CS-95-25

Investigations into the Grammar Underlying the Penn Treebank II

R. Gaizauskas

Investigations into the Grammar Underlying the Penn Treebank II

Robert Gaizauskas

1 Introduction

Given a bracketed corpus like the Penn Treebank II (Marcus et al., 1995) an obvious thing to want to do is to extract the grammar underlying the bracketing and investigate some of its properties. How many rules are there? What is their distribution by category? What are their individual frequency distributions? How long are the righthand sides of the rules? In addition to properties of the grammar, it is also of interest to examine properties of the corpus, as revealed by the grammar. How deep are the trees? What are the bounds on tree depth by sentence length?

This note provides answers to these and other questions, reporting the results of some straightforward investigations using the Penn Treebank II corpus, available from the Linguistic Data Consortium[1]. The author's original motivation in undertaking these investigations was to determine the feasibility of using the extracted grammar for parsing texts similar to those in the corpus. This interest has coloured the sorts of questions asked and prompted simplifying assumptions made in order to derive 'useful' results. Thus, this investigation is partial at the very best and leaves many questions about this valuable resource unanswered.

The programs used to derive the results are freely available: short descriptions of them and instructions for retrieving them by anonymous *ftp* are included in an appendix.

2 The Corpus and Tagging Scheme

These investigations consider only the Wall Street Journal portion of the Penn Tree Bank II (PTB II). This comprises 2,337 files containing 49,208 sentences consisting of 1,253,013 tokens. Each text (file) has been separately part-of-speech tagged and parsed by the University of Pennsylvania team resulting in two files for each text, one containing tagged versions of the text and one containing the parsed (bracketed) version of the text. Using PTB I anyone who wanted to derive the full grammar needed to write a program to merge the tagged and parsed files (a program to do this is described in (Krotov, Gaizauskas, and Wilks, 1994)). With PTB II, however, merged files have been supplied (as well as separate tagged and parsed files). The remainder of this note considers these merged files only.

In PTB I there were 48 part-of-speech or word class tags, 14 syntactic or constituent tags, and 4 null element tags, according to Marcus, Santorini, and Marcinkiewicz (1993) (this was in theory; in the actual data extra tags crept in, apparently due to annotator inconsistency). In PTB II the tagging scheme has been made more sophisticated with a view to annotating predicate-argument structure. In this scheme there is a base set of tags consisting of 45 part-of-speech tags, 26 syntactic tags, and according to Marcus et al. (1995), 6 null element tags (though there are 10 to be found in the data). Additionally, however, constituent and null elements tags from the base set may now have multiple extra tags attached to them by either a hyphen or an equal sign. These tag extensions fall into several classes:

- hyphen-attached functional tags indicating such things as

[1] `http://www.ldc.upenn.edu/`

  - text category (e.g. headline (-HLN), list marker (-LST)),
  - grammatical function (e.g. subject (-SBJ), topicalised constituent(-TPC)),
  - semantic role (e.g. location (-LOC), manner(-MNR));

- hyphen-attached numeric co-indices indicating the connection between tagged null elements and the lexical material for which they stand (transposing the lexical material to the position of the null element is intended to allow for straightforward predicate-argument interpretation);
- equals-attached co-indices indicating how elements in gapped clauses should be mapped onto constituents in the parallel ungapped clause.

Here is an example of a PTB II-style annotation (sentence 2 in `wsj_0601.mrg`):

```
( (S
    (NP-SBJ-1 (DT A) (NN judge) )
    (VP (MD must)
      (VP
        (VP (VB jump)
          (PP-DIR-2 (IN from)
            (NP (NN murder) ))
          (PP-DIR-3 (TO to)
            (NP (JJ antitrust) (NNS cases) )))
        (, ,)
        (VP
          (PP-DIR=2 (IN from)
            (NP (NN arson) ))
          (PP-DIR=3 (TO to)
            (NP (NNS securities) (NN fraud) )))
        (, ,)
        (PP (IN without)
          (S-NOM
            (NP-SBJ (-NONE- *-1) )
            (VP (VBG missing)
              (NP (DT a) (NN beat) ))))))
    (. .) ))
```

Grammatical functions are indicated by the `-SBJ` and `-NOM` tags and semantic roles by the `-DIR` ('direction') tag. The null element `*-1` is co-indexed with the `NP-SBJ-1` 'A judge' indicating that this material needs to be interpolated in the position of the null element for predicate-argument interpretation. The `PP-DIR=2` annotation indicates that the structure of the gapped clause is to be recovered from the parallel one in which the `PP-DIR-2` tag occurs.

3 What Do We Count?

If one is interested solely in the 'surface' context-free grammar revealed in the bracketing with a view to using it for parsing, then the extra annotation appearing in PTB II may not be of interest. Certainly the numeric coindexing tags must be removed as they flag nonlocal dependencies that cannot be captured in phrase structure rules employing only the simple (atomic) syntactic categories used in the bracketing. Further, the choice of actual numbers used for coindexing in a given sentence does not carry any significance. Failure to remove them leads to a repetition of structurally identical rules distinguished only by variances in the choice of numeric co-indices. However, one may want to remove or alter other aspects of the tagging as well, depending on one's interests.

3.1 Tag Extensions

The text category, grammatical function, and semantic role tag extensions may or may not prove to be of use to a parser. Whether they are may depend on other information being available to it, say from lexical representations. For example, compare *John wore trousers in Leeds* with *John wore trousers in September*. These give rise to the two PP annotations (PP-LOC (IN in) (NP (NNP Leeds))) and (PP-TMP (IN in) (NP (NNP September))). If our grammar contains separate PP rules for each of these tag extensions – i.e. PP-LOC → IN NP and PP-TMP → IN NP then unless lexical NNPs come marked with LOC and TMP information, we will generate two analyses for each of these sentences with no information to decide between them. On the other hand, if we do have semantic feature information stored with our lexical items, then we can propagate this information upwards to the mother node, if we so choose, and there would appear to be no need to have separate rules. However, this ignores the possibility that some PP rules only occur with LOC or only with TMP roles – in this case having lexical semantic role information would mean that by separating PP-LOC and PP-TMP rules where they are different we would generate fewer inappropriate analyses. Further, since PP-LOC and PP-TMP constituents may occur in different contexts, i.e. in the righthand sides of different higher level rules, distinguishing PP-LOC and PP-TMP rules could again result in fewer inappropriate analyses.

All of this leaves us with problems when setting out to extract a grammar from PTB II or to count the rules occurring therein. Do we count PP-LOC → IN NP, PP-TMP → IN NP, and PP → IN NP as three rules or as one (all three occur in PTB II)? Do we count NP → NP PP-LOC, NP → NP PP-TMP, NP → NP PP as three rules or one (again, all three occur in PTB II)? Should we merge PP tags and their hyphenated extensions when they occur in rules which are otherwise indistinguishable and retain them when they do not all occur in parallel rules? [2]

3.2 Null Elements

A further problem arises with regard to the treatment of null elements. One of the motivations for including null elements in the PTB II analyses was to provide traces which may be coindexed with the relevant source lexical material so as to facilitate predicate-argument interpretation. Once the coindices are removed, as we have argued above they must be in any attempt to arrive at the context-free grammar underlying the PTB II, then this motivation for retaining null elements in the grammar disappears. Since parsing using a grammar with substantial occurrences of null elements can be problematic (null elements may be hypothesized anywhere, leading to an explosion of inappropriately proposed constituents) there are strong practical motivations for removing them, if one is setting out to acquire a grammar to be used by a parser.

For example, consider this PP from our previous sample bracketed sentence (from which we have removed the coindex as we are now supposing):

```
(PP (IN without)
  (S-NOM
    (NP-SBJ (-NONE- *) )
    (VP (VBG missing)
      (NP (DT a) (NN beat) ))))
```

If we allow a (bottom-up) parser to hypothesize an NP null element before every VP then we will hypothesize an S constituent whenever a VP is found – clearly an expensive business. The matter gets worse when one sees in how many other contexts in the corpus null elements may also occur[3].

[2] Where a nonterminal may or may not take an additional hyphenated tag, it is not clear how occurrences *without* the tag are to be interpreted. Suppose, for example, a PP tag occurs in a rule without a semantic role tag. Does this mean that the role player in the PP must *not* be either of type LOC or TMP since otherwise we would expect to see this marked? This is not made clear in Marcus et al. (1995).

[3] It may be possible to mitigate the effects of empty category rules in a bottom-up parser through careful algorithm design – see Moore and Dowding (1991), for example. However, their approach crucially relies upon employing richer representations of categories (unification grammars) to permit non-local information about gap contexts to control when null elements get hypothesised. For an atomic categoried context-free grammar of the sort we propose to extract from the PTB II such an approach is not an option.

Are there, however, reasons for keeping null elements? Again it depends on what one wishes to do with the extracted grammar. If it is to be used in investigating the distribution of various phenomena as identified in the linguistic theory of movement adopted by the coders then clearly the null elements are invaluable. Further, regardless of whether the actual displaced material can be recovered (which becomes impossible once co-indices are removed), the null elements do provide a particular theory's view of where 'real' argument positions occur. Thus, for example Marcus et al. (1995)'s example:

```
(SBARQ (WHNP-1 What)
    (SQ is
        (NP-SBJ Tim)
        (VP eating
            (NP *T*-1)))
    ?)
```

shows, amongst other things, that a transitive verb is required in this context. Finally, even without the coindices the null elements serve the purpose of indicating the syntactic type of material to be located elsewhere in the sentence in order for semantic interpretation to proceed. However, that this is useful may in turn be debated. The bulk of null elements are NP's marking, e.g., subject position in infinitival or gerundive phrases or object position in passive constructions. In these cases semantic interpretation of the VP implicitly requires the sort of syntactic object the trace marks explicitly. This requirement must be encoded in the rules for semantic interpretation anyway and hence the need for explicit marking via a null element appears to be redundant. So, in the above example, though the phrase structure does not explicitly supply a subject for missing we can infer, from its being an active transitive verb with an NP following in object position, that there must be an NP subject around somewhere. So the explicit trace marker, once its coindex is removed, tells us nothing further. A similar argument can be made for null elements which are not NP's.

It should be noted that there is a wide range of phenomena involving null element markers in the PTB II and of course one need not adopt an all-or-nothing approach to including or excluding null elements from an extracted grammar. That is, in extracting a grammar one could choose to exclude certain null elements and to retain others, depending on one's theoretical sympathies with the approach taken towards marking certain phenomena. For example, one might choose to retain the wh- and topicalisation traces, while removing the more transformational grammar style traces, such as passive and raising or equi subject traces.

If null elements are removed then in many cases the phrase structure rules in which they appear reduce to unary rules. In our running example, for instance, if we remove the null element we are left with:

```
(PP (IN without)
  (S (VP (VBG missing)
         (NP (DT a) (NN beat) ))))
```

Extracting the structural rules from this annotated phrase leaves us with, amongst others, the rule S $\rightarrow$ VP. One of the reasons we argued for the removal of the NP null element was to avoid hypothesizing an S constituent every time we discovered a VP; but now by admitting an S $\rightarrow$ VP rule we appear driven to exactly to same result. This suggests that when the removal of a null element leads to the creation of a unary structure we should merge that structure into the next highest level. So, in our example, we could remove the S constituent altogether to arrive at:

```
(PP (IN without)
    (VP (VBG missing)
      (NP (DT a) (NN beat) )))
```

As with the tag extensions, therefore, we see that there are several ways one might decide to extract grammar rules from the PTB II corpus depending on how one chooses to treat null

elements. One could leave them in place and extract the phrase structure grammar complete with null element rules exactly as they stand. Or one could remove the null element constituents (or a subset of them), and leave the embedding structures in which they occur, even when these reduce to unary rule structures. Or one could remove null elements and in addition merge unary structures into the higher level structures in which they appear, so eliminating many unary rules. Our motivation for extracting a grammar from the PTB II annotations will determine which option we feel is most appropriate.

3.3 The Approach Adopted

Of course these options need not be exclusive: once the options have been distinguished one may extract grammars with or without tag extensions and with or without some or all null elements. They may be compared directly, or in various applications. In this report, however, I have chosen to report on just one choice of options for the sake of brevity. The programs that have been developed to extract the grammar are flexible enough to allow any of the other options discussed above to be selected equally easily and the author hopes interested readers will chose to do so. The general flavour of the results presented below will almost certainly carry over to other options.

Given the motivation of wanting to extract a relatively simple grammar for use in an automatic parser, the results presented in the rest of this paper are based on a corpus derived from the PTB II corpus through the following steps:

1. eliminate all hyphen- or equals-attached suffix tags;
2. eliminate any constituents whose label is `-NONE-` (all null element tags in PTB II are uniformly treated as lexical tags occurring in a unary constituent whose label is `-NONE -`);
3. if as a result of 2. any constituent now has an empty set of children then remove this constituent (this may percolate recursively up the tree);
4. if as a result of 3. any constituent now has a single child which is labelled with a nonlexical tag then remove the constituent and then merge the child into parent's position in the next higher level constituent (this has the effect of removing unary rules whose righthand sides are not word class tags).

This process may be illustrated by reference to the last `PP` in the preceding example. After step 1 we have:

```
(PP (IN without)
  (S
    (NP (-NONE- *) )
    (VP (VBG missing)
        (NP (DT a) (NN beat) ))))
```

after step 2:

```
(PP (IN without)
  (S
    (NP  )
    (VP (VBG missing)
        (NP (DT a) (NN beat) ))))
```

after step 3:

```
(PP (IN without)
  (S (VP (VBG missing)
         (NP (DT a) (NN beat) ))))
```

after step 4:

```
(PP (IN without)
    (VP (VBG missing)
        (NP (DT a) (NN beat) )))
```

4 The Programs

The results reported in the next sections were obtained by using a suite of Perl programs written by the author. Further details of the programs may be found in the Appendix and the programs are freely available to anyone who should want them. Here we give a brief overview of the key programs and their capabilities.

exgrule This is the principle program for extracting grammar rules. It reads one or more PTB II merged files, extracts the grammar rules, and keeps counts of how many times each rule occurs, and in how many sentences. Options allow hyphen and equals attached tags to be filtered/not filtered, null elements to included/removed, and unary rules with non-preterminal righthand sides to be retained/ absorbed. Further options allow information about tree statistics to be gathered – tree depth and depth bound by sentence length. Finally, an index maybe created that records for each extracted rule pointers to all sentences in which it occurs. This index is used by the findgrule program.

findgrule Assuming an index has been created while running `exgrule`, the `findgrule` program prints all sentences in the indexed PTB II files in which a supplied grammar rule occurs. This is of use when manually investigating the grammar, in order to recover specific examples of particular constructions.

ptbcounts/ptbruledist These programs process the extracted grammar rule file produced by `exgrule` and yield the summary information reported in the following sections. Options allow for the computation of rule and rule occurrence totals by syntactic category, distribution of rule occurrences by rank, rate-of-growth of rule set figures, and distribution of righthand side lengths.

5 Results: Grammar Characteristics

5.1 How Many Rules Are There?

Using `exgrule`, each grammar rule in the PTB II was extracted together with the number of times it occurred and the number of sentences in which it occurred (the latter may be a smaller number, since some rules occur multiple times in the same sentence – knowing this number is of use if you want to answer the question "How many sentences in the corpus would I fail to parse correctly if I didn't have this rule?"). As indicated above, all hyphen- and equals-attached tags were removed from the initial syntactic tags and all null constituents were discarded and resulting unary rules collapsed.

The totals for number of rules falling into each of the PTB II's 26 syntactic categories are summarised in Table 1 together with the number of occurrences of rules in each category and the number of sentences in which rules of each category occur. Totals are displayed at the bottom, and indicate the total number of rules, total number of rule occurrences, and, in the sentence occurrence column, not the total number of sentence occurrences, but the total number of sentences (hence the figure at the bottom of this column is not the sum of the figures above it, since many rules will occur in each sentence). The percentage columns indicate, for each category, the percentage of total rules in that category, the percentage of total rule occurrences in that category, and the percentage of total number of sentences in which rules of that category appear (so no total for the final column is meaningful). Rounding to the nearest integer value means some percentages round to zero. These figures were obtained using the `ptbcounts` program with the `-s` option.

5.2 Are These All the Rules?

How likely is it that these are 'all' the rules needed to analyse *Wall Street Journal* English? Given as many rules as the preceding section has revealed, it seems unlikely that every last rule has been discovered, but we might hope that most of them have been. One way to get a feeling for how

SYNCAT	RULES	%	RULE-OCCS	%	SENT-OCCS	%
ADJP	691	4	16908	2	13144	27
ADVP	340	2	24503	3	18769	38
ADVP|PRT	1	0	1	0	1	0
CONJP	10	0	367	0	343	1
FRAG	271	2	495	0	424	1
INTJ	27	0	159	0	151	0
LST	10	0	70	0	56	0
NAC	57	0	543	0	523	1
NP	7237	41	378662	44	49110	100
NX	163	1	1680	0	558	1
PP	401	2	116713	13	42152	86
PRN	296	2	2972	0	2779	6
PRT	10	0	3238	0	3102	6
PRT|ADVP	1	0	1	0	1	0
QP	375	2	11452	1	8167	17
RRC	17	0	52	0	51	0
S	2059	12	87714	10	46860	95
SBAR	223	1	24744	3	18662	38
SBARQ	81	0	263	0	253	1
SINV	263	1	2563	0	2534	5
SQ	145	1	374	0	353	1
UCP	212	1	592	0	583	1
VP	4416	25	180116	21	47669	97
WHADJP	11	0	66	0	65	0
WHADVP	22	0	2539	0	2422	5
WHNP	127	1	9058	1	8095	16
WHPP	5	0	477	0	473	1
X	63	0	187	0	133	0
TOTALS	17534	100	866511	100	49208	-

Table 1: Rules per Category in PTB II

close the PTB II grammar might be to being complete is to see at what rate new grammar rules are being discovered as new texts are processed (the 'accession rate'). One would expect that the more texts are processed the smaller the number of new rules being added per text, i.e. that we were asymptotically approaching a complete grammar for this text type.

PTB II comes divided into 25 sets of files, each set containing up to 100 texts. These provide a natural subdivision of the corpus to use in analysing the incremental increase in grammar rules. For each of these 25 sets, which it should be noted are not precisely identical in size, the grammar rules occurring in the set were extracted and incremental sums of rules in each category, as well as of total rules were computed. The results of this exercise are displayed displayed graphically in Figure 1 plotted as number of rules against number of sentences. Separate plots are shown for rules of all categories and for rules of the five categories with the greatest number of rules – `S`, `NP` and `VP`, `ADJP` and `PP`.

5.3 Which Rules Occur Most Frequently?

From Table 1 it is obvious there is a large number of rules. We may wonder how much of the work is being done by how many rules. I.e. what is the frequency distribution of rule occurrences by rule within a syntactic category? The `exgrule` program counts rule occurrences and the `sortgrule` program sorts the resultant rule file by syntactic category and within that by number of rule occurrences. While there is far too much information to display here, the top 10 rules from the most frequently occurring categories – `S`, `NP`, `VP` and are displayed in Table 2.

Perhaps more interesting than seeing which rules occur most frequently is getting a feeling for how many rules in each category account for how many rule occurrences in that category. This information has been collected in two ways. First, by simply sorting all the rules by frequency of occurrence within category, as described above one can produce a frequency distribution by rank. The frequency distribution of the top ten rules in all categories is shown in Table 3 with

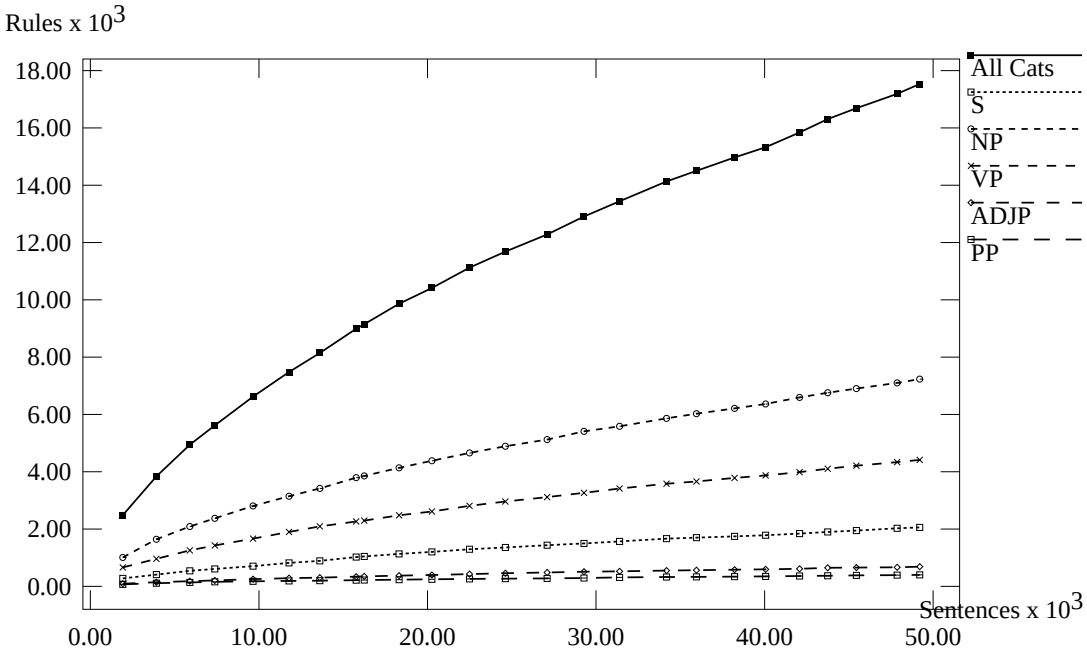

Figure 1: Rate of Growth of Rule Set Size

S -> NP VP	NP -> NP PP	VP -> TO VP	PP -> IN NP	ADJP -> JJ
S -> NP VP .	NP -> DT NN	VP -> VB NP	PP -> TO NP	ADJP -> RB JJ
S -> PP , NP VP .	NP -> PRP	VP -> MD VP	PP -> IN VP	ADJP -> JJ PP
S -> NP ADVP VP .	NP -> NN	VP -> VBZ VP	PP -> IN QP	ADJP -> CD NN
S -> CC NP VP .	NP -> NNS	VP -> VBN PP	PP -> TO QP	ADJP -> JJ VP
S -> ADVP , NP VP .	NP -> NNP	VP -> VBD VP	PP -> VBG NP	ADJP -> RB VBN
S -> S , CC S .	NP -> NNP NNP	VP -> VBD NP	PP -> IN PP	ADJP -> JJ CC JJ
S -> NP ADVP VP	NP -> DT JJ NN	VP -> VBZ NP	PP -> ADVP IN NP	ADJP -> ADJP PP
S -> S , NP VP .	NP -> JJ NNS	VP -> VBG NP	PP -> VBG PP	ADJP -> RBR JJ
S -> SBAR , NP VP .	NP -> DT NNS	VP -> VBP VP	PP -> IN SBAR	ADJP -> JJR

Table 2: Ten Most Frequent Rules for S, NP, VP, PP, ADJP in PTB II

the figures normalised as a percentage of the total number of rule occurrences in that category in each case. The final totals column indicates what percentage of rules occurrences in each category is accounted for by the top ten rules.

Figure 2 shows another view of rule occurrence distribution. Here rule occurrences are shown by rank for all ranks with greater than one rule occurrence for the five categories with the most rules (S, NP, VP, ADJP, and PP). These are shown on a doubly logarithmic scale.

Finally, one may divide the rules for each category into 10 ranks (the first rank containing the 10% of the rules occurring most frequently, the next rank containing the 10% of the rules occurring next most frequently, and so on). Results of this analysis are shown in Table 4.

5.4 How Long are the Right Hand Sides?

The distribution of rule occurrences by righthand side length is shown in Figure 3. Two outliers, of length 32 and of length 51 have been discarded – each occurred exactly once. Ignoring these, the righthand sides ranged in length from 1 to 21 with 4 the median value.

SYNCAT	RANK										Total %
	1	2	3	4	5	6	7	8	9	10	Coverage
ADJP	22	11	9	5	5	3	3	2	2	2	64
ADVP	72	4	3	2	2	1	1	1	1	1	88
ADVP|PRT	100	-	-	-	-	-	-	-	-	-	100
CONJP	42	29	15	6	3	2	2	1	0	0	100
FRAG	7	4	3	3	2	2	2	1	1	1	26
INTJ	52	16	9	4	2	2	2	1	1	1	90
LST	37	21	19	6	4	3	3	3	3	1	100
NAC	37	20	12	5	3	2	2	2	1	1	85
NP	11	10	6	4	4	4	4	3	3	2	51
NX	21	11	9	6	6	5	5	4	3	3	73
PP	80	8	3	1	1	1	1	1	0	0	96
PRN	19	18	7	4	4	3	3	2	2	2	64
PRT	94	3	3	0	0	0	0	0	0	0	100
PRT|ADVP	100	-	-	-	-	-	-	-	-	-	100
QP	44	16	6	3	2	2	2	2	1	1	79
RRC	27	23	10	8	6	4	4	2	2	2	88
S	40	25	4	2	2	1	1	1	1	1	78
SBAR	46	27	9	7	2	2	1	1	1	1	97
SBARQ	19	14	11	5	5	3	3	2	2	1	65
SINV	43	8	8	5	5	3	2	2	1	1	78
SQ	14	8	6	5	3	3	3	2	2	2	48
UCP	12	5	4	4	4	3	3	2	2	2	41
VP	9	6	5	4	4	3	3	3	3	3	43
WHADJP	77	8	3	2	2	2	2	2	2	2	100
WHADVP	93	2	2	1	1	0	0	0	0	0	99
WHNP	57	31	5	1	1	1	0	0	0	0	96
WHPP	95	4	0	0	0	-	-	-	-	-	99
X	24	12	7	6	4	3	3	2	2	2	65

Table 3: Frequency Distribution of 10 Most Frequent Rules

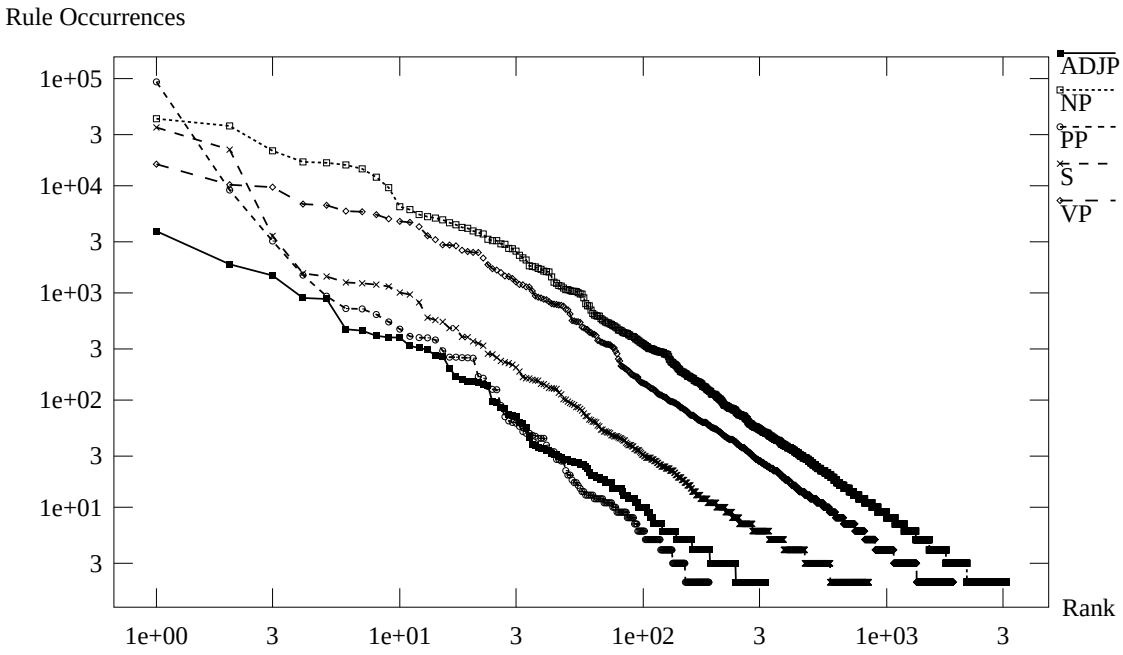

Figure 2: Distribution of Rule Occurrences by Rank

SYNCAT	RULES/DECILE	1	2	3	4	5	6	7	8	9	10
ADJP	70	91	4	2	1	1	0	0	0	0	0
ADVP	34	96	2	1	0	0	0	0	0	0	0
ADVP|PRT	1	100	-	-	-	-	-	-	-	-	-
CONJP	1	42	29	15	6	3	2	2	1	0	0
FRAG	28	41	13	8	5	5	5	5	5	5	5
INTJ	3	76	8	4	3	2	2	2	1	1	1
LST	1	37	21	19	6	4	3	3	3	3	1
NAC	6	80	8	3	2	2	1	1	1	1	1
NP	724	96	2	1	0	0	0	0	0	0	0
NX	17	83	6	3	1	1	1	1	1	1	1
PP	40	99	1	0	0	0	0	0	0	0	0
PRN	30	83	6	3	2	1	1	1	1	1	1
PRT	1	94	3	3	0	0	0	0	0	0	0
PRT|ADVP	1	100	-	-	-	-	-	-	-	-	-
QP	38	91	4	2	1	1	0	0	0	0	0
RRC	2	50	17	10	6	4	4	4	2	2	2
S	206	96	1	1	0	0	0	0	0	0	0
SBAR	22	98	1	0	0	0	0	0	0	0	0
SBARQ	8	63	9	6	3	3	3	3	3	3	3
SINV	26	86	4	2	2	1	1	1	1	1	1
SQ	15	55	12	7	4	4	4	4	4	4	4
UCP	21	56	12	7	4	4	4	4	4	4	4
VP	442	95	2	1	0	0	0	0	0	0	0
WHADJP	1	85	3	2	2	2	2	2	2	2	2
WHADVP	2	97	2	0	0	0	0	0	0	0	0
WHNP	13	97	1	0	0	0	0	0	0	0	0
WHPP	1	95	4	0	0	0	-	-	-	-	-
X	6	60	10	7	3	3	3	3	3	3	3

Table 4: Distribution of Rule Occurrences by Decile

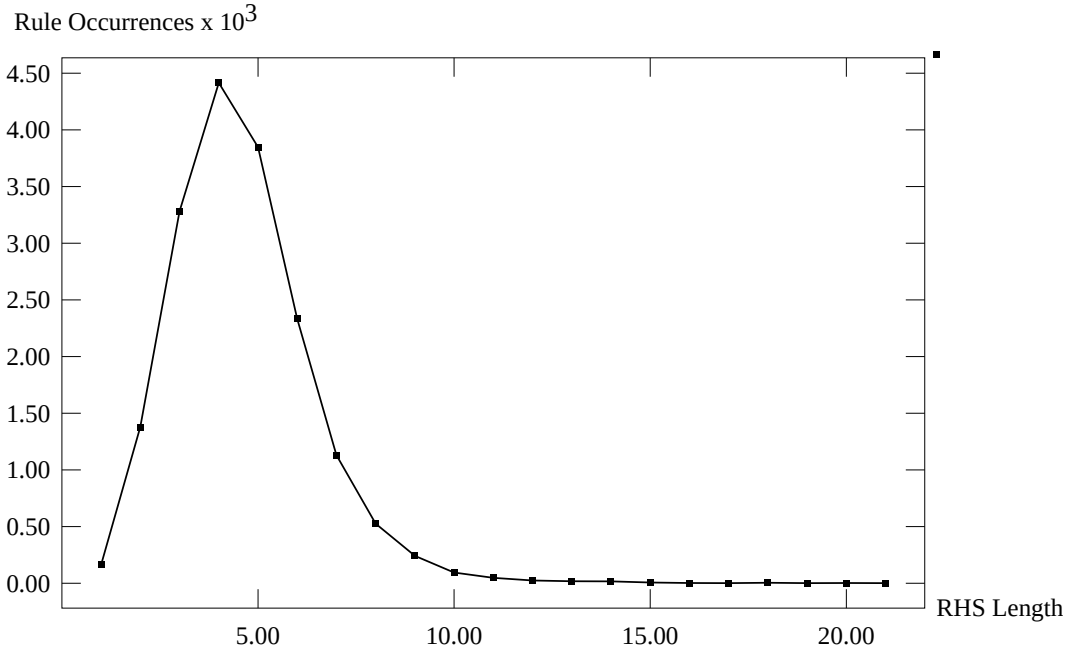

Figure 3: Distribution of Righthand Side Rule Length

6 Results: Corpus Characteristics

In addition to determining various characteristics of the grammar of the PTB, as detailed in the preceding section, various characteristics of the corpus are revealed by the treebanking exercise. Knowing these facts could be of use in designing parsers, where implementing depth bounds could significantly improve performance.

6.1 How Deep are the Trees?

The distribution of tree occurrences by depth is shown in Figure 4. Tree depths ranged from 2 to 36 with a median value of 9.

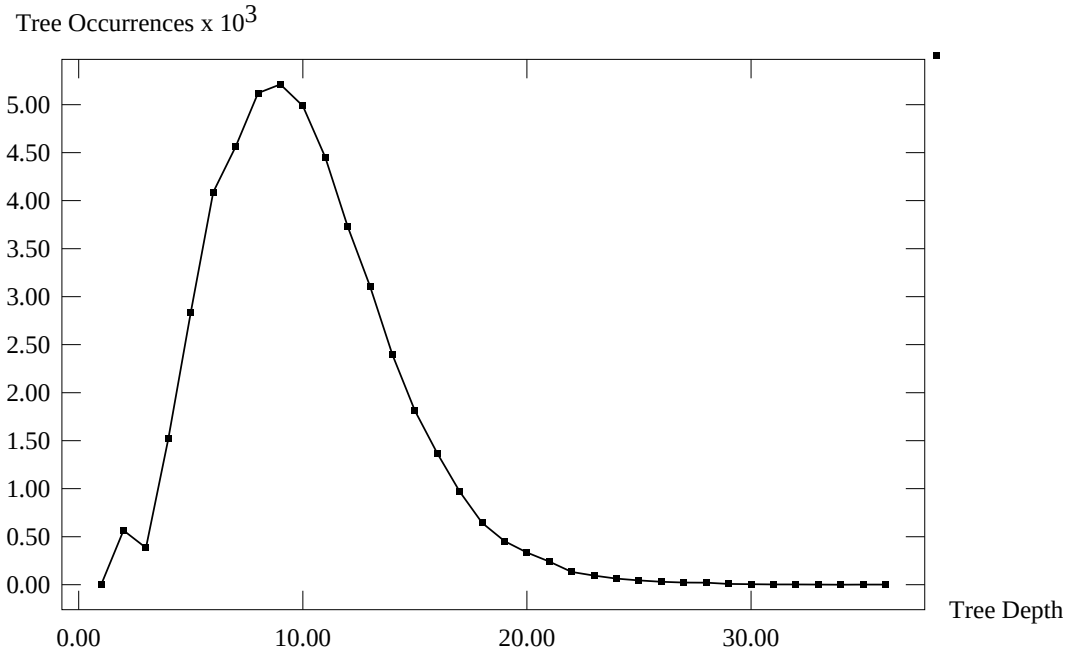

Figure 4: Distribution of Tree Occurrences by Depth

6.2 What are the Bounds on Tree Depth by Sentence Length?

Bounds on tree depth by sentence length – the deepest tree found for a sentence of a given length – are shown in Figure 6 [4]. But first it is interesting to see what the distribution of sentence lengths across the corpus. This is shown in Figure 5. Ten outliers, ranging in length from 106 to 271 have been discarded – each occurred exactly once, save one (of length 125) which occurred twice. Every sentence length from 1 to 96 occurred at least once and the median value was 21.

Tree depth bounds ranged from 2 to 36. As sentence length grows beyond about 60 this data gets chaotic – at this point the number of sentences of this length is dropping rapidly, hence the figures may be becoming unreliable.

[4] It should be noted that these tree depths are the depths not of the original trees in the tree bank, but of those obtained by eliminating null elements and unary rules with nonlexical righthand sides, following the procedure described in section 3.3.

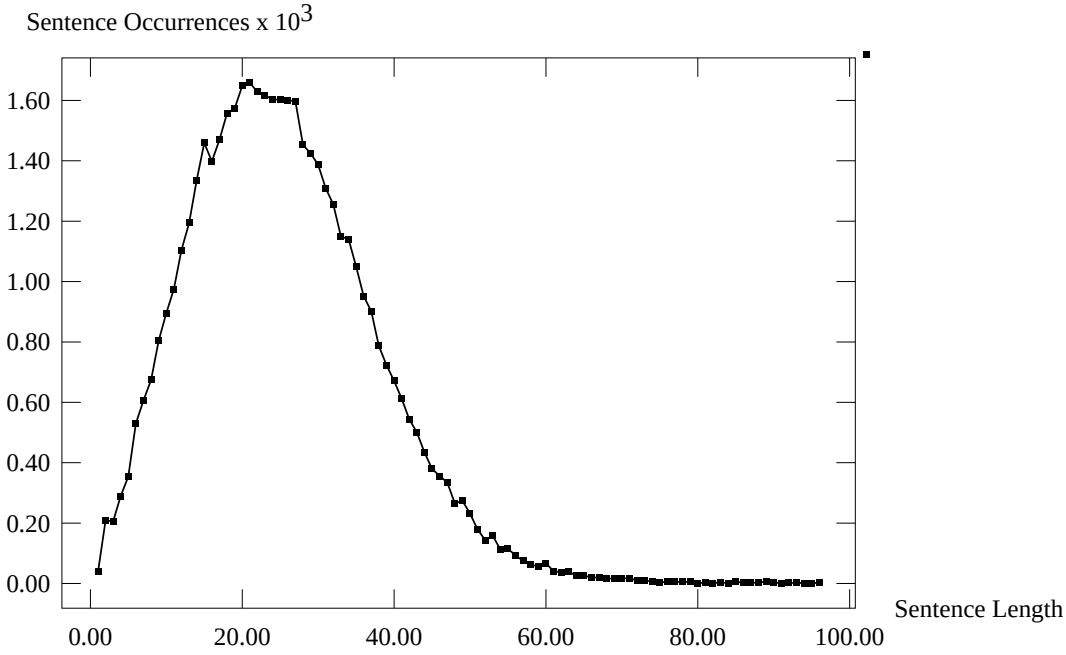

Figure 5: Distribution of Sentences by Length

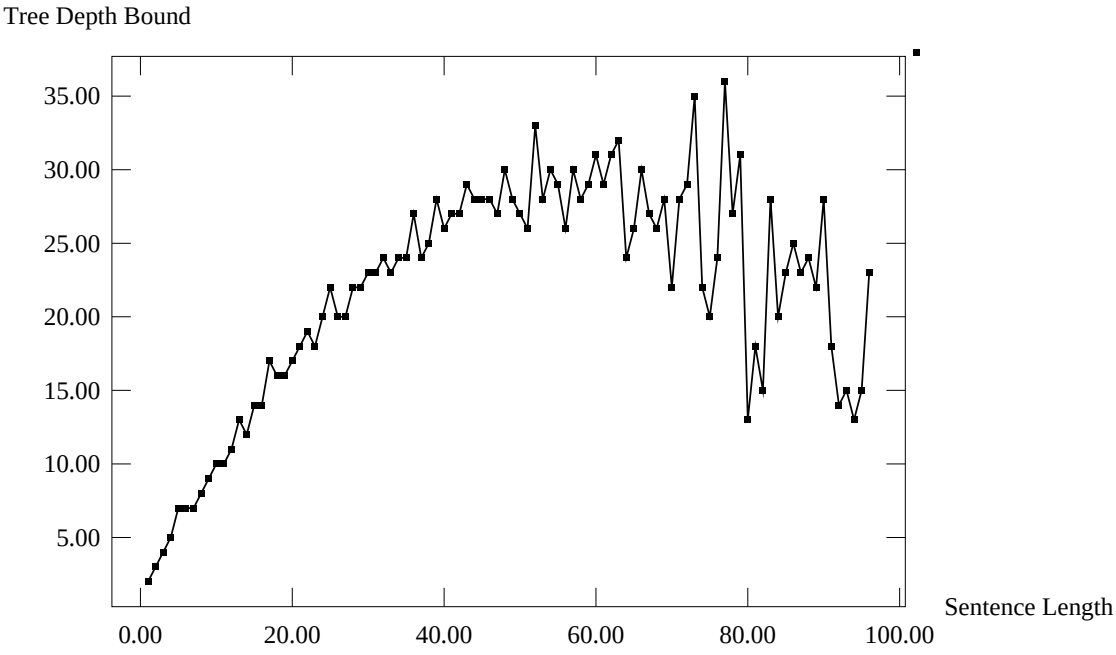

Figure 6: Tree Depth Bound by Sentence Length

7 Discussion

The absolute number of rules is surprising. Approximately 17,500 rules are required to analyse just under 50,000 sentences – or about one distinct rule for every three sentences. Perhaps even more surprising (Figure 1) is the continued rate of rule growth, which does not show convincing signs of having levelled off even after 50,000 sentences have been analysed.

It is difficult to know how to interpret this. One possible explanation is that not enough text has been tree-banked. On this view, given another n sentences, one would reach a levelling off point after which the grammar would cease to grow. Just what n might be is, of course, impossible to know *a priori*. Another possible explanation is that the coders were simply not able to perform a consistent or detailed enough analysis across the corpus – that there is a smaller grammar than the one extracted here that accounts for all the sentences, but that the coders kept introducing new rules when existing rules were in fact appropriate. Some evidence for this view is found in the large number of low occurrence NP rules with very long righthand sides, where more embedded structure could easily have been added in the analysis using existing rules. For example, consider the NP rule

```
NP -> PRP$ ADJP NNP NNP NNP NNP NN VBN NN NN
```

which occurs exactly once in the corpus and is derived from the following:

```
(NP (PRP$ its)
    (ADJP (RB newly) (VBN formed) )
    (NNP Enron) (NNP NGL) (NNP Partners) (NNP L.P.)
        (NN master) (VBN limited) (NN partnership) (NN subsidiary) )
```

Alternate analyses would add more structure within this phrase, for example making an embedded NP of the proper name *Enron NGL Partners L.P.* (the rule `NP -> NNP NNP NNP NNP` occurs in the NP rule set, as does a rule `NP -> VBN NN NN`). Of course, there is ambiguity concerning the modification relationships in such complex NPs and it appears that the coders' response in such cases was to be noncommittal by adopting very shallow analyses.

A final possible explanation is that there is no finite grammar to be obtained by this sort of treebanking and extraction procedure because there is some sort of meta-generative process going on, whereby new sentence structures that cannot be reduced to a phrase structure grammar are continually being created. Just how one might go about attempting to discover such a set of rules for generating rules is not at all clear.

However, what is clear (Tables 3, 4 and Figure 2) is that a small number of the rules account for most of the rule occurrences, and that this is true fairly uniformly across syntactic categories. The rank-frequency distributions uncovered by this analysis for the high rule-numbered categories are very reminiscent of the rank-frequency distributions of words that Zipf (1935) discovered held across many human languages. Indeed, the graphical presentation of these results in Figure 2 is uncannily similar to Zipf's graphical results (see, e.g., (Zipf, 1935) pp. 44-45). This could be interpreted as indirect support for the radical lexicalist view that grammar rules are no more than properties of lexical items [5].

For most parsers, 17,500 rules is simply too many to contemplate. A number of avenues suggest themselves for reducing this number. Given that many of the rules occur so infrequently, one could apply a simple thresholding mechanism to prune rules from the grammar. This could be applied to remove all rules that occur fewer than n times, or to remove rules that account for fewer than $n\%$ of rule occurrences or sentence occurrences of rules in each category. Simple experiments to this end reveal that a grammar accounting for 95 % of rule occurrences within each category consists of 2144 rules, while one accounting for 90 % of rule occurrences contains only 872 rules. For grammars accounting for 80 % and 70 % of rule occurrences the figures are

[5] I am indebted to Gerald Gazdar for having suggested this connection between the results presented here and Zipf's work, and for the further observation that these parallels are in conformity with the radical lexicalist view.

240 and 112 respectively [6]. It should be noted that a threshold that retains only rules accounting for $n\%$ of rule occurrences does not guarantee that $n\%$ of the sentences in the corpus will remain correctly parsable: since the rules removed may occur in separate sentences, fewer than $n\%$ of the sentences may remain parsable after pruning.

By adopting such a simple thresholding technique, various grammars were extracted for use within the parser of an information extraction system designed to work on *Wall Street Journal* texts. Experiments in trading off between parsing time and grammar size were carried out and eventually a grammar was adopted that consisted of just over 100 rules but which accounted for 70 % of the rule occurrences in the PTB. This was then manually modified to deal with various anomalies. See Gaizauskas et al. (1995) for more details.

A less crude technique, and one that throws light on other aspects of the grammar and the coding procedures, is to attempt to parse the righthand sides of the rules with other rules in the grammar [7]. If the righthand side of a rule R in grammar G is parsable by remainder of the grammar $G - R$ then any string of terminals which could be derived from G is clearly also derivable from $G - R$. Thus, eliminating R from G will not affect the language generated by G. The derivations available from $G - R$ will be a subset of those available from G and may exclude the most linguistically plausible (or may exclude some that were linguistically implausible). In particular carrying this approach to the extreme could result in the elimination of rules which generate alternative structures for constructions which are genuinely ambiguous (for example ambiguous PP attachment constructions).

Further work needs to be carried out to see just how much such compaction techniques may achieve. At the very least they will afford an insight into how ambiguous the grammar is, and could be used to drive a manual process of reducing the number of long, flat rules in the grammar.

The results shown in Figure 6 concerning bounds on tree depth by sentence length were used to limit the search of a bottom-up chart parser which employed the pruned grammar described above. In practice this appeared to have very little effect. While no controlled experiment was carried out, the simple expedient of recording whenever search was terminated due to encountering a depth bound revealed that this was happening only extremely rarely.

8 Conclusions

We have investigated various properties of a context-free grammar extracted from the PTB II, using a set of programs written for this purpose. This grammar is not the only one which could have been extracted. Any grammar extracted from this bracketed corpus requires the adoption of a number of assumptions whose alteration would lead to different results. In our case we made numerous simplifying assumptions, dispensing with null elements, and disregarding semantic tags. These assumptions may not be shared by others, but sufficient flexibility has been built into the extraction programs that others should be able to extract grammars under differing assumptions without much difficulty.

The results show a surprisingly large grammar (approximately 17,500 rules) and one that appeared to be growing at a nearly constant rate over the addition of further material to the corpus. However, in nearly all syntactic categories most rule occurrences are accounted for by a very small number of rules, in a distribution which appears Zipfian. One possible explanation for the large number of rules is that in contexts where the annotators were unsure of the syntactic structure they created long rules that avoided issues of the internal structure of constituents. Further exploration of this hypothesis, and of others, is necessary to more fully understand the nature of the grammar marked on the PTB II.

[6] The procedure adopted during thresholding was to accumulate in a set the nonlexical categories appearing on the righthand sides of rules after the initial pruning, and check whether all possible nonlexical categories were still represented somewhere in this set. If any nonlexical category had disappeared completely then all rules with it appearing on the lefthand side were removed from the grammar, as they could never be reached in a derivation. By the time a grammar accounting for 90 % of the rule occurrences within each category had been reached, for example, all rules in the categories CONJP, FRAG, LST, RRC, UCP, WHADJP and X had been removed.

[7] I am endebted to Mark Hepple for suggesting this idea.

A usable grammar has been derived from the extracted grammar by using a simple thresholding mechanism on rule occurrence frequency to eliminate low frequency rules. While no explicit evaluation has been carried out on this grammar, it has been used reasonably successfully within an information extraction system. This demonstrates the viability of using a manually bracketed corpus to acquire a grammar.

Acknowledgements The author would like to thank Gerald Gazdar, Mark Hepple and Mike Johnson for very helpful comments made on drafts of this report, and Alex Krotov for pointing out a bug in the code.

References

Gaizauskas, R., T. Wakao, K. Humphreys, H. Cunningham, and Y. Wilks. 1995. University of Sheffield: Description of the LaSIE system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 207–220. Morgan Kaufmann.

Krotov, A., R. Gaizauskas, and Y. Wilks. 1994. Acquiring a stochastic context-free grammar from the Penn Treebank. In *Proceedings of the Third International Conference of the Cognitive Science of Natural Language Processing (CSNLP94)*, pages 79–86, Dublin, July.

Marcus, M., G. Kim, M.A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K.Katz, and B. Schasberger. 1995. The Penn Treebank: Annotating predicate argument structure. Distributed on The Penn Treebank Release 2 CD-ROM by the Linguisitc Data Consortium.

Marcus, M.P., B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Moore, R. and J. Dowding. 1991. Efficient bottom-up parsing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 200–203.

Zipf, G. K. 1935. *The Psycho-Biology of Language*. Houghton Mifflin, Boston. Republished by MIT Press, Cambridge, MA, in 1965.

Appendix: Details of the Programs

The programs used to produce the results in this paper may be obtained by `ftp` from the author's web page: `http://www.dcs.shef.ac.uk/people/r.gaizauskas/`

The following is a copy of the `README file` that accompanies the programs.

```
This directory contains PERL scripts for processing the Penn Treebank II (PTB II).

exgrule -- extract the grammar rules from the combined (.mrg) files of the
    PTB II corpus.

    Input:  files containing bracketed sentences as per PTB II
       optionally [-r rule_file] a file of previously extracted grammar rules
        to which those extracted from the current input will be added (no real
        need to use this option as the mergegrule utility provides the same
        functionality)
       optionally [-x index_file] a previously created index file (for use with
        the findgrule utility) which will be augmented by the current input
       optionally [-t tree_stat_file] a file name into which to write stats
        about tree depth and depth bound by sentence length
       optionally [-c tag_check_file] a file name into which to write all
        syntactic and POS tags found during the analysis (requires [-f] flag )
```

```
       optionally [-f] filtering - and = attached tags from base tags (this
        may also be achieved using the filtertags utility, but should be done
        here if indexing on tag-free rules is required)
       optionally [-n] a flag to indicate grammar rules with null RHS are NOT
        to be ignored (default is to ignore them and to remove the LHS of such
        rules from higher level rules)
       optionally [-u] a flag to indicate unary rules whose RHS is a syn tag
        (i.e. not a POS tag) are NOT to be ignored -- default is to ignore
        them and replace the LHS of such a rule by its RHS in the next higher
        level rule
       optionally [-l] a flag to log on STDERR each file being processed

    Output: a stream of newline separated records of the form:
            LHS -> RHS+::rule_occ_count:rule_occ_sent_count:
       where   LHS is a PTB constituent tag
               RHS is a PTB constituent tag or POS tag
               rule_occ_count is a count of how many occurrences of this rule
                there have been
               rule_occ_sent_count is a count of how many sentences this rule
                has occurred in
       optionally an index file for use with the findgrule utility

    Usage:  exgrule [-r rule_file] [-x index_file] [-fnul] bracketed_text_files

exscript -- Shell script to run exgrule over all files in the PTB II.

    Runs exgrule over all files in each of the 25 directories in the PTB in
    turn, then runs mergegrule, sortgrule, mergeindex, mergets, and mergetags on
    the output of exgrule.

    Requires user to set shell variable to the root of the PTB II tree on his
    system (i.e. to the mount point of the CRDOM)

filtertags -- Get rid of - and = attached tags (but save  -> and -NONE- !).

    Can be used to postprocess a grule file extracted by exgrule rather than
    rerunning exgrule with the -f option. Will produce a grule file with
    multiple occurrences of the same rule. Hence output must be processed by
    mergegrule to merge these multiple occurrences and preserve count information.

    Input: one or more grule files as produced by exgrule

    Output: a stream of grules

    Usage: filtertags grulefile(s)

findgrule -- find sentences containing a grammar rule in the combined (.mrg)
    files of the PTBII corpus using an index built by the exgrule utility.

    Input: a CF grammar rule in the form 'A -> B C D' and an index file to the
        PTB II as build by exgrule -x.

    Output: All sentences of PTB II in which the rule occurs.
```

```
    Usage:  findgrule 'grammar_rule' ptb_index_file

mergegrule -- merge one or more grammar rule files extracted from PTB II by
    exgrule (i.e. sum counts for multiple occurrences of the same rule).

    Input: one or more grule files as created by exgrule

    Output: a stream of grule records

    Usage: mergegrule grulefile(s)

mergeindex -- merge two or more index files as created by the exgrule program.

    Input:  index files as created by exgrule -x

    Output: a single index file

    Usage:  mergeindex index_files

mergetags -- merge one or more tag check files extracted from PTB II by exgrule.

    Input: one or more tag check files as created by exgrule -c

    Output: a merged tag check file

    Usage: mergetags tag_check_file

mergets -- merge one or more tree stat files extracted from PTB II by exgrule.

    Input: one or more tree stat files as created by exgrule -t

    Output: a merged tree stat file

    Usage: mergets tree_stat_file

nullel -- extract the null elements from the combined (.mrg) files of PTB II.

    Can be used to extract occurrences of lines with -NONE- from the PTB II,
    so as to see which null elements are actually used.

    Input: one or more PTB II .mrg files

    Output: a stream of lines with -NONE- in them

    Usage: nullel bracketed_text_file(s)

ptbcounts -- perform various counting functions on the grammar rule file
    extracted from the PTB by the exgrule utility.

    Input:  One or more grule_files extracted from PTB II by exgrule
       Optionally [-s] print sums of number of grules and grule ocurrences by
        syntactic category (use ALL for all categories)
       Optionally [d num_ranks] print distributions of occurrences of grule by
        number of ranks within category (use num_ranks = 10 for deciles)
```

```
       Optionally [-i syncat] print incremental figures of number of grules in
        a series of grule files (to gauge rate of rule number growth across
        subsets of PTB; use ALL for all categories)
       Optionally [-r] print the distribution of righthand side of grules lengths
       Optionally [-R rhs_len] print rules with righthand sides of length rhs_len
       Optionally [-x index_file] for sentence occurrence totals to also occur
        in tables

    Output: Tables of figures as per input option

    Usage:  ptbcounts [-d num_ranks] [-R rhs_len] [-i syncat] [-x index_file]
                [-rs] grule_files

ptbruledist -- print rule occurrences by rank within category using a sorted grule
    file as produced by sortgrule.

    Duplicates some of ptbcounts functionality, but works much faster as it
    presumes a sorted grule file.

    Input:  a sorted grule file as produced by sortgrule
       Optionally [-n] produce occurrence figures as a normalised percentage of
        the whole (default is to produce raw figures)
       Optionally [-f threshold] filter all rules which occur threshold or
        fewer times
       Optionally [-r num_ranks] produce figures only for the top num_ranks ranks
       Optionally [-t] produce output as a table with one row per syncat and
        one column per rank -- should be used with -r and a reasonable value
        for num_ranks.

    Output: a file with for each category rule occurrences in ranked order with
        one rank and frequency figure per line (suitable for input to xgraph)

    Usage: ptbruledist [-nt] [-r num_ranks] [-f filter_threshold] sorted_grule_file

sortgrule -- sort a grule file alphabetically by LHS category and within this
    by frequency of rule occurrence.

    Input:  file containing grammar rules and counts as created by exgrule

    Output: file containing sorted grammar rules

    Usage:  sortgrule grulefile

stripcounts -- remove the rule occurrence and sentence occurrence counts from
    grule files created by exgrule.

    Can be used to derive a simple CFG from a grule file.

    Input:  file containing grammar rules and counts as created by exgrule

    Output: file containing grammar rules with counts removed

    Usage:  stripcounts grulefile
```