CONSTITUENT-LIKELIHOOD GRAMMAR

Eric Steven Atwell
University of Lancaster, England

A *INTRODUCTION*

The paper by Leech *et al.* describes the aims of the LOB Corpus Grammatical Tagging project, and explains the suite of programs we are using to achieve these aims. In this paper, I would like to look in greater detail at the theoretical basis of these programs; I shall attempt to explain exactly what *constituent-likelihood grammar* involves, and suggest some other applications of this probabilistic approach to natural language syntax analysis.

A.1 General principles of CL grammar

The CL grammar used in the LOB Corpus project is specifically designed to be used in tagging, that is, in assigning a grammatical-class marker to each word in a text. In fact, the basic principles could be generalized to apply to other levels of linguistic analysis (parsing, semantic analysis, etc.); in general, if the analysis involves assigning 'labels' to 'constituents', then a CL grammar could be devised for this analysis.

The CL method of grammatical analysis involves two steps:

(i) Each 'constituent' is first assigned a set of *potential* 'labels'. This can be done by some quite simple mechanism such as dictionary-lookup; this may well mean that some of the possible labels are in fact inappropriate in the given context, but this does not matter, since they will be eliminated during the second stage.

(ii) Each of the potential labels of a constituent is then assigned a *relative likelihood* figure, using a formula which takes into account contextual and other relevant factors; having done this, we can then choose the single 'best' label for the constituent, and disregard all the others (no matter how many others there happen to be).

Thus a CL grammar should not be viewed as a set of rules for generating sentences; rather, it is characterized by:

(i) an algorithm for assigning a set of possible 'labels' or tags to any given constituent; and

(ii) a general *relative likelihood formula* which can be used to calculate the relative likelihood of any given label or tag in any given context.

A.2 The LOB CL grammar

In the CL grammar used to analyse the LOB Corpus, the 'labels' are grammatical tags, and the 'constituents' are words (in this special case, all the 'constituents' are at the same 'level'; but this does not mean that CL grammar could not be generalized to deal with more complex structuring).

The tag-assignment algorithm is embodied in the program WORDTAG. Tags are assigned mainly by dictionary-lookup; but since the set of possible words in the English language is open-ended, the algorithm also includes a number of 'default' routines to deal with words which 'fall through the net' (as explained in Leech *et al.*). This means that the tag-assignment algorithm can be used to assign a set of potential tags to *any* word, and this set will almost always include the 'intuitively correct' tag.

Probably the most innovative part of the LOB CL grammar is the general *relative likelihood formula* used by the 'tag-disambiguation' program CHAINPROBS. When a word has been assigned more than one potential tag, this formula is used to find the relative likelihood of each candidate. We have found that a very simple formula, taking into account only the immediate context, will correctly choose the 'best' tag in c. 96-97% of all cases (moreover, this high success rate is consistent regardless of style: novels, newspapers, magazines, etc. all have approximately the same success rate). Section B explains the Tag Relative Likelihood formula in greater detail.

A.3 Other applications of CL grammar

The CL-grammar approach to language analysis was developed specifically for the LOB Corpus Grammatical Tagging research project. However, it has become clear that this method of analysis has many other possible areas of application. The two main advantages of CL grammar

over other methods of natural language analysis are:

(i) *Generality and robustness*: 'Rule-based' analysis algorithms tend to work only with sentences that 'follow the rules', and will fail if presented with 'non-standard' English, accidental misspellings, or other 'deviant' input. Unfortunately, as become clear when researching with a large corpus, 'real-life' English texts are often dotted with many of these 'imperfections'! In contrast, the LOB Corpus Tagging programs are extremely general and 'robust', since they will produce a reasonably acceptable analysis of *any* input (they have successfully dealt with newspaper 'telegraphese', 'foreigner English', Sci-Fi neologisms, and even a 'humorous' text peppered with *deliberate* misspellings!).

(ii) *Simplicity*: Most syntax-analysis programs build a complex 'parse tree' for each sentence, which requires much complicated and time-consuming computation. CL grammar, on the other hand, involves analysis at a 'local level' only; the tag-likelihood function looks at the immediate context only, *not* at a whole sentence; and even within this localized context, the computation is very straightforward. This means that the amount of computation is much less; the analysis is much simpler and faster.

These advantages make CL grammar particularly suitable for applications requiring a simple and fast analysis of a wide range of possible linguistic input. In sections C, D, and E I shall look briefly at three potential uses of CL grammar; a spelling and grammar 'checker' for use in Word Processors, a speech analysis program for converting from spoken to written English, and a general Grammatical Parser for the LOB Corpus.

B THE LOB TAG RELATIVE LIKELIHOOD FUNCTION: HOW WE DEVELOPED THE FORMULA

To give the reader a clearer idea of how 'likelihoods' are calculated in a CL grammar, I will attempt in this section to explain the Tag Relative Likelihood Function used in tagging the LOB Corpus; I will do this by explaining step by step how we developed the mathematical formula.

B.1 A formalism for words and tags

The programs before CHAINPROBS (where the likelihood formula is applied) divide the texts of the LOB Corpus into *records*, where each record contains a single word and a set of potential tags, and each record has a unique reference number (in fact, each record is a separate line of text; but I prefer the term 'record' (rather than 'line'), since this avoids confusion (different words which were on the same line in the original Corpus are in different records)). If we denote the record-number by r, the word by W<r>, and the set of tags by T<r,1>, T<r,2>, T<r,3>, ... T<r,n(r)>, where n(r) is the number of potential tags in the record r, then a typical sequence of records from the LOB Corpus is:

record-no.	word	tags
.		
.		
.		
r-1	W<r-1>	T<r-1,1>, T<r-1,2>, ... T<r-1,n(r-1)>
r	W<r>	T<r,1>, T<r,2>, ... T<r,n(r)>
r+1	W<r+1>	T<r+1,1>, T<r+1,2>, ... T<r+1,n(r+1)>
.		
.		
.		

B.2 Relative and absolute likelihood

CHAINPROBS assigns a percentage likelihood figure to each tag in a record. This percentage is the *relative likelihood* of the tag, relative to all the other potential tags in the record. The *relative likelihood* l of a tag T<r,a> is the *absolute likelihood* L of that tag, divided by the sum of the absolute likelihoods of all the potential tags in the record r:

$$l(T\langle r,a\rangle) = \frac{L(T\langle r,a\rangle)}{\sum_{i=1..n(r)} L(T\langle r,i\rangle)}$$

B.3 'Factorizing' likelihood: L = Lb * Lf * Lw

The absolute likelihood function must now be defined. Ideally, we would like this function to take into account *all* relevant contextual information; this would be the 'perfect' absolute tag-likelihood function. Unfortunately, it is not immediately apparent exactly what such a formula should look like. However, we *can* work towards this 'perfect' formula, step by step: first we must write some simple formula which approximates to the 'ideal'; then, we can add on extra factors to take into account more peripheral information.

To begin with, we can say that the absolute likelihood of a tag is dependent on the '*backward context*' (i.e. the preceding tags) and the '*forward context*' (i.e. the following tags); this allows us to separate out '*backward likelihood*' Lb and '*forward likelihood*' Lf. Another important factor in deciding the likelihood of a tag is of course the *word* it is to be assigned to: for example, with the word "water", the tag NN (noun) is likelier than the tag VB (verb). Thus the absolute likelihood formula must also take into account Lw, the '*word-tag likelihood*'.

The simplest formula for absolute likelihood which takes these three factors into account is:

L = Lb * Lf * Lw

(where * represents multiplication). This is our first approximation to a 'perfect' likelihood function.

B.4 Tag-pair bond B

To calculate the likelihood of a tag T<r,a>, let us assume to begin with that the records immediately before and after r each have only one unambiguous tag. Furthermore, let us assume that the only thing relevant in the 'backwards context' is the single tag in the previous record, T<r-1,1>; and likewise that the only relevant factor in the 'forward likelihood' is the tag T<r+1,1>.

This means that the 'backward likelihood' can be defined as simply the '*bond*' between T<r,a> and the preceding tag T<r-1,1>; and likewise, that Lf is simply the '*bond*' between T<r,a> and T<r+1,1>:

Lb(T<r,a>) = B(T<r-1,1>, T<r,a>)

Lf(T<r,a>) - B(T<r,a>, T<r+1,1>)

Values of the *tag-pair bond* function B are stored in a table with a row and column for every tag in the LOB tagset.

The 'bond' between a pair of tags T1, T2 is dependent on the frequency of cooccurrence, f(T1,T2), compared to the frequency of occurrence of each tag individually, f(T1) and f(T2). These statistics must be extracted from texts which have already been tagged unambiguously (in the LOB Corpus Grammatical Tagging project, we extracted these figures from the Brown Corpus initially (making adjustments where the tagsets differ), but later statistics include figures drawn from the first sections of the LOB Corpus to be analysed).

B.5 Calculating values of B for each tag-pair (T1,T2)

If tags were combined randomly (i.e. if context had no influence on the choice of tag with a word), then the ('random') probability of tag T1 being followed by tag T2 would be

$$P_{\langle random \rangle}(T1,T2) = \frac{f(T1) * f(T2)}{N1}$$

(N1 is a constant, dependent on the number of tags in the sample.)

The actual ('true') probability of the tag-pair (T1,T2) is

$$P_{\langle true \rangle}(T1,T2) = \frac{f(T1,T2)}{N2}$$

(N2 is another constant.)

If we divide P<true> by P<random>, we get a very simple measure of the 'correlation' or *bond* between T1 and T2; ignoring the constant factor (N1/N2) we get the formula:

$$B(T1,T2) = \frac{f(T1,T2)}{f(T1) * f(T2)}$$

The value of B(T1,T2) for any tag-pair (T1,T2) is thus dependent on the sample from which the frequency statistics are derived, so clearly it is important that the sample is representative, and reasonably large. However, even with a very large sample, we cannot be certain

that the figures are perfect, especially if a particular frequency figure happens to be very low or zero; for example, if for a given sample f(DT,DOD)=0, does this mean that the tag-pair (DT,DOD) can *never* cooccur in English, or is this simply a failing of this particular sample? It is safer to assume the latter; so we must add a constant k1 to *all* tag-pair frequency figures, to ensure that all values are greater than zero. Similarly, we should add a constant k2 to all single-tag frequency figures, to ensure that we can never divide by zero. Thus, the new definition of B is

$$B(T1,T2) = \frac{f(T1,T2) + k1}{(f(T1)+k2) * (f(T2)+k2)}$$

B.6 Word-tag likelihood Lw

'Word-tag likelihood' is the likelihood that a given word will have a given tag, regardless of other factors. Dictionary-lookup (or equivalent mechanisms) can give us a very crude measure of Lw: if the tag occurs with the word in the dictionary, then Lw is 1, otherwise 0 (e.g. Lw("the",ATI)=1, but Lw("the",VB)=0).

In the LOB Corpus CL grammar, we found that this 'binary' likelihood function was too crude and simplistic, so we included *four* 'levels' of word-tag likelihood. The 'binary' values of Lw, 0 and 1, are *implicitly* assigned by straightforward dictionary-lookup, as explained above; in addition, the Wordlist used in the LOB Corpus CL grammar has two *explicit* Lw 'weighting markers' (@ and %): if a tag appears with a word only rarely, then that tag is marked @, and if the tag is *very* rare with a given word, it is marked %, for example:

alert JJ VB NN@
water NN VB%
major JJ NN@ VB%

(Notionally @ means that the tag appears with the given word in 10% or less of all uses, and % means 1% or less. In fact often the assignment of weightings was based on 'intelligent guesses', particularly with rare words; this is one reason why we decided to limit ourselves to only four 'grades' of word-tag likelihood (this decision has since been vindicated by the consistently high success rate of the tagging programs: it is clear that a much more 'refined' system of gradations of Lw is unlikely to improve tagging results very

significantly.))

These weighting-markers appear in the LOB WORDLIST, SUFFIXLIST, and IDIOMLIST, and are assigned by WORDTAG (and IDIOMTAG). In fact, within the theoretical framework of a CL grammar, the assignment of these weightings is *not* a necessary part of the tag-assignment algorithm; more correctly, it 'belongs' with the mechanism for calculating tag likelihoods. In other words, if the two tasks of
(i) assigning potential tags to each word, and
(ii) calculating likelihoods for each potential tag
were autonomously dealt with by WORDTAG and CHAINPROBS respectively, then the @ and % 'weighting-markers' would *not* be assigned by WORDTAG; instead, every time CHAINPROBS applied the tag-likelihood function to a tag, it would have to find the appropriate value of Lw for that word-tag combination. Of course, this would require exactly the same word-tag lookup algorithm as was used by WORDTAG to assign the potential tag in the first place; so, to save time, WORDTAG assigns potential tags *and* Lw weighting-markers (where appropriate) in a single search.

B.7 Generalizing the formula to deal with ambiguous contexts

The formulae for Lb and Lf given in B.4 assume that the records immediately before and after the current record are unambiguously tagged, so that in working out the likelihood of tag T<r,a> the only tags we need take into account are T<r-1,1> and T<r+1,1>. However, if either of these records are in fact *ambiguous*, we must take the other tags into account also. For example, if the immediately *preceding* record is ambiguously tagged, then the formula for *backward likelihood* Lb must take into account not only T<r-1,1>, but also all the other potential tags in record r-1: T<r-1,2>, T<r-1,3>, ... T<r-1,n(r-1)>.

For each potential preceding tag T<r-1,i>, we must take into account the *bond* between T<r-1,i> and T<r,a>, 'weighted' by the Backward Likelihood in turn of T<r-1,i>, and also the Word-Tag Likelihood Lw of T<r-1,i>. Thus, *backward likelihood* must be redefined as a recursive function:

$$Lb(T\langle r,a\rangle) = \sum_{i=1..n(r-1)} \begin{pmatrix} B(T\langle r-1,i\rangle, T\langle r,a\rangle) \\ *Lw(W\langle r-1\rangle, T\langle r-1,i\rangle) \\ *Lb(T\langle r-1,i\rangle) \end{pmatrix}$$

Forward likelihood must also be redefined, so it can deal with sequences of tag-ambiguities:

$$Lf(T\langle r,a\rangle) = \sum_{i=1..n(r+1)} \begin{pmatrix} B(T\langle r,a\rangle, T\langle r+1,j\rangle) \\ *Lw(W\langle r+1\rangle, T\langle r+1,j\rangle) \\ *Lf(T\langle r+1,j\rangle) \end{pmatrix}$$

Notice that the recursive definition of Lb means that the *backward likelihood* of a tag T<r,a> theoretically takes into account *all* tags preceding T<r,a>; however, in calculating *relative* likelihood, the set of possible 'backward contexts' before the last *unambiguous* tag is the same for all the potential tags in record r, so this can be "cancelled out". Similarly, *forward likelihood* recursively defined should theoretically involve *all* tags after T<r,a>; but in calculating *relative* likelihood all bonds after the next unambiguous tag "cancel out" and can thus be ignored.

In other words, when calculating the relative likelihood of any tag using the general formulae for Lb and Lf, we need only 'look back' as far as the *last unambiguous tag*, and we need only 'look forward' as far as the *next unambiguous tag*. In general, tags are 'disambiguated' by looking *only* at the words in the *immediate context*.

B.8 The relative likelihood function

As an example, let us take a sequence of five records, with five consecutive words: A, B, C, D, E; and with tags: a, b, b', c, c', d, d', e (the first and last records are unambiguously tagged, while the intermediate records have two tags each):

record no.	word	tags
r1	A	a
r2	B	b, b'
r3	C	c, c'
r4	D	d, d'
r5	E	e

To show how the formulae are applied, let us calculate l(d), the relative likelihood of the tag d. The formula from B.2 tells us

$$l(d) = \frac{L(d)}{L(d) + L(d')}$$

L(d) and L(d') can be expanded using the formula from B.3:

$$L(d) = Lb(d) * Lf(d) * Lw(D,d)$$
$$L(d') = Lb(d') * Lf(d') * Lw(D,d')$$

Applying the recursive formulae for Lb and Lf from B.7, these equations expand to:

$$L(d) = Lb(a) * Lw(A,a) * Lf(e) * Lw(E,e) * \left(\begin{array}{l} B(a,b)*Lw(B,b)*B(b,c)*Lw(C,c)*B(c,d)*Lw(D,d)*B(d,e) \\ +B(a,b')*Lw(B,b')*B(b',c)*Lw(C,c)*B(c,d)*Lw(D,d)*B(d,e) \\ +B(a,b)*Lw(B,b)*B(b,c')*Lw(C,c')*B(c',d)*Lw(D,d)*B(d,e) \\ +B(a,b')*Lw(B,b')*B(b',c')*Lw(C,c')*B(c',d)*Lw(D,d)*B(d,e) \end{array} \right)$$

$$L(d') = Lb(a) * Lw(A,a) * Lf(e) * Lw(E,e) * \left(\begin{array}{l} B(a,b)*Lw(B,b)*B(b,c)*Lw(C,c)*B(c,d')*Lw(D,d')*B(d',e) \\ +B(a,b')*Lw(B,b')*B(b',c)*Lw(C,c)*B(c,d')*Lw(D,d')*B(d',e) \\ +B(a,b)*Lw(B,b)*B(b,c')*Lw(C,c')*B(c',d')*Lw(D,d')*B(d',e) \\ +B(a,b')*Lw(B,b')*B(b',c')*Lw(C,c')*B(c',d')*Lw(D,d')*B(d',e) \end{array} \right)$$

We can think of a term such as

B(a,b)*Lw(B,b)*B(b,c)*Lw(C,c)*B(c,d)*Lw(D,d)*B(d,e)

as a *chain*, represented by [abcde]. This notational simplification allows us to rewrite the equation for the relative likelihood thus:

$$l(d) = \frac{L(d)}{L(d) + L(d')}$$

$$= \frac{[abcde] + [ab'cde] + [abc'de] + [ab'c'de]}{[abcde] + [ab'cde] + [abc'de] + [ab'c'de] + [abcd'e] + [ab'cd'e] + [abc'd'e] + [ab'c'd'e]}$$

$$= \frac{\text{(SUM OF ALL POSSIBLE 'CHAINS' FROM a TO e THROUGH d)}}{\text{(SUM OF ALL POSSIBLE 'CHAINS' FROM a TO e)}}$$

This can be generalized to give us the relative likelihood of any tag T in terms of '*chains*':

$$l(T) = \frac{\text{(sum of all possible 'chains' from the last unambiguous tag to the next unambiguous tag THROUGH TAG T)}}{\text{(sum of all possible 'chains' from the last unambiguous tag to the next unambiguous tag)}}$$

CHAINPROBS actually uses a definition of the likelihood function in terms of 'chains', since it is computationally more efficient; but this new definition is entirely equivalent to the likelihood formulae previously given.

B.9 Modifying the 'one-step' formula in special cases

So far, we have assumed that the tag-likelihood function is a First-Order Markov process: we have assumed that a '*chain*' is composed of a sequence of independent 'links', *bonds* between *pairs* of tags. In trials on a section of the LOB Corpus (over 20,000 words), we found that the formulae above correctly yielded the 'best' tag for c 93-94% of words; so the 'one-step' function is in fact a very close approximation to the 'perfect' likelihood function (we were actually quite surprised that such a simple set of formulae could be

so successful!).

However, among the errors in the remaining 6-7%, there were a significant number of cases where the function clearly needed to look *two* tags backwards or forwards (rather than just one) to calculate the likelihood of a 'link' in a 'chain'. These exceptional cases fell into two main categories:

(i) tag-sequences involving a "noise-tag" such as RB (adverb), e.g. in

"she began to seductively reveal herself"
PP3A VBD TO RB VB PPL

the forward likelihood of TO is much more dependent on VB than on RB, and the backward likelihood of VB is more dependent on TO than RB. In effect, when calculating the likelihood of the tag-sequence, we would like to 'ignore' the "noise-tag" RB.

(ii) tag triples around CC (coordinating conjunction), of the form T<a> CC T<b> : Tag-triples in which T<a> and T<b> are in fact the same tag (e.g. NN CC NN, JJ CC JJ) are far likelier than tag-triples in which T<a> and T<b> differ (e.g. JJ CC NN).

The 'one-step' likelihood function can be used to calculate a likelihood figure for any sequence of three tags T1, T2, T3, essentially by multiplying B(T1,T2) * B(T2,T3). In a few special cases, this tag-triple likelihood must be modified by a *tag-triple scaling factor*, S(T1,T2,T3). These special cases are ones where the overall likelihood of the tag-triple depends on the 'bonding' of T1 and T3, rather than B(T1,T2) and B(T2,T3).

B.10 Summary of the final formula

How is S(T1,T2,T3) to be incorporated into the likelihood formulae? If the immediate context were assumed to be unambiguous, we could simply add a new factor to the formula for absolute likelihood (L(T<r,a>):

$$L(T<r,a>) = Lb(T<r,a>) * Lf(T<r,a>) * Lw(W<r>,T<r,a>) * S(T<r-1,1>,T<r,a>,T<r+1,1>)$$

To be able to deal with ambiguous contexts, we must generalize this formula to:

$$L(T\langle r,a\rangle) = Lw(W\langle r\rangle, T\langle r,a\rangle) * \sum_{\substack{i=1..n(r-1) \\ j=1..n(r+1)}} \left(\begin{array}{l} B(T\langle r-1,i\rangle, T\langle r,a\rangle) * Lw(W\langle r-1\rangle, T\langle r-1,i\rangle) * Lb(T\langle r-1,i\rangle) \\ * B(T\langle r,a\rangle, T\langle r+1,j\rangle) * Lw(W\langle r+1\rangle, T\langle r+1,j\rangle) * Lf(T\langle r+1,j\rangle) \\ * S(T\langle r-1,i\rangle, T\langle r,a\rangle, T\langle r+1,j\rangle) \end{array} \right)$$

The formulae for Lb and Lf must be similarly modified to take S into account. The above formula for L is considerably more complex than that of B.3. However, since S(T1,T2,T3) only 'comes into play' in a few special cases, the extra computation is often redundant. There is an alternative (equivalent) formula which is computationally much more efficient (even though the formula looks more complicated at first sight); it contains a separate factor dealing with S, which 'cancels out' to 1 (and can thus be ignored) in most cases. This formula is given below, in the following summary of the LOB CL Grammar tag likelihood formulae:

Relative likelihood:

$$l(T\langle r,a\rangle) = \frac{L(T\langle r,a\rangle)}{\sum_{i=1..n(r)} L(T\langle r,i\rangle)}$$

Absolute likelihood:

$$L(T\langle r,a\rangle) = Lb(T\langle r,a\rangle) * Lf(T\langle r,a\rangle) * Lw(W\langle r\rangle,T\langle r,a\rangle) *$$

$$\sum_{\substack{i=1..n(r-1)\\ j=1..n(r+1)}} \left(\begin{array}{l} B(T\langle r-1,i\rangle,T\langle r,a\rangle) * Lw(W\langle r-1\rangle,T\langle r-1,i\rangle) * Lb(T\langle r-1,i\rangle) \\ * B(T\langle r,a\rangle,T\langle r+1,j\rangle) * Lw(W\langle r+1\rangle,T\langle r+1,j\rangle) * Lf(T\langle r+1,j\rangle) \\ * S(T\langle r-1,i\rangle,T\langle r,a\rangle,T\langle r+1,j\rangle) \end{array} \right)$$

$$\sum_{\substack{i=1..n(r-1)\\ j=1..n(r+1)}} \left(\begin{array}{l} B(T\langle r-1,i\rangle,T\langle r,a\rangle) * Lw(W\langle r-1\rangle,T\langle r-1,i\rangle) * Lb(T\langle r-1,i\rangle) \\ * B(T\langle r,a\rangle,T\langle r+1,j\rangle) * Lw(W\langle r+1\rangle,T\langle r+1,j\rangle) * Lf(T\langle r+1,j\rangle) \end{array} \right)$$

Backward likelihood:

$$Lb(T\langle r,a\rangle) =$$

$$\sum_{i=1..n(r-1)} \left(\begin{array}{l} \sum_{h=1..n(r-2)} \left(\begin{array}{l} B(T\langle r-1,i\rangle,T\langle r,a\rangle) * Lw(W\langle r-1\rangle,T\langle r-1,i\rangle) * Lb(T\langle r-1,i\rangle) \\ B(T\langle r-2,h\rangle,T\langle r-1,i\rangle) * Lw(W\langle r-2\rangle,T\langle r-2,h\rangle) * Lb(T\langle r-2,h\rangle) \\ * S(T\langle r-2,h\rangle,T\langle r-1,i\rangle,T\langle r,a\rangle) \end{array} \right) \\ \hline \sum_{h=1..n(r-2)} \left(B(T\langle r-2,h\rangle,T\langle r-1,i\rangle) * Lw(W\langle r-2\rangle,T\langle r-2,h\rangle) * Lb(T\langle r-2,h\rangle) \right) \end{array} \right)$$

Forward likelihood:

$$Lf(T\langle r,a\rangle) = \sum_{j=1..n(r+1)} \left(\frac{\sum_{k=1..n(r+2)} \left(B(T\langle r,a\rangle, T\langle r+1,j\rangle) * Lw(W\langle +-1\rangle, T\langle r+1,j\rangle) * Lf(T\langle r+1,j\rangle) \right.}{\left. B(T\langle r+1,j\rangle, T\langle r+2,k\rangle) * Lw(W\langle r+2\rangle, T\langle r+2,k\rangle) * Lf(T\langle r+2,k\rangle) \right.} \right.$$
$$\left. \left. * S(T\langle r,a\rangle, T\langle r+1,j\rangle, T\langle r+2,k\rangle) \right) \right)$$
$$\left(\sum_{k=1..n(r+2)} \left(B(T\langle r+1,j\rangle, T\langle r+2,k\rangle) * Lw(W\langle r+2\rangle, T\langle r+2,k\rangle) * Lb(T\langle r+2,k\rangle) \right) \right)$$

The alternative definition of relative likelihood in terms of '*chains*' is now:

$$l(T) = \frac{\begin{array}{l}\text{sum of all possible 'CHAINS'}\\ \text{FROM the LAST unambiguous tag}\\ \text{not in the middle of a 'special case' tag-triple}\\ \text{TO the NEXT unambiguous tag}\\ \text{not in the middle of a 'special case' tag-triple}\\ \text{THROUGH TAG T}\end{array}}{\begin{array}{l}\text{sum of all possible 'CHAINS'}\\ \text{FROM the LAST unambiguous tag}\\ \text{not in the middle of a 'special case' tag-triple}\\ \text{TO the NEXT unambiguous tag}\\ \text{not in the middle of a 'special case' tag-triple}\end{array}}$$

B.11 Potential for further improvement

The current success rate of CHAINPROBS is consistently 96.5-97%. Theoretically this could be improved by adding further factors to the formulae, taking more contextual information into account by going beyond the simple 'Augmented First-Order Markov' model (CL Grammar is ideally suited to 'enhancement through feedback').

However, the law of diminishing returns suggested to us that it would probably be easier simply to correct remaining tagging-errors 'by hand' than to spend time and effort enhancing the formulae further (at least, this is quicker in the short term, for the immediate task of tagging the LOB Corpus; for new corpora, improvements may well be worthwhile).

The types of construct in which the remaining errors tend to occur are listed in the Manual Postedit Handbook (Atwell *et al.*). In general, many of these problem-cases call for 'higher-level' grammatical or semantic analysis, which would require major enhancements of the present tagging programs. Nevertheless, we feel that our remarkable success rate using such a simple model of language is highly significant.

C ADAPTING THE LOB CL GRAMMAR TO DETECT SPELLING AND GRAMMATICAL ERRORS

As explained in section A, the LOB Grammatical Tagging Programs perform a very simple grammatical analysis of input texts. This 'surface' approach makes the programs much faster than 'full-blooded' parsers; so they are ideally suited to applications where a 'basic' level of grammatical analysis is all that is required.

One such application is in the automatic detection of spelling and grammatical errors in input English texts. In this section, I shall explain how the current LOB Grammatical Tagging programs have been superficially modified to detect such errors in a short sample text; and I shall discuss what further research is required to produce an efficient general-purpose *automatic error-detection* program for commercial Word Processing applications.

C.1 Current 'spelling-checkers' do not look at context

A number of programs are currently available which claim to 'check spelling' in English texts. However, these programs are limited to simple dictionary-lookup: each input word is checked against a large Lexicon, and any word not found is assumed to be misspelt. Unfortunately, this simple method allows many errors to 'slip through' undetected: if a misspelling happens to coincide with another valid word (as in "I now how to prophecy the whether!"), then it is accepted.

Errors such as "now", "prophecy", and "whether" in the example *could* be detected by simple grammatical analysis: for example, the subordinating conjunction "whether" is easily confused with the noun "weather"; and a noun is much likelier than a subordinating conjunction in the context

"... the X!"

so "whether" is probably a misspelling of "weather" in this context.

C.2 Adapting the LOB Grammatical Tagging Programs

Notice that this sort of error can be detected simply by comparing relative likelihoods of word-tags; no higher level of grammatical analysis is required. Clearly the LOB CL Grammar is ideally suited to this kind of analysis. Only a few superficial modifications were needed to convert the current Grammatical Tagging Programs into a prototype 'context-sensitive' spelling-checker (these mainly related to input/output formats).

More important than the adjustments to the programs was the change in the role of the wordlist. In Grammatical Tagging, wordlist-lookup is just one of several methods of tag-assignment available to WORDTAG: there were a number of 'default' routines for words not found in the wordlist. In a spelling-checker, these 'default' routines are not required, in fact, they must not be used at all: if a word is not found in the wordlist, then we can assume it is a misspelling immediately, without the need for 'context-compatibility' checking. Therefore, the Lexicon of a spelling-checker must be much larger than the current LOB wordlist.

Another difference is that each entry in the Lexicon must not only contain a word's 'own' tags, but also the tags of any similar words, the *error-tags*. For example, in the sentence given above ("I now how to prophecy the whether!"), the misspelt word "prophecy" can be detected by grammatical analysis *only* if we know that it is a noun, *and* that there exists a very similar verb ("prophesy"); so the Lexicon entry for "prophecy" must give not only the word's 'own' tag NN, but also the error-tag VB:

WORD	TAG(S)	ERROR-TAG(S)
prophecy	NN	VB£

Note that error-tags are marked with £ to distinguish them from 'own' tags.

C.3 Trial run of the adapted LOB tagging programs

To put the theory to the test, a short text was devised, full of deliberate spelling mistakes which could *only* be detected by grammatical analysis. Also, a sample Lexicon was compiled, with an entry for each word in the text. This text was then processed by the adapted LOB tagging programs:

(i) VERTICALIZE put each word on a separate line (record), and also tagged punctuation marks (so these do not have to be included in the Lexicon)

(ii) WORDTAG assigned a set of tags and error-tags to each word, by Lexicon-lookup (any word not found in the Lexicon can be marked as an error at this stage)

(iii) CHAINPROBS used the Tag Likelihood function to choose the 'best' tag for each word; if an error-tag (marked £) was chosen, then this indicated a probable misspelling

(iv) LOBFORMAT (renamed MARKERRORS) 'rehorizontalized' the text, writing the message "ERROR?" underneath all words which had been 'error-tagged'.

The output from this trial run is shown in Appendix A. Almost all the errors in the text are flagged; but *none* would be uncovered by current 'spelling-check' programs.

C.4 From prototype to general-purpose program

Much research still has to be carried out to transform a 'prototype' into a general-purpose spelling-checker for commercial Word Processing packages:

(i) Compile a very large Wordlist, much bigger than the current LOB wordlist.

(ii) Modify the LOB Tagset (and Tag-Pair Bond function table): the number of tags in the current LOB Tagset is 134, but experience has shown that many tags could be 'merged' or eliminated with little loss of accuracy (many of the finer distinctions drawn in the LOB Tagset are linguistically interesting, but not required for spelling-checking); this makes the program much smaller and more efficient.

(iii) A set of potential tags must be added to every word in the Lexicon: this can be done by running WORDTAG over the untagged Lexicon, and then 'manually' checking the decisions reached.

(iv) We must design an algorithm to discover, for each word in the Lexicon, a set of 'similar' words. This algorithm must find words which have very similar spellings to the 'target' word (e.g. *now* is a common 'typo' misspelling of *know*); and also, it must find words which can easily be confused because they *sound* the same (e.g. *there* vs. *their*).

(v) Using this 'similar-word-finding' algorithm, every word in the Lexicon must be assigned a set of *error-tags*: first, a set of similar words is associated with each 'target' word; then, the tags from these similar words become the error-tags of the 'target' word.

(vi) The current LOB Tagging programs were originally written to be run on University Mainframe computers, and we paid scant attention to questions of speed and efficiency; the programs contain a number of routines which, in the light of experience, are clearly not necessary in a spelling-checker (for example, the programs are designed to collect large amounts of statistical feedback; but once a satisfactory success level is achieved, this will not be needed). Everything but the essential 'core' of the analysis can be cut out, and the suite of programs can be combined into one single program, performing the analysis in a single pass. In effect, then, the LOB CL Analysis suite must be completely rewritten, to make it much faster and more efficient.

C.5 Checking grammar and style

So far, we have only discussed *spelling* errors which can be detected by grammatical analysis. In essence, such errors are detected because the misspelling causes an incongruity in the grammatical structure of the sentence; the position of the incongruity is marked by the warning message "ERROR?", which is to be interpreted as a spelling-error.

In general, though, *any* striking grammatical incongruity is liable to be marked by the warning message "ERROR?"; and although up till now we have assumed this indicates a spelling-error, this is not necessarily so: the user of the system must be aware that this warning may be triggered by a *grammatical* infelicity (for example, if a word is not just misspelt, but accidentally missed out altogether,

then if an 'ungrammatical' sentence results, an "ERROR?" warning will be triggered.

Rather more insidious and problematic than blatantly 'incorrect' grammar is the use of obscure and unnecessarily complex grammar, which can make documents unintelligible; this is a problem of *style* rather than simple grammaticality. Fortunately, the spelling-check program is readily adapted to check 'grammatical style' as well. Currently, the tagging programs choose the 'best' analysis by comparing the *relative likelihoods* of alternative analyses. A fairly simple modification would allow us to elicit an *absolute likelihood* figure for the 'best' analysis of each sentence (normalized to fall within the range 0 to 1). This figure amounts to a measure of *'grammatical deviance'*: sentences with a normalized absolute likelihood of nearly 1 have simple, 'ordinary' grammatical structure, while sentences with a normalized absolute likelihood near zero are highly 'deviant'.

Thus, the 'Automatic Text-Checker' will not only mark out blatant errors in spelling and grammar, but it will also grade sentences along a sliding scale according to 'grammatical deviance' (sentences which fall below an 'acceptability threshold' (chosen by the user) can even be specifically marked out). Word Processors equipped with this Automatic Text-Checker will hopefully encourage the use of Plain English in official and business documents!

D CL GRAMMAR IN SPEECH SYNTHESIS AND ANALYSIS

Converting between written and spoken English is a trivial operation for humans, but has proven extremely difficult for computers. CL Analysis may prove a useful tool in tackling this problem.

D.1 Graphemic to phonemic transcription

It is generally agreed that an important stage in speech synthesis is the translation of ordinary written text into some phonetic form, in which each symbol corresponds to some specific sound. Some simple speech-synthesis systems have a straightforward dictionary-lookup algorithm to do this, using a dictionary which gives a single phonetic equivalent of each written word. A more refined version of this algorithm also has a 'default' rule-system to translate words not found in the dictionary, so that *any* input word can be assigned a

phonetic transcription; this is analogous to the default routines in WORDTAG, which ensure that *any* input word is assigned a set of potential tags.

Unfortunately, some words turn out to be 'ambiguous', in that they can have varying pronunciation and/or stress, depending on their grammatical function, e.g.:

"John wanted to *read* the paper"
vs.
"Has he *read* it yet?"

"She seems to *reject* all my advances"
vs.
"I put the *reject* in the dustbin"

A grammatical tagging algorithm could be used to disambiguate such examples. The great advantage of CL Analysis is that we do not have to analyse a whole sentence, but only the immediate context; a 'Grapheme-to-Phoneme Transcription' program could 'turn on' the CL tagging and disambiguation algorithm whenever such an ambiguity arose, but keep it 'turned off' the rest of the time.

However, if we wish to include sentence intonation in our phonetic transcription, then grammatical analysis of the whole sentence clearly *is* required. For this, the CL Grammatical Parser to be described in Section E would be a useful tool.

D.2 Speech analysis in terms of constituent-likelihood analysis

CL Analysis plays an even more important part if we view the whole process of speech analysis, from sound to written form, in terms of 'tagging', that is, assignment of 'labels' to 'constituents'.

The first step in speech analysis is to convert 'raw' sound into a digital form which can be readily manipulated by digital computer (the Lancaster University Linguistics Department has an ACT Sirius 1 computer which has this facility). Next, this 'digital sound' must be converted into a sequence of phonetic symbols; and then, the sequence of phonetic symbols must be converted into normal written English. However, these two conversion processes are far from trivial. The 'units' of speech sound (phones) are of variable length (e.g. a vowel sound is longer than a plosive), and also, the 'same'

utterance recorded several times will yield a slightly different digital recording each time. This leads to uncertainty and ambiguity in the phonetic transcription of a digital recording of an utterance. Moreover, even if we could be sure of choosing the correct phonetic transcription, converting this to normal written English is still a big problem. Again, the 'units' are of variable length (unlike written English, spoken utterances generally have nothing like a space at every word-boundary). Also, there is another level of ambiguity, e.g. *make up* and *may cup* may both be valid interpretations of a given phonetic transcription.

This second level of ambiguity can only be resolved by grammatical analysis: the 'best' interpretation must be chosen on the basis of contextual compatibility. Clearly, this problem can be tackled in terms of CL Analysis:

(i) given a phonetic transcription of an utterance, assign a set of potential written English interpretations; then

(ii) assign a likelihood to each potential 'labelling' or grapheme-string, using a Likelihood Function (L<g>) which measures the internal grammatical consistency of the grapheme-string in terms of the contextual compatibilities of the constituent graphemes (so that grapheme-strings which constitute 'grammatical' sentences are assigned higher likelihoods than grapheme-strings which involve grammatical inconsistencies).

In fact, the first level of ambiguity, encountered when moving from digital recording to phonetic transcription, can also be dealt with in terms of CL Analysis:

(i) given a digital recording of an utterance, assign a set of potential phonetic transcriptions; then

(ii) assign a likelihood to each potential 'labelling' or phone-symbol-string using a Likelihood Function (L<p>) which measures the internal lexical consistency of the phone-symbol-string in terms of the contextual compatibilities of the constituent phone-symbols (so that phone-symbol-strings which constitute a sequence of valid lexical items (words) are assigned higher likelihoods than phone-symbol-strings which involve non-existent 'words').

A great advantage of this approach is that it allows both levels of disambiguation to be combined in an integrated analysis algorithm:

we can calculate the overall likelihood that a particular grapheme-string is the correct interpretation of a given digital recording, simply by multiplying L<p> by L<g>. This is useful for two reasons:

(i) the 'best' phonetic transcription of a digital recording may turn out to be grammatically inconsistent, while a 'less likely' phonetic transcription (rejected during the first stage of disambiguation) might have had some graphemic interpretation which is grammatically 'acceptable'. In other words, if the two stages of disambiguation are separate, we may eliminate some of our options 'too early'; by disambiguating only on the basis of 'overall' likelihood, we are effectively hedging our bets until *all* relevant factors have been taken into account.

(ii) The division of the problem of speech analysis into two main subtasks, as described above, is in fact contentious; for example, many linguists would say that the transition from phonetic transcription to phonemic transcription is an important separate subtask. However, if the aim of the CL Analysis is to assign some 'overall' Likelihood figure to any given mapping between digital recording and grapheme-string, then it does not really matter how many subtasks this 'overall' process is divided into: the 'overall' Likelihood is simply a product of a number of factors, one for each subtask.

D.3 A CL Grammar of spoken English

The CL Grammar used by the LOB Corpus Tagging program suite is based on statistics derived from written English texts (initially, texts from the Brown Corpus). In a sense, we can say that the CL Grammar was 'extracted' from these texts: although we decided upon the tagset (using 'intuitive' knowledge of the important grammatical word-classes of English), the texts provided the frequency statistics which constituted the 'rules' of syntactic patterning.

The grammar of spoken English is statistically different from the grammar of written English (for example, written English tends to include more lengthy, complex sentences); the CL approach allows us to quantify these differences systematically. First, a Corpus of spoken English is needed (the London-Lund Corpus of Spoken English could be used, or alternatively, if a sufficiently general and robust speech-analysis program could be devised, we might even compile a

new Corpus using this program (the actual compilation of this new Corpus would serve as a very thorough 'test' of such a program!). This Corpus must then be grammatically analyzed, by running the present LOB Grammatical Tagging programs over it, and then 'manually' correcting the errors (many of which will be due to the imposition of a Written English Grammar over Spoken English). From the analyzed Corpus, we can then 'extract' a CL Grammar of Spoken English, by gathering the relevant frequency statistics. The differences between this CL Grammar of Spoken English and the LOB CL Grammar of Written English will be reflected in the differences in Tag-Pair Bond function values for certain tag pairs, and also in other related statistical differences such as the average Absolute Likelihood assigned to a sentence.

Thus, a speech-analysis program can be used in the compilation of a Corpus of Spoken English, from which we can 'extract' a CL Grammar of Spoken English; and this grammar will then be very useful to researchers in speech analysis and synthesis, since it is specifically geared to spoken English. Potentially, the two fields of CL Grammar and Speech Synthesis and Analysis have much to offer each other.

E CL GRAMMATICAL PARSER

The current LOB Corpus Grammatical Tagging programs assign a grammatical tag to each word in a text, showing its grammatical function; but 'higher-level' constituents are not analysed. To do this, we need a *grammatical parser*; and it turns out that it should be possible to perform a grammatical parse of the LOB Corpus using algorithms very similar to those of the present tagging-suite.

E.1 Tags and hypertags

In general, each tag in the LOB Tagset can only appear in certain syntactic (syntagmatic) positions, for example:
AT (article) comes at the start of a Noun Phrase;
IN (preposition) comes at the start of a Prepositional Phrase;
CS (subordinating conjunction) comes at the start of a Subordinate Clause;
. (full stop) comes at the end of a Sentence;

NN (singular common noun) comes

(i) at the start of a Noun Phrase *or*

(ii) at the end of a Noun Phrase *or*

(iii) within a Noun Phrase *or*

(iv) as a Noun Phrase in its own right (i.e. start *and* end of a Noun Phrase)

These syntactic positions within higher-level constituents can be symbolized by 'higher-level tags' or *hypertags*. By analogy with the present WORDLIST (a list of words and their possible tags), we could construct a TAGLIST of tags and their possible hypertags, with entries such as

tag	possible hypertags
AT	[N
.	S]
IN	[P
CS	[F
NN	N] N [N] [N@
PP$	[N
VB	[V] V] V@ [V@

etc.

(NB [V]does *not* include the object Noun Phrase, but only Verb-constituents; however, [N] *does* include subordinate prepositional phrases, etc.)

As with tags in the WORDLIST, hypertags are ordered, with @ and % markers for rare syntagmatic functions.

E.2 Hypertag-assignment

A program analogous to WORDTAG could give each tag in a sentence its appropriate hypertags, as given by the TAGLIST (this program would in fact be much simpler than WORDTAG, as there are only 134 tags in the LOB Tagset, instead of an open-ended set of possible words).

Sometimes, the hypertag(s) required is(/are) indicated better by a particular *combination* of tags, rather than by the tags taken individually. For example, IN (preposition) is 'hypertagged' [P (open prepositional phrase), and WDT (WH-determiner) is 'hypertagged' [F[N (open subordinate clause *and* open noun phrase); but the combined

tag-pair IN WDT must be 'hypertagged' [F[P [N (this is for clauses beginning "of which...", "for what...", etc.). These 'special-case' tag-pairs and their corresponding hypertag-pairs must be listed in a TAG-PAIR-LIST, analogous to the current IDIOMLIST of exceptional word combinations; a program analogous to IDIOMTAG could 'overwrite' hypertags assigned by simple TAGLIST-lookup whenever a tag-pair matches an entry in this TAG-PAIR-LIST.

Since these two 'hypertag-assignment' programs will be considerably simpler than WORDTAG and IDIOMTAG, it will be practicable to combine them into a single program: each tag-pair in a text is first looked up in the TAG-PAIR-LIST; but if no match is found, then hypertags are assigned to the tags individually, according to the TAGLIST. This unified hypertag-assignment program will be much more efficient, since unnecessary lookups are avoided, and all hypertags are assigned in a single pass.

E.3 Hypertag-disambiguation

Each record has now been assigned a set of potential hypertags. Next, a program analogous to CHAINPROBS must assign a relative likelihood figure to each hypertag in a record, using a *hypertag likelihood function* very similar to the Tag Likelihood Function described in Section B. We can then choose a single 'best' sequence of hypertags. For example, the sentence "As I was eating my lunch I decided to get a cup of coffee" would be hypertagged as follows:

WORD	TAG	HYPERTAG
----------		[S
as	CS	[F
I	PP1A	[N]
was	BEDZ	[V
eating	VBG	V]
my	PP$	[N
lunch	NN	N]
I	PP1A	[N]
decided	VBD	[V]
to	TO	[T
get	VB	[V]
a	AT	[N
cup	NN	N
of	IN	[P
coffee	NN	[N]
.	.	S]

E.3 Building a syntactic parse tree

Tags have now been grouped into higher-level constituents (N, V, S, etc.); but there are still some 'unmatched brackets'. This is because certain tags specifically mark the *start* of a higher-level constituent (e.g. CS-[F; IN-[P; AT-[N), but often there is no such corresponding 'end-of-phrase word'.

What we need now is a program which can insert extra closing brackets where needed. One way to find out where to add these brackets is to try to convert the labelled bracketing into a tree data-structure, following simple 'conversion rules':

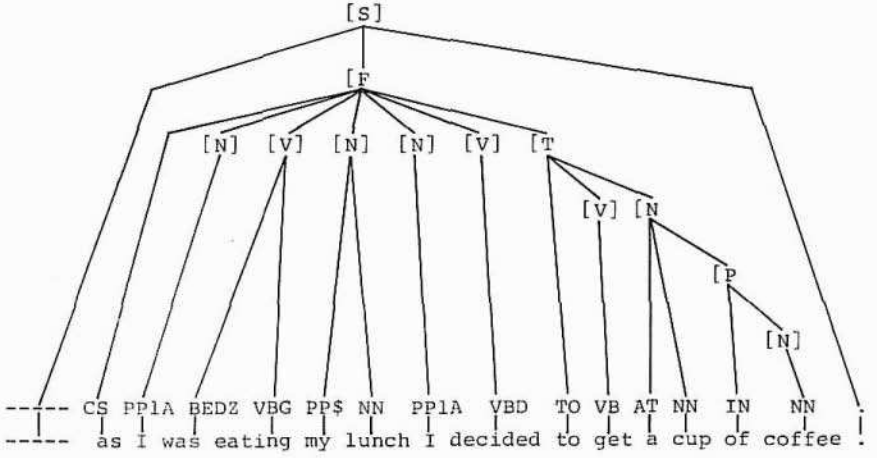
(i) X [Y means "Y is the daughter of X"

(ii) X] Y means "X is the daughter of Y"

(iii) X] [Y means "Y is the right sister of X"

(iv) X ... X is represented by a single node X *if* both Xs are at same 'level' of nested bracketing and they are not sisters . there is no][interposing between the two Xs *at the same level* as the Xs. Note that X ... Y (where X and Y are different, and X is at the same level as Y but not a sister) is *invalid*, since it requires a single node to be tagged both X and Y; this is an indication that some phrase-boundary (labelled bracket(s)) is missing.

Using such rules, we can build the following tree:



E.4 Inserting missing closing brackets

The 'conversion rules' carry on adding daughters to a node until that node's closing bracket is found; so, if the closing bracket is missing, the node will continue to have daughters attached to it until the sentence-end is reached. This means that the rightmost daughters of an 'unclosed' node are *suspect*: each non-leaf node in the tree should have at least one daughter (the first or left-most daughter), but the nodes further to the right could well be not daughters but right-hand sisters (or even 'aunts'!) of the 'unclosed' node.

An example of this is the unclosed [F node (marking a subordinate clause) in the tree above; its daughters are apparently

[N] [V] [N] [N] [V] [T

Clearly this is wrong - this sequence of daughter-constituents could not be a valid subordinate clause. The reason for this error in the tree is that the missing closing bracket F] should be inserted between *lunch* and *I*, so that the subordinate clause becomes

[N] [V] [N]

and the remaining 'daugthers' become *sisters* of [F]. However, the tree-building algorithm does not know this, so it carries on adding daughters to the unclosed [F node instead of the root [S].

Nevertheless, despite being 'lopsided', the tree built in this way is still useful. The tree shows us where missing closing brackets *might* be inserted: for example, the tree becomes well-formed only if the F] is inserted after a daughter of [F.

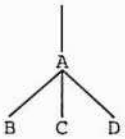
In general, an unclosed node [X with n daughters in the original tree can be 'closed' in n different ways, leading to n different parse-subtrees. So, if an 'unclosed' tree such as the one shown above has q 'unclosed' hypertag-nodes [H<1>, [H<2>, [H<3>, ..., [H<q>, where

[H<1> has n<1> daughters
[H<2> has n<2> daughters
[H<3> has n<3> daughters
⋮
[H<q> has n<q> daughters

then there are (n<1>*n<2>*n<3>*...*n<q>) *potential parse-trees*.

E.5 Choosing the 'best' parse-tree

The final stage in parsing is a program which, starting from an 'unclosed' tree such as the one above, generates all possible parse-trees and compares the likelihood of each (note the analogy with CHAINPROBS: this program effectively generates all possible tag-sequences and compares their likelihoods; the difference is that now we are dealing with trees rather than simple strings). To do this, we must be able to associate a likelihood with a potential parse-tree; this is done using a *hypertag-node likelihood function* Lhn which assigns any given node a likelihood figure dependent on its daughter nodes and their likelihoods in turn. If a node A has daughters B, C, D:



then at A we must store the likelihood that BCD is a 'valid' A (the *constituent likelihood* Lc(A,BCD)), multiplied by the hypertag-node likelihoods of B, C, and D in turn:

Lhn(A) = Lc(A,BCD) * Lhn(B) * Lhn(C) * Lhn(D)

This recursive definition allows us to calculate a likelihood figure for the root [S] node which takes into account all nodes and subtrees in the parse-tree.

E.6 The phrase dictionary

Values of the Constituent-Likelihood function Lc are stored in a *Phrase Dictionary*, which states, for each of the higher-level constituents (N, S, V, P, etc.), the set of possible 'daughter-constituent-sequences', along with the relative likelihood of each possible sequence. For example, the Phrase Dictionary will tell us that, in a subordinate clause (hypertagged [F]), the following daughter-constituent-sequences are very likely:

CS [N] [V] [N]

CS [N] [V]

; the following sequences are less likely, but still possible:

CS [N] [V] [P]

CS [N] [V] [T]

; but the following sequences are *very* unlikely:

CS [N]

CS [V]

CS [N] [N]

Any daughter-sequences not found in the Phrase Dictionary get a very low default probability (just above zero); in this way, we are ensured of *some* analysis for *any* sentence (the analogy in CHAINPROBS is that the Tag-Pair Bond function always has a value greater than zero, to ensure that no potential tag is ever assigned a zero likelihood; see Section B.5).

E.7 A parse in three passes

To summarize, the CL Grammatical Parser outlined above will build a syntactic parse-tree in three passes. First, every tag in a text is assigned a set of potential hypertaggings, using a *tag-pair-list* and *taglist*. Secondly, the set of hypertags at each tag is disambiguated, by eliminating all but the likeliest hypertag-sequence; this is done using a *hypertag-likelihood function* very similar to the tag-likelihood function currently used by CHAINPROBS. Thirdly, this 'disambiguated' hypertag-sequence is converted into a set of potential parse-trees, where each potential parse-tree has the missing closing brackets inserted differently; a *hypertag-node likelihood function* is used to compare likelihoods of competing potential parse-trees.

In the final output, it will probably be useful to include not only the single 'best' parse-tree, but also a number of 'runners-up' (say three), in case the 'best' parse is found to be incorrect in postediting. This can be done quite easily, if we adopt an output format similar to that shown in Section E.3: there are columns for *word*, *tag*, and *hypertag*; and in addition, we need three more columns to show the three 'likeliest' combinations of inserted closing brackets. For example, the final output of the 'parse' of our earlier example sentence might be:

WORD	TAG	HYPERTAG	THREE LIKELIEST PARSES		
			59%	39%	2%
----------		[S			
as	CS	[F			
I	PP1A	[N]			
was	BEDZ	[V			
eating	VBG	V]			
my	PP$	[N			
lunch	NN	N]	F]	F]	F]
I	PP1A	[N]			
decided	VBD	[V]			
to	TO	[T			
get	VB	[V]			
a	AT	[N			
cup	NN	N]		N]	N]T]
of	IN	[P			
coffee	NN	[N]	P]N]T]	P]T]	P]
.	.	S]			

This representation may seem difficult to understand at first; but hopefully posteditors will soon get to grips with it. The great advantage is of course the economy of space: to show three potential analyses, we do not need three complete trees.

E.8 Residual problems

Finally, it must be remembered that, of course, not all sentences will be as straightforward as the example above! There are many problems not touched upon (e.g. when the phrase-boundary is not explicitly marked, as in "I gave the baby milk to drink"); but then, *any* approach to syntactic parsing will encounter difficulties with these and other stumbling blocks. The success rate of CHAINPROBS turned out to be much higher than we expected; the lesson to be learnt was that in the 'real' language found in a corpus, very few 'pathological cases' actually turn up! Therefore, we have every hope that the CL Grammatical Parser will also be very successful.

F OTHER APPLICATIONS OF CL GRAMMAR

As explained in Section A.3, CL Grammar is generally applicable to many different forms of linguistic analysis. So far we have not explored all the possibilities: for example, CL Analysis may also be useful in formal semantic analysis. Other applications will doubtless suggest themselves as our research continues.

In general, we hope we have shown that *statistical, probabilistic* methods of analysis *do* have a place in linguistics, and specifically in the field of syntax. Furthermore, statistical analysis should *not* be seen simply as a 'heuristic' to fall back on when all else fails; CL analysis is entirely based on probabilities, and the Tagged LOB Corpus will be overwhelming evidence that this approach works.

REFERENCES

Atwell, Eric Steven. 1982. 'LOB Corpus Tagging Project: Manual Pre-edit Handbook'. Department of Linguistics and Modern English Language and Department of Computer Studies, University of Lancaster.

Atwell, Eric Steven. 1982. 'LOB Corpus Tagging Project: Manual Post-edit Handbook (A mini-grammar of LOB Corpus English, examining the types of error commonly made during automatic (computational) analysis of ordinary written English)'. Department of Linguistics and Modern English Language and Department of Computer Studies, University of Lancaster.

Francis, W. Nelson and Henry Kučera. 1964 (rev. eds. 1971 and 1979). *Manual of Information to Accompany a Standard Sample of Present-Day Edited American English, for Use with Digital Computers.* Department of Linguistics, Brown University.

Garside, Roger and Geoffrey N. Leech. 1982. 'Grammatical Tagging of the LOB Corpus: General Survey'. In Stig Johansson, ed. *Computer Corpora in English Language Research.* Norwegian Computing Centre for the Humanities, Bergen.

Greene, Barbara and Gerald Rubin. 1971. *Automatic Grammatical Tagging of English.* Department of Linguistics, Brown University.

Johansson, Stig, Leech, Geoffrey N. and Helen Goodluck. 1978. *Manual of Information to Accompany the Lancaster-Oslo/Bergen Corpus of British English, for Use with Digital Computers.* Department of English, University of Oslo.

Johansson, Stig and Mette-Cathrine Jahr. 1982. 'Grammatical Tagging of the LOB Corpus: Predicting Word Class from Word Endings'. In Stig Johansson, ed. *Computer Corpora in English Language Research.* Norwegian Computing Centre for the Humanities, Bergen. 118-146.

Leech, Geoffrey N., Garside, Roger and Eric Steven Atwell. 1983. 'The Automatic Grammatical Tagging of the LOB Corpus' (pp. 13-33 in this issue of *ICAME News*).

Marshall, Ian. 1982. 'Choice of Grammatical Word-Class without Global Syntactic Analysis for Tagging Words in the LOB Corpus'. Department of Computer Studies, University of Lancaster.

Peterson, James. 1980. 'Computer Programs for Detecting and Correcting Spelling Errors'. In *Communications of the Association for Computing Machinery*, 23, 12. 676-87.

```
ATWELL1 my farther was very crawl . he bald at me if I dud anything wrong , and
          ERROR?          ERROR?     ERROR?          ERROR?

ATWELL2 sometimes he would hot and bit me , until I was so week and miserable
                           ERROR? ERROR?                    ERROR?

ATWELL3 that I wanted to due . finally , won day , I decided to got my won back
                         ERROR?                                 ERROR?

ATWELL4 on him : *' I 'll mike him pay ; he will n't get away with this ! **' I stole
                          ERROR?

ATWELL5 a meat clever , and I maid several dense in his hid with it ! it must
            ERROR?          ERROR?     ERROR?          ERROR?

ATWELL6 have hurt a lit ! son *' the gruesome tame of Eroc Attwell **' appeared
                    ERROR? ERROR?                ERROR?

ATWELL7 in all the papers . perhaps my friends would learnt to spell my name
                                                  ERROR?

ATWELL8 correctly at last !


END OF LISTING OF FILE :ENA001.ERIC(1,*,1).EXAMPLE6(5) FOR USER :ENA001 AT 1983/03/18__
```