T, Q, B\rangle$ are a pair of Post grammars, and S,X,c,d,e are new distinct symbols. This grammar is unambiguous. The parser which chooses the production $\langle X \rightarrow Ad\rangle$ before the production $\langle X \rightarrow Bde\rangle$ succeeds prematurely if and only if a word generated by Ad is an initial subword of a word generated by Bde. Clearly this is the case iff A and B can generate the same word. So if there was a procedure for deciding if an arbitrary grammar is prematurely successful, it would yield a decision for a grammar of the above kind, and hence decide whether or not A and B can generate a common string. That is, it would solve Post's problem.

4.4.20 Corollary

The above theorem also holds for LR(k) parsers however large k.

Proof: A similar argument can be gone through using the grammar G' constructed from two Post grammars and the additional productions $\langle S \rightarrow Xc^k\rangle$, $\langle X \rightarrow Ad\rangle$, $\langle X \rightarrow Bdc^k e\rangle$.

4.4.21 Remark

To sum up, there is no general algorithm to answer even the minimal possible question about the size of the subset parsable by a top-down parser - is it the set of all strings in the language?

The theory for bottom-up parsers is almost the same and so will only be sketched.

4.4.22 Theorem

A bottom-up parser will find a correct parse for a word if and only if no first incorrect domino grounds.

Proof: If a first incorrect domino grounds it can only be removed along with some dominoes placed before it in time, which must therefore be correct. Hence by lemma 4.4.12 the parser fails.

On the other hand if it fails to ground it will eventually be removed with all the dominoes placed after it, leaving only correct dominoes.

4.4.23 Remark

The definitions for premature success and initialising in bottom-up context can now be made in a similar way to those for top-down. But in bottom-up context if P_1 initialises P_2 then P_1 must have the form $\langle X_1 \rightarrow x \alpha_1 \rangle$ and P_2 $\langle X_2 \rightarrow x \alpha_2 \rangle$.

4.4.24 Theorem

The set of prematurely successful bottom-up parsers is recursively enumerable but not recursive.

Proof: The proof is similar to that of the corresponding result for top-down parsing (theorem 4.4.19), and can even use exactly the same grammars.

4.4.25 Remark

Finally to round off the question of what is the subset parsable by a parser, a precedence parser always succeeds on all inputs and there is a finite test for whether or not a grammar is precedence [35].

Although there is no general testing algorithm to decide whether or not all parsing strategies are universally successful, there are tests for restricted classes of grammars. For instance, it is not too difficult to see that there is a finite test for any top-down or bottom-up strategy acting on an LR(k) grammar (where k is any specified

value chosen in advance). This is so because every first incorrect domino is removed after at most k input symbols are looked at.

4.4.26 Remark

The final problem to be dealt with in this section is the speed of the parser. In practice this will depend on all kinds of hardware and software details such as the relative amount of time needed to execute different hardware instructions and the exact way in which a strategy is programmed. To calculate the speed of a parser taking these details into account would be complicated and tedious (although a compiler designer would find the results of such a computation for his language useful), so two simpler proxies for the time taken by a parser will be examined here. First the total number of dominoes placed (at any time) on the board. Second the number of table look-ups needed. These proxies are reasonable because a real compiler will probably have a particular finite piece of code which has the same effect as if a domino were placed on the board and another piece to do the table look-up. Possibly the amount of time required may vary from one domino to another, but this will just complicate the calculations, not rearrange their basic form.

4.4.27 Theorem

For any string α generated from a grammar G, all successful deterministic parsers of G place the same number of dominoes on the board when parsing α. Any parser which backtracks on α places more dominoes.

Proof: The number of dominoes placed by a deterministic parser is the same as the number on the board when it is finished. This number is given by the structure of the parse and so independent of the

parsing method.

A backtracking parser ends up with the same number of dominoes but has removed some during its parsing. So in total it must have placed more.

4.4.28 Remark

Despite the above result there are a few occasions when backtracking parsers are required in practice. The following two problems are concerned with calculating their relative speeds.

4.4.29 Theorem

If P_1 is a successful LR(k) parser for a grammar and P_2 is the compatible LR(k+1) parser, then on no input does P_2 place more dominoes than P_1.

Proof: The order in which P_1 and P_2 lay down and pick up dominoes is the same, except that P_1 may occasionally abutt an incorrect domino and its dependents against a node and then pick them up again before trying the next alternative, whereas P_2 lays the second (or later) alternative as its first attempt.

4.4.30 Remark

In general if two parsers have different orders of their parses in their parsing tables, then one will work faster on some inputs and the second on others. Although the next and final theorem of this section answers a question which is probably unlikely to be asked in practice, it does show that there are undecidability results connected with the speed of parsers, and is also used to prove a consequence in the next section.

4.4.31 Theorem

Given two grammars and a bijective correspondence between their languages, there is no finite algorithm which decides that two parsers

for the languages take exactly the same number of moves on corresponding inputs.

Proof: Consider two grammars each constructed from the same pair of Post grammars with start symbol A_1 and A_2. Both grammars have the start symbol S and other additional non-terminals C_1, C_2, D_1, D_2. The first grammar has additional terminals c_1, c_2, d_1, d_2, and additional productions $\langle S \rightarrow C_1 D_1 \rangle$, $\langle S \rightarrow C_2 D_2 \rangle$, $\langle C_1 \rightarrow c_1 \rangle$, $\langle C_2 \rightarrow c_2 \rangle$, $\langle D_1 \rightarrow d_1 \rangle$, $\langle D_2 \rightarrow d_2 \rangle$. The second grammar is the same except that it has the single terminal c instead of the two c_1 and c_2.

It is not too difficult to see that a parser for the first grammar will make exactly the same moves as the corresponding parser for the second except on words whose first part is in the intersection of the languages generated by A_1 and A_2. On these exceptions the first parser needs to read one less input letter than the second in order to resolve the ambiguity, and so if it backtracks it does so after placing one less domino.

Thus if there were a finite algorithm as stated in the theorem, Post's problem could be solved.

4.4.32 Remark

The above theorem does not state whether top-down or bottom-up parsers are intended, it works equally well for both. The grammar can be modified to show that there is no algorithm to decide that an LR(k+1) parser is strictly faster than a compatible LR(k) parser, the technical trick is to pad out most terminal symbols with k+1 preceding copies of a new dummy symbol to prevent the look ahead getting any valuable information except when dealing with the final ambiguity resolving c's and d's.

A final conjecture is that there is a way of defining compatibility

between top-down and bottom-up parsers, and that a bottom-up parser is never slower than its corresponding compatible top-down parser.

4.5 THE PROBABILISTIC EFFECTIVENESS OF PARSERS

In this section it is assumed that a parser is dealing with an input randomly selected from its language with probability distribution generated by a preprobability function. Two problems are tackled. What is the measure of the parsable subset? What is the average number of moves needed to carry out the parsing? First the questions are answered for the successful deterministic parsers, then they are attempted for backtracking parsers, but it is shown that here many versions of the questions have no solutions.

4.5.1 Theorem

The measure of the parsable set of a successful deterministic parser is one.

Proof: All inputs are parsable.

4.5.2 Theorem

The average number of dominoes placed by a successful deterministic parser can be calculated.

Proof: By theorem 4.4.27 this is the same as the average number of non-terminals in a parse which can be calculated in a similar way to that given in theorem 1.5.6.

4.5.3 Theorem

The average number of table look-ups required by deterministic successful bottom-up or top-down parsers can be calculated and in addition the average number of timeseach table entry is used.

Proof: Each domino requires exactly one table access and the probability that it is found in response to a particular input k-tuple can be found in the same way as in theorem 1.5.19.

4.5.4 Theorem

The average number of table look-ups required for precedence parsing can be calculated, and also the average number of times each table entry is used.

Proof: Theorem 4.2.3.12 shows that if a parse P has X at its root, first production $\langle X \to X_1 \ldots X_k \rangle$, and the subparses $P_1,\ldots,P_k$ hanging below $X_1,\ldots,X_k$ respectively, then the precedence relations which appear when parsing P are all those which appear when parsing the P_i plus the additional one relation $\lessdot$ between X_1 and the initial dummy, one relation $\gtrdot$ between X_k and the final dummy, and k-1 relations $\doteq$ between adjacent symbols X_i, X_{i+1}; except that relations between the initial dummy and any leftmost node of P_i are changed to the same relation between X_{i-1} and that leftmost node (i=2,...,k), and relations between any rightmost node of P_i and the final dummy are changed to the same relation between that rightmost node and the leftmost terminal of P_{i+1} (i=1,...,k-1).

Hence a recursive set of linear equations for the averages can be constructed and solved in the usual way.

4.5.5 Remark

The deterministic questions are answered; next the backtracking parsers will be dealt with.

4.5.6 Definition

If A and B are sets of numerical expressions then A is B-comparable if, given any expressions $x \in A$ and $y \in B$, it is always the case that

one of the three relations $x<y$, $x=y$, $x>y$ can be demonstrated to hold with finite effort. (As usual, singleton sets will be written x rather than $\{x\}$, and in particular a set will be 1-comparable rather than $\{'1'\}$-comparable.)

4.5.7 Theorem

There is no finite algorithm which always yields a 1-comparable expression for the measure of the set of words on which a top-down or bottom-up parser succeeds.

Proof: If there were such an algorithm then it could be run to yield in finite time an expression E for the measure of the parsable set, and then by the definition of 1-comparability a further finite calculation would show whether E=1 or not. As every input word has non-zero measure, if E=1 then the parser is universally successful; otherwise it fails on some input. So this is a finite algorithm to solve the premature success problem. As no such algorithm exists (theorems 4.4.21 top-down, and 4.4.24 bottom-up), the theorem follows.

4.5.8 Corollary

There is no general way to obtain a linear equation of the form

$$Ax = b$$

which is satisfied by the probability of success, where the elements of A and b are rational functions of the preprobabilities.

Proof: Solutions of such equations are 1-comparable.

4.5.9 Corollary

There is no general way to obtain an algebraic equation of the form

$$f(x) = 0$$

which is satisfied by the probability of success, where the coefficients of f are rational functions of the preprobabilities.

Proof: Solutions of such equations are 1-comparable.

4.5.10 Remark

Because the above theorem only assumes that the measure of each word is non-zero, it holds for a far larger class of measures than those generated by preprobabilities. It is reasonable to assume that if a word is in a language it should occur occasionally, and the theorem holds of all these reasonable measures.

The theorem does not absolutely exclude the possibility of finding a closed expression for the probability of success (it does not seem unlikely that some of the classes of expressions in everyday mathematical use are not 1-comparable, for instance expressions involving complicated integrals seem likely candidates).

Finally it is easy to approximate as closely as desired to the success probability, given any small ϵ there is always a finite set of words whose measure is greater than $1-\epsilon$.

4.5.11 Remark

It is just conceivable that despite the previous theorem, the average of the number of moves made by any parser (or perhaps just non prematurely successful parser) could be calculated. The next theorem shows that here too it is impossible to get a reasonable result.

4.5.12 Theorem

If E is the set of all expressions which can be the result of an algorithm which calculates the average number of moves made by any (non prematurely successful) parser, then E is not E-comparable.

Proof: This depends on theorem 4.4.31 but is otherwise proved in the same way as theorem 4.5.7 above. The pairs of grammars used in 4.4.31 have parsers which succeed on all inputs.

4.5.13 Corollary

There is no general way to obtain either a linear or an algebraic equation which is satisfied by the average number of moves required by a parser, where the coefficients of the equation are rational functions of the preprobabilities.

Proof: If F is the set of such solutions, F is F-comparable.

4.5.14 Remark

Although there is no algorithm which calculates the probability of success, or the average number of moves for all parsers, there are limited algorithms which work for some parsers. The next theorem gives an example.

4.5.15 Theorem

It is possible to calculate the success probability and average speed of an $LR(k_1)$ parser on an $LR(k_2)$ grammar.

Proof: If $k_1 \geq k_2$ then the parser is deterministic (and successful). If not, every first incorrect domino is removed before at most k_1+1 symbols of the input have been scanned. It is possible to calculate the probability of every type of node (domino) being generated, and also the relative probabilities of every initial k_2-tuple of symbols generable by a node. It is a finite (but very large) amount of work to calculate for each domino and each k_2-tuple of symbols which can be generated by another domino which shares an entry with the first domino in the parsing table, how many moves are required when the first domino is placed incorrectly, and whether the backtracking works

correctly. From this information the measures and averages may be obtained.

APPENDIX

This is to prove the theorem given in section 2.1.5. The proof is due to Dr. G. Segal.

A.1 Notation

$\mathbb{C}$ is the set of all complex numbers, $\mathbb{C}[x_1,\ldots,x_n]$ is the set of all multinomials with complex coefficients and variables $x_1,\ldots,x_n$.

A.2 Definition

An algebraic variety is a subset $V \subseteq \mathbb{C}^n$ such that there is a set $\{F_\alpha\}_{\alpha \in S}$ of multinomials in $\mathbb{C}[x_1,\ldots,x_n]$ such that

$V = \{(z_1,\ldots,z_n) : F_\alpha(z_1,\ldots,z_n) = 0 \text{ for all } \alpha \in S\}$.

A.3 Note

By Hilbert's basis theorem, S may be assumed finite.

A.4 Definition

An algebraic variety V is irreducible if whenever $F, G \in \mathbb{C}[x_1,\ldots,x_n]$ are such that $F(z_1,\ldots,z_n)G(z_1,\ldots,z_n) \equiv 0$ for all $z = (z_1,\ldots,z_n)$, then either $F(z) \equiv 0$ or $G(z) \equiv 0$.

A.5 Proposition

An algebraic variety V can be decomposed uniquely as a finite union, $V = V_1 \cup \ldots \cup V_k$, where V_i is irreducible for all i.

A.6 Remark

An irreducible algebraic variety has a dimension which can be defined in various non-trivially equivalent ways.

A.7 Proposition

If V is an algebraic variety in $\mathbb{C}^n$ defined by k multinomials

$F_1,\ldots,F_k \in \mathbb{C}[x_1,\ldots,x_n]$ and if $z \in V$ is such that the jacobean $(\partial F_i/\partial x_j)$ evaluated at z has rank k then

(1) z is an element of exactly one of the irreducible components of V, say V_1;

(2) dimension $(V_1) = n-k$.

A.8 Proposition (Elimination Theorem)

If V is an irreducible algebraic variety in $\mathbb{C}^n$ and V' is its projection onto $\mathbb{C}^k$ (i.e. $V' = \{(z_1,\ldots,z_k): \exists (z_{k+1},\ldots,z_n) \in \mathbb{C}^{n-k}$ s.t. $(z_1,\ldots,z_n) \in V\}$) then V' is an irreducible algebraic variety, and dimension (V') $\leq$ dimension (V).

A.9 Proposition

A one-dimensional variety in $\mathbb{C}^2$ consists of the zeroes of a single polynomial.

A.10 Remark

The next theorem is reason for this appendix.

A.11 Theorem

If $A_i:\mathbb{C}\to\mathbb{C}$ $\quad (i=1,\ldots,n)$

and there are n multinomials $\mathcal{P}_j$ $\mathbb{C}[x,w_1,\ldots,w_n]$ such that

(1) $\mathcal{P}_j(z,A_1(z),\ldots,A_n(z)) \equiv 0 \quad (j=1,\ldots,n)$

(2) The jacobean $(\frac{\partial \mathcal{P}_j}{\partial w_i}) \neq 0$ at at least one point

$$\langle z_o,A_1(z_o),\ldots,A_n(z_o)\rangle$$

then each function A_i is algebraic.

Proof: Let V be the algebraic variety defined by $\mathcal{P}_1,\ldots,\mathcal{P}_n$. By proposition A.7 $\langle z_o,A_1(z_o),\ldots,A_n(z_o)\rangle$ is in just one irreducible component V_1 of V, and dimension $(V_1) = 1$. Let $V_{1,i}$

be the projection of V_1 onto the $\mathbb{C}^2$ space corresponding to the coordinates z and w_i.

Now $\langle z, A_1(z), \ldots, A_n(z) \rangle$ must be in V_1 for all z close enough to z_o because irreducible components are closed subsets of $\mathbb{C}^{n+1}$.

Hence $\langle z, A_i(z) \rangle \in V_{1,i}$ for all z near z_o. Hence $V_{1,i}$ contains ∞ points so is not 0-dimensional. So by proposition A.8 dimension $(V_1, i) = 1$. So by A.9 there is a polynomial $F \in C[x, w_i]$ such that $V_{1,i} = \{(z,z') : F(z,z') = 0\}$. But then $F(z, A_i(z)) \equiv 0$, or in other words A_i is algebraic.

REFERENCES

Note

There are three main continuing sources for papers related to this thesis. JACM (Journal of the Association for Computing Machinery) contains mathematical and algebraic papers about the theory of formal languages, their grammars and their parsers. CACM (Communications of the Association for Computing Machinery) contains more practical descriptions of languages in terms of how they have been (might be) implemented on a computer. Inf. Contr. (Information and Control) contains papers more concerned with information theory.

(1) AHLFORS, L.V. Complex Analysis. McGraw-Hill, New York (1966).

(2) CHOMSKY, N. & SCHÜTZENBERGER, M.P. The Algebraic Theory of Context-Free Languages. In: Computer Programming and Formal Systems, pp.118-161. Braffort, P. & Hirschberg, D. (eds). North-Holland, Amsterdam (1967).

(3) DAVIS, M. Computability and Unsolvability. McGraw-Hill, New York (1958).

(4) ELLIS, C.A. Probabilistic Tree Automata. Inf.Contr. 19 (1971), 401-416.

(5) -------- The Halting Problem for Probabilistic Context-Free Generators. JACM 19, 3 (July 1972), 396-399.

(6) FELDMAN, J. & GRIES, D. Translator Writing Systems. CACM 11, 2 (Feb.1968), 77-113.

(7) FELLER, W. An Introduction to Probability Theory and Its Applications, vol.I. Wiley, New York (1968).

(8) FLOYD, R.W. Syntactic Analysis and Operator Precedence. JACM 10, 3 (July 1963), 316-333.

(9) FULTON, W. Algebraic Curves. Benjamin, New York (1969).

(10) GANTMACHER, F.R. Applications of the Theory of Matrices. Interscience, New York (1959).

(11) GINSBURG, S. The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York (1966).

(12) GREIBACH, S.A. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. JACM 12, 1 (Jan.1965), 42-52.

(13) HARRIS, T.E. The Theory of Branching Processes. Springer-Verlag, Berlin (1963).

(14) HILLE, E. Analytic Function Theory, vol.I. Ginn & Co., Boston (1959).

(15) HOPCROFT, J.E. & ULLMAN, J.D. Formal Languages and Their Relation to Automata. Addison-Wesley, Reading, Massachusetts (1969).

(16) HUTCHINS, S.E. Stochastic Sources for Context-Free Languages. Publication of Dept. Appl. Physics & Information Sci., Univ. Calif. (1968/69).

(17) IBRAGIMOV, I.A. & LINNIK, Yu. V. Independent and Stationary Sequences of Random Variables. Wolters-Noordhoff, Groningen (1971).

(18) INGERMAN, P.Z. A Syntax-Oriented Translator. Academic Press (1967/68).

(19) KAMINGER, F.P. The Non-Computability of the Channel Capacity of Context Sensitive Languages. Inf. Contr. 17 (1970), 175-182.

(20) KHINCHIN, A.I. Mathematical Foundations of Information Theory. Dover, New York (1957).

(21) KNUTH, D.E. On the Translation of Languages from Left to Right. Inf.Contr. 8 (Oct.1965), 607-639.

(22) KUICH, W. On the Entropy of Context-Free Languages. Inf.Contr. 16, 2 (April 1970), 173-200.

(23) ------ The Structure Generating Function and Entropy of Tuple Languages. Inf.Contr. 19 (1971), 195-203.

(24) ------ The Complexity of Skewlinear Tuple Languages and O-regular Languages. Inf.Contr. 19 (1971), 353-367.

(25) LOÈVE, M. Probability Theory. Van Nostrand, Princeton, New Jersey (1963).

(26) McAFEE, J. & PRESSER, L. An Algorithm for the Design of Simple Precedence Grammars. JACM 19, 3 (July 1972), 385-395.

(27) McMILLAN, B. The Basic Theorems of Information Theory. Annals of Maths & Statistics 24 (1953), 196-219.

(28) MINSKY , M.L. Computation: Finite and Infinite Machines. Prentice-Hall, New Jersey (1967).

(29) PARRY, W. Entropy and Generators in Ergodic Theory. Benjamin, New York (1969).

(30) RUDIN, W. Principles of Mathematical Analysis. McGraw-Hill, New York (1953).

(31) SANTOS, E.S. Probabilistic Grammars and Automata. Inf.Contr. 21 (1972), 27-47.

(32) SHAMIR, E. Algebraic, Rational, and Context-Free Power Series in Non-commuting Variables. In: Algebraic Theory of Machines, Languages and Semigroups, pp.329-341. Arbib, M.A. (ed.) Academic Press, New York (1968).

(33) SHANNON, C.E. & WEAVER, W. The Mathematical Theory of Communication. Univ. of Illinois Press, Urbana (1972).

(34) WALK, K. Entropy and Testability of Context-Free Languages. In: Formal Language Description Languages for Computer Programming, pp.105-123. Steel, T.B. (ed.) North-Holland, Amsterdam (1966).

(35) WIRTH, N. & WEBER, H. EULER - a Generalization of ALGOL, and its Formal Definition, Part I, Part II. CACM 9, 1 & 2 (Jan. Feb. 1966), 13-25, 89-99.