Parsing (with) Punctuation etc.

Ted Briscoe (briscoe@xerox.fr)
Rank Xerox Research Centre
Grenoble Laboratory
France

Abstract

In this paper, I describe an approach to robust domain-independent syntactic parsing of unrestricted naturally-occurring (English) input. The technique involves parsing sequences of part-of-speech and punctuation labels using a unification-based grammar coupled with a probabilistic LR parser. I describe the coverage of several corpora using this grammar and report an experiment to derive a probabilistic LR parser for the grammar from bracketed training data. I describe a systematic and declarative text grammar for English and its (modular) integration with the syntactic grammar. I evaluate the contribution of punctuation to deriving an accurate syntactic analysis through experiments with the trained parser on identical texts either with or without naturally-occurring punctuation marks. I briefly outline how the resulting system might be used to acquire an accurate valency / argument structure dictionary. [1].

1 Introduction

This paper is part of a continuing effort to develop a robust, practical domain-independent syntactic parser capable of yielding the one correct analysis for unrestricted naturally-occurring input. Our goal is to develop a system with performance analogous to extant part-of-speech tagging and disambiguation systems (e.g. Church, 1988) returning a syntactic analysis from which predicate-argument structure can be recovered and which can support semantic interpretation. The requirement for a domain-independent analyser obliges us to utilise statistical techniques to resolve ambiguities, whilst the latter goal obliges us to utilise a more sophisticated grammatical formalism than is typical in statistical approaches to robust analysis of corpus material. I argue that this probabilistic and grammatical model can be utilised successfully with grammars encoding constraints about punctuation and part-of-speech sequences.

Briscoe & Carroll (1993) describe an approach to probabilistic parse selection using a realistic unification-based grammar of English consisting of approximately 800 phrase structure rules (written in the Alvey Natural Language Tools (ANLT) formalism (Briscoe *et al.* 1987) which produces rules in a syntactic variant of the Definite Clause Grammar formalism (DCG, Pereira & Warren 1980). The ANLT grammar is linked to a lexicon containing about 64,000 entries for 40,000 lexemes, including detailed valency information appropriate for

[1]This research was partially supported by ESPRIT BRA 7315 'The Acquisition of Lexical Knowledge' (ACQUILEX II) and by SALT/SERC IED4/1/5808 'The Integrated Language Database'

the grammar, built semi-automatically from the *Longman Dictionary of Contemporary English* (LDOCE, Procter 1978). The resulting parser is efficient and capable of rapidly constructing a parse forest and returning the ranked n-best analyses for sentences of up to around 30 words (Carroll & Briscoe 1992; Carroll 1993).

The probabilistic model is a refinement of probabilistic context-free grammar (CFG) conditioning CFG rule application on LR state and lookahead item. A maximally informative CFG 'backbone' is derived automatically from the ANLT grammar (in which all categories are represented as feature bundles). This backbone is used to construct a generalised, non-deterministic LR parser (e.g. Tomita 1987) based on a LALR(1) table. Unification of the 'residue' of features not incorporated into the backbone grammar is performed at parse time in conjunction with reduce operations. Unification failure results in the reduce operation being blocked and the associated derivation being assigned a probability of zero. Probabilities are assigned to transitions in the LALR(1) action table via a process of supervised training based on computing the frequency with which transitions are traversed in a corpus of parse histories constructed using an unlabelled-bracketing derived from a treebank or a user-driven, interactive version of the parser. The result is a probabilistic parser which, unlike a PCFG, is capable of probabilistically discriminating derivations which differ only in terms of order of application of the same set of CF backbone rules (within a context defined by the LALR(1) table) but which remains a stochastic first-order Markov model, because the LALR(1) table defines a non-deterministic finite-state machine (FSM) and the total probability of an analysis is computed from the sequence of transitions taken to construct it.

There are three major problems to be overcome in the development of a practical automatic parsing technology: text chunking, ambiguity resolution, and undergeneration (see Briscoe 1994 for further discussion). Preliminary experiments with our system (mostly on dictionary definitions) have demonstrated that the approach can rank parses with a 75% chance that the correct analysis will be the most highly ranked (Carroll, 1993). Examination of the failures revealed structural preferences for 'low' or 'local' attachment (see Whittemore *et al.* 1990) of PPs, and for compound analyses of noun sequences, as well as other more idiosyncratic stuctural misanlyses. The assignment of correct analyses to noun compounds and multiple PPs will require lexical as well as structural preferences, as these ambiguities are resolved by the semantic interpretation of the lexical items involved (see e.g. Hindle & Rooth 1992 for one approach to integrating such lexical information into a statistical model). However, the biggest source of error on unseen input was the lack of appropriate valency (subcategorisation information) for lexical items (mostly verbs) which will always prevent the system from finding the correct analysis, and sometimes prevents it from finding any analysis at all. The work described below is intended to firstly circumvent and ultimately resolve this problem of undergeneration; in §3 I describe an approach to 'shallow' parsing which does not rely on the availability of a lexicon containing valency information, but which allows the recovery of such information for subsequent fuller syntactic analysis or construction of improved lexicons.

Our first experiments largely avoided the problem of segmenting input into units amenable to syntactic analysis, which I refer to as 'chunking', by parsing self-contained and clearly delimited dictionary definitions (which by convention most often reduce to a single syntactic constituent) with all punctuation removed. At first sight, it appears that chunking trivially reduces to extracting text sentences (delimited by capitalisation and periods) from running text and tokenising them for input to the parsing system. However, for most text genres such a strategy will fail frequently. For example, the simple (and artificial) text sentences in (1a,b,c,d) all convey a similar narrative, but only the first will be successfully analysed by a combination of naive chunking and syntactic parsing.

(1) a Max fell. John had kicked him.

b Max fell; John had kicked him.

c Max fell – John had kicked him.

d Max fell – John had kicked him – and he died.

In each of (1a,b,c,d), there is a discourse relation of explanation or elaboration, indicating a causative relationship between the second and first events described in the narrative. But in no case, are the two clauses describing the events related syntactically. Therefore, a parser searching for a *syntactic* relation between these clauses will fail even if supplied with a perfect grammar. Most extant systems attempt to treat all punctuation analogously to syntactic uses of commas or they ignore it. In §2, I describe a more satisfactory approach which makes full use of information derivable from punctuation to both chunk input and constrain the range of possible syntactic analyses.

2 Punctuation Analysis

The analysis of punctuation is essential to a solution to the chunking problem, however, it can also contribute to resolving structural ambiguity. Nunberg (1990) argues that the punctuation system is, firstly, systematic and linguistic, in the sense that it obeys principles and constraints of a type familiar from work on other linguistic subsystems such as phonology or syntax, and secondly, it is a separate linguistic subsystem not reducible to principles of (prosodic) phonology or syntax. Nunberg develops a partial grammar of English text sentences which incorporates many constraints that (ultimately) restrict the syntactic and semantic interpretation of the text. For example, one such constraint is that textual adjunct clauses introduced by colons scope over following punctuation, as (2a) illustrates; another is that textual adjuncts introduced by dashes cannot intervene between a bracketed adjunct and the textual unit to which it attaches, as in (2b).

(2) a *He told them his reason: he would not renegotiate his contract, but he did not explain to the team owners. (vs. but would stay)

b *She left – who could blame her – (during the chainsaw scene) and went home.

I have developed a succinct, declarative grammar in the ANLT formalism, based on Nunberg's procedural description. This grammar captures the bulk of the text-sentential constraints described by Nunberg with a grammar which compiles into 26 object grammar (DCG format) rules.[2] The grammar treats all words uniformly and groups them iteratively into flat textual units according to surounding punctuation; for example, the text sentence in (3a) is tokenised as in (3b) and receives four analyses of which that in (3c) is correct.[3]

(3) a A Yale historian, writing a few years ago in the Yale review, said: "we in New England have long since segregated our children".

b
```
a_wd Yale_wd historian_wd _pco write_wd a_wd few_wd year_wd ago_wd
in_wd the_wd Yale_wd review_wd _pco say_wd _pcl we_wd in_wd new_wd
England_wd have_wd long_wd since_wd segregate_wd our_wd child_wd
```

c
```
(T/txt-sc1/--
 (T/t_ta_t (T/w a_wd Yale_wd historian_wd)
  (Ta/comma _pco
   (T/w write_wd a_wd few_wd year_wd ago_wd in_wd the_wd
                                       Yale_wd review_wd)
   _pco)
  (T/t_ta-cl_t (T/w say_wd)
   (Ta/colon _pcl
    (T/txt-sc1/--
     (T/w we_wd in_wd new_wd England_wd have_wd
       long_wd since_wd segregate_wd our_wd child_wd))))))
```

The skeletal description shown in (3c) uses mnemonic rule names, based on Nunberg's description such as text unit (T), text adjunct (Ta) and text-sentence (T/txt-sc) which indicate that this text sentence consists of two text units (*a Yale historian*, *say: ...*) with an intervening text adjunct delimited by commas (*writing... ago*) and the final text unit contains a text adjunct introduced by a colon (*we... children*). Such analyses are a useful first step in robust parsing as they clearly demarcate some of the syntactic boundaries in the text sentence and thus constrain the possible analyses that a syntactic parser can assign. In other cases, this step is indispensable because it also identifies the units for which a syntactic analysis should, in principle, be found; for example, in (4a), the absence of dashes would mislead a parser into seeking a syntactic relationship between *three* and the following names, whilst in fact there is only

[2] I do not deal with sentence-final periods or with quotation since these are paragraph-level punctuation markers.

[3] These and the following examples are drawn from the Susanne corpus (Sampson, 1994) or the Spoken English Corpus (Taylor & Knowles, 1988).

a discourse relation of elaboration between this text adjunct and pronominal *three*. The correct analysis in (4b) – one of six found by the system – makes this relationship clear.

(4) a The three – Miles J. Cooperman, Sheldon Teller, and Richard Austin – and eight other defendants were charged in six indictments with conspiracy to violate federal narcotic law.

b
```
(T/txt-sc1/--
  (T/t_ta-da+_t (T/w the_wd three_wd)
   (Ta/dash+ _pda
    (T/t_co_t (T/w Miles_wd J_wd Cooperman_wd) _pco
     (T/t_co_t (T/w Sheldon_wd Teller_wd) _pco
        (T/w and_wd Richard_wd Austin_wd)))
    _pda)
   (T/w and_wd eight_wd other_wd defendant_wd be_wd charge_wd
     in_wd six_wd indictment_wd with_wd conspiracy_wd to_wd
     violate_wd federal_wd narcotic_wd law_wd)))
```

The rules of the text grammar divide into three groups: those introducing text-sentences, those defining text adjunct introduction and those defining text adjuncts. An example of each type of rule is given below:

a) T/txt-sc1 : TxtS → (Tu[+sc])* Tu[-sc] (+pex|+pqu)

b) Ta/dash- : Tu[-sc] → T[-sc, -cl, -da] Ta[+da, da-]

c) T/t_ta-da-_t : Ta[+da, da-] → +pda Tu[-sc, -da]

These rules are phrase structure rewrite rule schemata employing standard operators, such as Kleene star, optionality and disjunction, preceded by a mnemonic name. Non-terminal categories are text sentences, units or adjuncts which carry features mostly representing the punctuation marks which occur as daughters in the rules (e.g. +sc represents presence of a semi-colon marker), whilst terminal punctuation categories are represented as +pxx (e.g. +pda represents a dash). For example, a) states that a text sentence can contain zero or more text units (with a semi-colon at their right boundary) followed by a text unit without the semi-colon, optionally followed by a question mark or exclamation mark. Features are unified between categories at parse time and serve to enforce constraints on presence/absence of marks and also enforce some scope constraints between them. For example, b) states that a text unit not containing a semi-colon can consist of a text unit or adjunct not containing dashes, colons or semi-colons followed by a text adjunct introduced by a dash. This type of 'unbalanced' text adjunct can only be expanded out by c) which states that it consists of a single opening dash followed by a text unit which doesn't itself contain dashes or semi-colons. The effect of the features on the first daughter of b) is to enforce dash adjuncts to have lower precedence and narrower scope than colons or semi-colons and to block interpretations of multiple dashes as sequences of 'unbalanced' adjuncts. Thus these rules will license analyses such as (5a) but not (5b).

(5) a

```
         T/txt-sc1/+-
            .    .
          .        .
        .             .
   T/txt-cl1/-  T/t_ta-da-_t
      . .          .   .
    .   .       .       .
  .     .     .          .
 T/w   psc   T/w     Ta/dash-
  .           .         . .
  .           .      .    .
  .           .     .      .
words       words  pda   T/w
                           .
                           .
                           .
                        words
```

b

```
         T/txt-sc1
            .    .
          .        .
        .             .
   T/txt-cl1/-  T/t_ta-da-_t
      . .  .          .
    .   .   .          .
  .     .    .          .
 T/w   psc  T/w     Ta/dash-
  .          .       .  .
  .          .      .    .
  .          .     .      .
words      words  pda   T/w
                          .
                          .
                          .
                        words
```

Nunberg (1990) invokes rules of (point) absorption which delete punctuation marks (inserted according to a simple context-free text grammar) when adjacent to other 'stronger' punctuation marks. For instance, he treats all dash interpolated text adjuncts as underlyingly balanced, but allows a rule of point absorption to convert (6a) into (6b).

(6) a *Max fell – John had kicked him –.

b Max fell – John had kicked him.

The various rules of absorption introduce procedurality into the grammatical framework and require the positing of underlying forms which are not attested in text. For this reason, I make no use of such rules but rather capture their effects through propagation of featural constraints in parse trees. For instance, (6a) is blocked by including distinct rules for the introduction of balanced and unbalanced text adjuncts and only licensing the latter text sentence finally.

2.1 Coverage and Performance

The text grammar has been tested on the Susanne corpus, a 100,000 word parsed subset of the Brown corpus (Sampson, 1994), and covers 99.8% of the text sentences extracted. In fact, the true parse failure rate is lower because approximately half of the 91 failures result from failures of segmentation or tokenisation of the input. For example, a significant number of 'sentences' extracted began with an initial dash, probably indicating list or itemised formatting in the original text. However, the grammar does not license text sentences beginning with a dash. The genuine counter examples found were mainly highly genre specific forms of punctuation, such as citation punctuation from academic papers and varian of itemising punctuation, such as a colon followed by a sequence of dash introduced items. It would be straightforward to extend the grammar to cover such cases, but this has not been undertaken since they occur rarely.

The number of analyses varies from one (71%) to the thousands (.1%). Just over 50% of Susanne sentences contain some punctuation, so this means that around 20% of the singleton parses are punctuated. The major source of ambiguity in the analysis of punctuation concerns the function of commas and their relative scope as a result of a decision to distinguish delimiters and separators (Nunberg 1990:36) and to represent potential scopings via binary branching arrangements of delimited text units (as in the textual adjunct in (6b)). These decisions mean that a text sentence containing eight commas (and no other punctuation) will have 3170 analyses. An alternate iterative version of this grammar (where recursion was replaced with Kleene operators) substantially reduced such ambiguity, but still resulted in hundreds of analyses for some examples containing multiple commas, as a consequence of ambiguity over the boundaries of comma-separated textual units.

It is difficult to assess the structural accuracy of the grammar; that is, the frequency with which one of the parses returned corresponds to the correct scoping of punctuation marks and correct discrimination of textual units from adjuncts. In the absence of a prior manual analysis of the corpus, this can only been done by manually evaluating some of the parses produced. Given the high degree of ambiguity of some examples this process is potentially very labour intensive. However, some idea of the accuracy of the grammar can be gleaned from the experiment reported below. It is certainly the case that in some instances the grammar is not accurate; for example, (7a) requires the analysis, indicated schematically in (7b), in which the dash adjunct modifies the phrase beginning with *but* and this whole unit modifies that beginning *exaggerating*, but the closest analysis returned by the text grammar has this unit scoping over all the previous material, as in (7c).

(7) a Truman Capote is still revelling in southern Gothicism, exaggerating the old southern legend into something beautiful and grotesque, but as unreal as – or even more unreal than – yesterday.

b TC... Gothicism, (exaggerating... grotesque, (but... as – or... than – yesterday))

c ((TC... Gothicism, exaggerating... grotesque,) (but... as – or... than – yesterday))

Whenever such examples have been noticed they have been used to refine the slightly-modified analysis incorporated into the integrated grammar described in §5.

2.2 Probabilistic Analysis of Punctuation

A probabilistic LR parser based on the (recursive) text grammar has been developed by training the model on approximately 1500 text sentences, using the techniques outlined in §1 and discussed more fully in Briscoe & Carroll (1993). The training data was derived by manually selecting between the alternative analyses offered by the system. The trained model assigned the correct analysis the highest probabilitiy in 81% of the ambiguous portion of the training corpus. This result suggests that a punctuation grammar and probabilistic parsing system of this sort could be utilised to provide an inital structuring of input for later syntactic analysis.

During the training phase a small number of cases of structural inaccuracy of the type illustrated in (7a) were identified. However, it is difficult to form a reliable estimate of how often the text grammar fails in this fashion as such observations require careful consideration of the semantics of each example. Nevertheless, I noticed about 20 such cases during manual training, and this is likely to be a considerable underestimate.

3 POS Sequence Analysis

Jensen (1991) argues that incorporating subcategorisation constraints representing valency / argument structure of (verbal, adjectival and nominal) predicates into lexical syntactic descriptions (as all extant syntactic theories do) is a mistake because the valency of a lexical item varies with sense, text genre and register, linguistic innovation, and so on. Our experiment reported in §1 convinced us that the valency information which we extracted from the LDOCE grammar codes (Carroll & Grover 1989) is, at the very least, inadequate and, if we accept Jensen's arguments, then this approach to valency / argument structure is flawed, in principle. Therefore, I decided to develop a 'shallow' grammar which did not rely on subcategorisation but which did make maximal use of other syntactic constraints, in contrast to more heuristic, partial or statistical approaches such as finite-state filtering (Manning, 1993), mutual information parsing (Magerman & Marcus, 1990), or syntactic sketches (Jensen *et al.* , 1994).

As in Fidditch (Hindle, 1989) or MITFP (de Marcken, 1990), the input to the shallow parser is a determinate sequence of extended part-of-speech (POS) labels of the type which can be practically disambiguated in context using a first-order Markov model (e.g. Church, 1988). However, I utilised the ANLT formalism to develop a feature-based, declarative description of POS label sequences for English. This grammar compiles into an object grammar (DCG format) of around 300 rules and utilises 21 features (maximum number of values, 15). The grammar has been designed to enumerate possible valencies for predicates (verbs, adjectives and nouns) by including separate rules for each pattern of possible complementation in English. However, some other types of structural ambiguity, such as coordination and compounding are deliberately reduced to a single canonical right-branching form to reduce the overall ambiguity rate. The distinction between arguments and adjuncts is expressed, following X-bar theory (e.g. Jackendoff, 1977), by Chomsky-adjunction of adjuncts to maximal projections (XP $\rightarrow$ XP Adjunct) as opposed to 'government' of arguments (i.e. arguments are sisters within X1 projections; X1 $\rightarrow$ X0 Arg1... ArgN). Furthermore, all analyses are rooted (in S) with the effect that the grammar assigns 'spurious' analyses to many sentences. For example, (8a) is tokenised and tagged with POS labels as in (8b), and this forms the input to the shallow parser which produces eighteen analyses, one of which is shown in (8c) (with mnemonic rule names substituted for categories on nodes). In the parses not shown, both PPs are either treated as sisters of the verb in a single flat VP (rule 'V/pp_pp'), corresponding to the analysis in which they are arguments, or as left-branching adjuncts (rule 'VP/vp_pp'), and the relative scope of pre- and post- modifiers and level of attachment of the conjuncts differs.

(8) a This will provide for long-term federal loans for construction of parochial and other private-school facilities.

b `this_DD1 will_VM provide_VV0 for_IF long+term_NNT1 federal_JJ loan_NN2 for_IF construction_NN1 of_IO parochial_JJ and_CC other_JB private+school_NNJ1 facility_NN2`

c

```
(S/np_vp this_DD1
 (VP/vp_pp
  (V/modal_bse/-- will_VM
   (V/pp provide_VV0
    (PP/p1
     (P1/p_np for_IF
      (NP/plu
       (N1/n_n1/- long+term_NNT1
        (N1/ap_n1/- (AP/a1 (A1/a federal_JJ))
                    (N1/n loan_NN2))))))))
  (PP/p1
   (P1/p_n1 for_IF
    (N1/n_of construction_NN1
     (PP/p1
      (P1/p_np of_IO
       (NP/plu
        (N1/ap_n1/-
         (AP/a1 (A1/a (A/cj_int parochial_JJ
                      (A/cj_end and_CC other_JB))))
         (N1/n_n1/- private+school_NNJ1
                    (N1/n facility_NN2))))))))))
```

Although the grammar, enumerates valency possibilities for predicates and checks for global sentential well-formedness, it is best described as 'shallow' as it does not attempt to associate 'displaced' constituents with their canonical position / grammatical role. The number of analyses assigned by this grammar can be in the thousands but for a sentence of about 20 words there are rarely more than 300 analyses. The 25 word sentence in (9) is typical of the Susanne corpus and has 161 analyses.

(9) It urged that the next legislature provide enabling funds and re-set the effective date so that an orderly implementation of the law might be effected.

The obvious way in which to cope with this ambiguity robustly is, as before, to construct a probabilistic LR parser for this grammar and use it to rank ambiguous analyses.

The principle difference between this grammar and a more conventional one, apart from the lack of subcategorisation, is that the grammar incorporates some rules specifically designed to overcome limitations or idiosyncracies of the tagging process. For example, past particles functioning adjectivally, as in (10a), are frequently tagged as past participles (VVN) as in (10b), so the grammar incorporates a rule (violating X-bar theory) which parses past participles in this context.

(10) a The disembodied head

b The_AT disembodied_VVN head_NN1

Similar idiosyncratic rules are incorporated for dealing with gerunds, adjective-noun conversions, idiom sequences, and so forth.

3.1 Coverage and Performance

This grammar has been continuously developed and refined in a corpus-based fashion (e.g. Black, 1994) by testing against sentences from the Susanne corpus. The coverage of the grammar on Susanne is currently around 78% with the parser set to accept only parses with root S nodes, and around 70% on the SEC (which I have treated solely as a test corpus). Many of the failures for shorter text sentences are a consequence of the insistence on root S nodes, since they are in fact (elliptical) noun or prepositional phrases. Many failures on sentences are a consequence of incorporation of complementation constraints for auxiliary verbs into the grammar but the lack of any treatment of unbounded dependencies. Nevertheless, I have chosen to continue with both of these 'deficiencies, since they have the effect of limiting the number of analyses recovered in other cases, and do not affect unduly the type of application we envisage (see §7).

I do not analyse the performance of the POS sequence grammar alone further, as these results are based on parsing text sentences with internal punctuation removed and our principal interest is in the integration of the two grammars to create a robust and accurate parser of POS and punctuation sequences.

4 The Integrated Grammar

It would be possible to construct a loosely-coupled system in which analysis of punctuation and POS sequences was performed in two separate stages. This seems attractive when one considers that analysis of punctuation yields a result in nearly all cases, whilst analysis of POS sequences is far less reliable – any closely-coupled system is liable to inherit the overall performance of the worst component. One way in which we might implement a loosely-coupled system would be by defining the input to POS sequence analysis to be the unlabelled bracketing (or parse forest of unlabelled bracketings) provided by analysis of punctuation and by requiring that POS sequence analysis only yield analyses compatible with these bracketings (see e.g. Pereira & Schabes 1992). There are two reasons why I have not adopted this approach: firstly, the potentially high level of ambiguity in analysis of punctuation, and secondly, the possibility of resolving many of these ambiguities through partial integration of punctuation and syntactic analysis.

4.1 Interaction of POS and Punctuation Constraints

I noted earlier that sentences containing many commas can be highly ambiguous according to our grammar of punctuation, mostly because they can function as syntactic separators in constructions such as coordination, as well as textual delimiters of textual adjuncts. These uses cannot be resolved without access to (at least) the syntactic context of occurrence. The example in (11) contains eleven commas, a colon and an opening and closing bracket.

(11) Those three other great activities of the Persian[1], the bath[2], the teahouse[3], and the zur khaneh (the latter a kind of club in which a leader and a group of men in an octagonal pit move through a rite of calisthenics[4], dance[5], chant poetry[6], and music)[7], do not take place in buildings to which entrance tickets are sold[8], but some of them occupy splendid examples of Persian domestic architecture: long[9], domed[10], chalk-white rooms with dais of turquoise tiling[11], their end walls cut through to the orchard and the sky by open arches.

The bracketed textual adjunct and the colon adjunct provide some restriction on the relative scope of the comma delimited units. Nevertheless, this example has thousands of analyses in terms of punctuation alone. When we examine the syntactic context of these commas though, it is easy to see that [2-6] and [8] function as coordination separators with scope defined by the coordination construction, and [9-10] separate adjectival premodifiers. By contrast [1] until [7] and [11] until the period delimit textual adjuncts containing elaborative material. Whilst recognising these latter commas as delimiters requires, ultimately, recognition of the elaborative nature of the enclosed material, it is comparatively easy to determinately recognise the others as separators by integrating their recognition with that of the syntactic constructions which utilise commas in this way.

4.2 Modularity

The integration of punctuation analysis and POS sequence analysis is straightforward and remains modular, in that the text grammar is 'folded into' the POS sequence grammar, by treating text categories and syntactic categories as overlapping and dealing with the properties of each using disjoint sets of features. The text grammar rules are represented as left- or right- branching rules of 'Chomsky-adjunction' to lexical or phrasal constituents. For example, the simplified rule for combining NP appositional or parenthetical text adjuncts is N2[+ta] $\rightarrow$ H2 Ta[+bal] which says that a NP containing a textual adjunct consists of a head NP followed by a textual adjunct with balanced delimiters (dashes, brackets or commas). Rules of this form ensure that the syntactic and textual analysis are mutually 'transparent' and orthogonal so, for example, if rules of semantic interpretation are associated with syntactic rules these could in a simplistic approach be set up to simply ignore adjuncts. Moreover, the rules of adjunction for the most part attach text adjuncts to the constituents over which they semantically scope, so it would be possible, in principle, to develop a semantics of such adjunction rules (where they do not it would be straightforward to add the appropriate rules to the text grammar – they have been excluded for the moment since without such a semantic component they introduce unnecessary ambiguity). In practice, the rules of text adjunct introduction are constrained to adjoin adjuncts to specific syntactic categories primarily to control the degree of ambiguity inherent in the approach. The correct overall generalisation is probably that text adjuncts can adjoin to any (major) syntactic category. If this approach were taken the modularity of the grammars would be enhanced at the expense of extra ambiguity. The approach taken here is in many ways similar to that taken independently by Jones (1994). However, one significant difference is that I have chosen to tokenise punctuation marks separately from words to which they were cliticised and to treat them using independent text gramamtical rules as far as possible. Jones (1994) opts to treat punctuation marks as clitics on words which introduce additional featural information into standard syntactic rules. Thus, his resulting grammar is thoroughly integrated and it would be harder to extract an independently appliable text grammar from it.

In addition to the core text grammatical rules which carry over essentially unchanged from the stand-alone punctuation grammar, 44 syntactic rules now include (often optional) comma markers (rules of pre- and post- posing, coordination, and so forth – corresponding to the purely 'syntactic' uses of punctuation). Here, I have moved the division between balanced comma delimiters indicating text adjuncts and individual comma separators somewhat to include some cases which Nunberg (1990) analyses as underlying delimiters with sentence -intitial or -final commas 'deleted' by absorption rules. Since the function of these commas seems to be to indicate syntactic boundaries and/or syntactic scop, I think this modification is defensible. (In fact, the decision could be reversed at the expense of some added complexity in the grammar.) The coverage of the integrated grammar appears to remain as good as the coverage of the POS sequence grammar applied to depunctuated input (that is, around 78% on Susanne). However, there are examples which can be analysed successfully using the integrated approach which are uncomputable or fail without punctuation constraints, and there are examples which cannot be parsed using the integrated grammar which do parse with the POS sequence grammar. (The full grammar is given as a listing in the first appendix.)

4.3 Illustrative Analyses

In this section, I include the best analysis provided by the grammar for many of the examples given in Nunberg (1990). This will give a feeling for the type of analyses produced and also allows us to briefly comment on some salient issues in the analysis of punctuation. For each example, I show in a) the original sentence, in b) the tokenised and tagged sentence which forms the input to the parser and the number of parses found in square brackets, and in c) the best parse found.

All the analyses of (12) correctly predict that the dash introduced adjunct modifies the text unit which begins after the semi-colon. The differences between the analyses concern the interaction of three ambiguities: whether the bracketed adjunct modifies the preceding noun or NP, whether its internal constituent is S or VP (essentially an ambiguity over the tag RT), and whether *else* should be analysed as a postmodifier of *what* or a sentential adverbial (a limitation of the grammar deriving from the lack of a complete treatment of unbounded dependencies). Only the first ambiguity is text grammatical and it is clear that the bracketed adjunct is a VP which modifies a NP argument. The analysis is not completely correct (even in its own terms) since the question mark is incorrectly given widest scope over the full text sentence in all parses. This limitation of the text grammar has not been corrected since it does not affect the intended application (see §7) particularly. Nevertheless, it does lead to parse failures in examples such as *He walked toward the hostile crowd anyway – why one asks in retrospect? – and almost immediately they began to stone him.* which do occur occasionally in corpora.

(12) a The poor (now called the disadvantaged) cried;
Caesar wept – what else could he do?

b The_AT poor_NN1 (_(now_RT called_VVD the_AT disadvantaged_NN1)_) cried_VVD ;_; Caesar_NP1 wept_VVD -_- what_DDQ else_RA could_VM he_PPHS1 do_VV0 ?_? [8]

c
```
(T/txt-sc1/++-
 (T/txt-cl1/-
  (S/np_vp
   (T/lmta_np (NP/det_n The_AT (N1/n poor_NN1))
    (Taph/brack (_(
     (Tph/vp
      (VP/npadv1_vp now_RT
       (V/np called_VVD (NP/det_n the_AT (N1/n disadvantaged_NN1)))))
     )_)))
   (V/0 cried_VVD))
  ;_;)
 (T/leta_s (S/np_vp (NP/n1_name/- (N1/n Caesar_NP1)) (V/0 wept_VVD))
  (Tacl/dash- -_-
   (S/whnp (NP/pro_ap what_DDQ (AP/a1 (A1/a else_RA)))
    (S/sai/- could_VM (S/np_vp he_PPHS1 (V/0 do_VV0))))))
 ?_?)
```

In (12c) as before I display a labelled bracketing with (mnemonic) rule names labelling nodes. These can be related to the actual rules in the second appendix. Both text adjuncts are left-adjoined to the units they modify by the rules, T/lmta_np and T/leta_s, which stand for 'left-adjoin a text adjunct in the middle of a text sentence to NP' and 'left-adjoin a text adjunct at the end of a text sentence to a clause'. These two text adjunct introduction rules are transparent to the syntactic rules in the sense that they do not modify the nature of the syntctic head though they do record featurally that the head consists of a text unit containing a text adjunct and they also impose restrictions on the nature of this adjunct. For example T/lmta_np slightly abbreviated is N2[+ta] → H2 Ta[-cl, +bal] and imposes the requirements that the text adjunct be a balanced version of a dash, bracket or comma introduced adjunct. In what follows I will point out salient features of the text grammatical aspects of each analysis but ignore most syntactic ambiguities. The reader interested in working through the analyses is referred initially to the second appendix, but will probably need to run the grammar on examples using the ANLT parser as well.

Example (13) illustrates the analysis of matrix level semi-colons. There are no text grammatical ambiguities. (The 8 analyses result from a systematic ambiguity in which the language names are interpreted as vocatives following an intransitive verb.)

(13) a John took French; Susan took Spanish; Annette took Russian.

b John_NP1 took_VVD French_NP1 ;_; Susan_NP1 took_VVD Spanish_NP1 ;_; Annette_NP1 took_VVD Russian_NP1 [8]

c
```
(T/txt-sc1/+--
  (T/txt-cl1/-
   (S/np_vp (NP/n1_name/- (N1/n John_NP1))
    (V/np took_VVD (NP/n1_name/- (N1/n French_NP1))))
   ;_;)
  (T/txt-cl1/-
   (S/np_vp (NP/n1_name/- (N1/n Susan_NP1))
    (V/np took_VVD (NP/n1_name/- (N1/n Spanish_NP1))))
   ;_;)
  (S/np_vp (NP/n1_name/- (N1/n Annette_NP1))
   (V/np took_VVD (NP/n1_name/- (N1/n Russian_NP1)))))
```

Question marks and exclamation marks are not identical to text sentence final periods in that they can occur text sentence internally, as (14) shows.

(14) a A quorum reached?; it is not likely.

b A_AT quorum_NP1 reached_VVD ?_? ;_; it_PPH1 is_VBZ not_XX likely_JJ [1]

c
```
(T/txt-sc1/+--
  (T/txt-cl1/+
   (S/np_vp (NP/det_n A_AT (N1/n quorum_NP1)) (V/0 reached_VVD))
   ?_? ;_;)
  (S/np_vp it_PPH1 (V/be_ap/+ is_VBZ not_XX (AP/a1 (A1/a likely_JJ)))))
```

Example (15) shows that colon expansions can be followed by a semi-colon delimited clause has wide scope over the colon introduced textual adjunct and the text unit to which it is left-adjoined. The other text grammatical analysis with narrow scope for the semi-colon would be appropriate for *He told the press his reason: he would not renegotiate his contract; but he wanted to play out the* *season.*

(15) a He told the press his reason: he would not renegotiate his contract; but he did not explain to the team owners.

b He_PPHS1 told_VVD the_AT press_NN1 his_APP$ rea-
son_NN1 :_: he_PPHS1 would_VM not_XX renegotiate_VV0 his_APP$ contract_NN1 ;_; but_CC he_PPHS1 did_VDD not_XX explain_VV0 to_II the_AT team_NN1 owners_NN2 [2]

c
```
(T/txt-sc1/---
 (T/leta_s
  (S/np_vp He_PPHS1
   (V/np_np told_VVD
    (NP/det_n the_AT (N1/n press_NN1))
    (NP/det_n his_APP$ (N1/n reason_NN1))))
  (Tacl/colon2 :_:
   (T/txt-sc2/--
    (T/txt-cl1/-
     (S/np_vp he_PPHS1
      (V/modal_bse/- would_VM
       (VP/not_vp not_XX
        (V/np renegotiate_VV0 (NP/det_n his_APP$ (N1/n contract_NN1))))))
     ;_;)
    (S/cj_end but_CC
     (S/np_vp he_PPHS1
      (V/do_bse/- did_VDD
       (VP/not_vp not_XX
        (V/pp explain_VV0
         (PP/p1
          (P1/p_np to_II
           (NP/det_n the_AT (N1/n1_nm team_NN1 (N1/n owners_NN2)))))))))))))
```

Example (16) only has one text grammatical analysis with low attachment and narrrow scope for the final comma-delimited clause. Thus, the text grammar predicts the ungrammaticality of **He told the press his reason: he would not renegotiate his contract, but he did not explain to the team owners.*

(16) a He told them his reason: he would not renegotiate his contract, but would stay.

b He_PPHS1 told_VVD them_PPHO2 his_APP$ reason_NN1 :_: he_PPHS1 would_VM not_XX renegotiate_VV0 his_APP$ contract_NN1 ,_, but_CC would_VM stay_VV0 [1]

c
```
(T/txt-sc1/---
 (T/leta_s
  (S/np_vp He_PPHS1
   (V/np_np told_VVD them_PPHO2 (NP/det_n his_APP$ (N1/n reason_NN1))))
  (Tacl/colon1 :_:
   (S/np_vp he_PPHS1
    (VP/cj_int/+
     (V/modal_bse/- would_VM
      (VP/not_vp not_XX
       (V/np renegotiate_VV0 (NP/det_n his_APP$ (N1/n contract_NN1)))))
     ,_, (VP/cj_end but_CC (V/modal_bse/- would_VM (V/0 stay_VV0))))))))
```

(17) illustrates that in some cases combination of bracketed and dash interpolated adjuncts is possible. Nunberg (1990) argues that this is only possible where the first modifies a lexical noun and the second the full NP. The current text grammar also licenses a reading where both modify either noun or NP.

(17) a We saw the movie (which had been banned in Boston) – Jane wanted to go – but we were unimpressed.

b We_PPIS2 saw_VVD the_AT movie_NN1 (_(which_DDQ had_VHD been_VBN banned_VVN in_II Boston_NP1)_) -_- Jane_NP1 wanted_VVD to_TO go_VV0 -_- but_CC we_PPIS2 were_VBDR unimpressed_VVN [12]

c
```
(T/txt-sc1/---
 (S/cj_int/-
  (S/np_vp We_PPIS2
   (V/np saw_VVD
    (T/lmta_np
     (NP/det_n the_AT
      (N1/n
       (T/lmta_n movie_NN1
        (Tacl/brack (_(
         (S/s_npvoc1/-
          (S/np_vp which_DDQ
           (VP/vp_pp
            (V/have_ppart/- had_VHD (V/be_ppart/- been_VBN (V/0 banned_VVN)))
            (PP/p1 (P1/p in_II))))
          (NP/n1_name/- (N1/n Boston_NP1)))
         )_)))))
     (Tacl/dash+ -_-
      (S/np_vp (NP/n1_name/- (N1/n Jane_NP1))
       (V/inf wanted_VVD (V/to_bse to_TO (V/0 go_VV0))))
     -_-))))
  (S/cj_end but_CC
   (S/np_vp we_PPIS2 (V/be_ppart/- were_VBDR (V/0 unimpressed_VVN))))))
```

The text grammar predicts (incorrectly?) that (18) only has an analysis in which Bill fell because Sue pushed him and Mary (then) tickled him whilst (she was) laughing. (To capture the (correct?) analysis in which the semi-colon is internal to a balanced text adjunct and Bill is doing the laughing requires addition of a balanced dash adjunct rule analogous to Tacl/colon2.)

(18) a Bill fell – Sue pushed him; Mary tickled him – laughing.

b Bill_NP1 fell_VVD -_- Sue_NP1 pushed_VVD him_PPHO1 ;_; Mary_NP1 tickled_VVD him_PPHO1 -_- laughing_VVG [1]

c
```
(T/txt-sc1/+--
 (T/txt-cl1/-
  (T/leta_s (S/np_vp (NP/n1_name/- (N1/n Bill_NP1)) (V/0 fell_VVD))
   (Tacl/dash- -_-
    (S/np_vp (NP/n1_name/- (N1/n Sue_NP1)) (V/np pushed_VVD him_PPHO1))))
  ;_;)
 (T/leta_s
  (S/np_vp (NP/n1_name/- (N1/n Mary_NP1)) (V/np tickled_VVD him_PPHO1))
  (Taph/dash- -_- (Tph/vp (V/0 laughing_VVG)))))
```

(19) illustrates that the text grammar correctly predicts the possibility of dashed adjuncts occurring internally to a bracketed adjunct. The sole ambiguity concerns the in this case semantically indistinguishable attachment of the bracketed adjunct as a noun modifier.

(19) a Jones (an employee – a part-time employee – of the firm) was present.

b Jones_NP1 (_(an_AT employee_NN1 -_- a_AT part-time_JJ employee_NN1 -_- of_II the_AT firm_NN1)_) was_VBDZ present_JJ [2]

c
```
(T/txt-sc1/---
 (S/np_vp
  (T/lmta_np (NP/n1_name/- (N1/n Jones_NP1))
   (Taph/brack (_(
    (Tph/np
     (NP/det_n an_AT
      (N1/n1_pp1
       (N1/n
        (T/lmta_n employee_NN1
         (Taph/dash+ -_-
          (Tph/np
           (NP/det_n a_AT
            (N1/ap_n1/- (AP/a1 (A1/a part-time_JJ)) (N1/n employee_NN1))))
          -_-)))
       (PP/p1 (P1/p_np of_II (NP/det_n the_AT (N1/n firm_NN1)))))))
    )_)))
  (V/be_ap/- was_VBDZ (AP/a1 (A1/a present_JJ)))))
```

(20) illustrates that adverbial textual adjunct modifiers are currently attached leftwards to NP (or noun) when they occur sentence-medially. This is semantically incorrect; however, inclusion of the obvious text adjunct introduction rules which right attach the adjunct to VP would be straightforward – the

problem is that they create additional unnecessary ambiguity from the point of view of the application (see §7).

(20) a The key, obviously, has been lost.

b The_AT key_NN1 ,_, obviously_RR ,_, has_VHZ been_VBN lost_VVN [2]

c
```
(T/txt-sc1/---
  (S/np_vp
   (T/lmta_np (NP/det_n The_AT (N1/n key_NN1))
    (Taph/comma+/- ,_, (Tph/a2 (AP/a1 (A1/a obviously_RR))) ,_,))
   (V/have_ppart/- has_VHZ (V/be_ppart/- been_VBN (V/0 lost_VVN)))))
```

In (21) *apparently* is correctly predicted to scope over the second text clause. In *John left, apparently; Mary stayed*, this prediction is reversed, correctly.

(21) a John left; apparently, Mary stayed.

b John_NP1 left_VVD ;_; apparently_RR ,_, Mary_NP1 stayed_VVD [1]

c
```
(T/txt-sc1/+--
  (T/txt-cl1/-
   (S/np_vp (NP/n1_name/- (N1/n John_NP1)) (V/0 left_VVD)) ;_;)
  (S/adv_s/+ (AP/a1 (A1/a apparently_RR)) ,_,
   (S/np_vp (NP/n1_name/- (N1/n Mary_NP1)) (V/0 stayed_VVD))))
```

Separator commas of coordination (and preposed or postposed, sentence -final or -initial phrases) as in (22), are dealt with by adding optional comma terminal daughters to existing syntactic rules. Only balanced comma delimiters, vocative phrases and a few other specific cases are currently treated text grammatically.

(22) a The woods are big, dark and deep.

b The_AT woods_NN2 are_VBR big_JJ ,_, dark_JJ and_CC deep_JJ [2]

c
```
(T/txt-sc1/---
  (S/np_vp (NP/det_n The_AT (N1/n woods_NN2))
   (V/be_ap/- are_VBR
    (AP/cj_int/+ (AP/a1 (A1/a big_JJ)) ,_,
     (AP/cj_int/- (AP/a1 (A1/a dark_JJ))
      (AP/cj_end and_CC (AP/a1 (A1/a deep_JJ))))))))
```

Adjectival premodifiers can optionally be separated by commas, so syntactic rule, N1/ap_n1, has been modified to allow an optional comma separator daughter. Occasionally this rule leads to misanalyses in *adjective, noun* sequences where the N1 is not itself premodified. (This could be prevented if a feature analogous to MOD for nominal premodifiers was introduced.)

(23) a It has a powerful, twenty-four valve engine.

b It_PPH1 has_VHZ a_AT powerful_JJ ,_, twenty-four_MC valve_NN1 engine_NN1 [3]

c
```
(T/txt-sc1/---
  (S/np_vp It_PPH1
   (V/have_np has_VHZ
    (NP/det_n a_AT
     (N1/ap_n1/+ (AP/a1 (A1/a powerful_JJ)) ,_,
      (N1/num2_n1 (NP/num (N1/n twenty+four_MC))
       (N1/n1_nm valve_NN1 (N1/n engine_NN1))))))))
```

Premodifiers of adjectival and nominal phrases, as well as clauses, can be bracketed adjuncts (usually with an optional and disjunctive interpretation). The text grammar allows this possibility with three right-adjoining rules for balanced bracketed adjuncts. The rules of right-adjunction of bracketed text adjuncts correctly predict that a textual unit consisting entirely of such a bracketed text adjunct will be ungrammatical because there will be no accessible material for it to right-adjoin to.

(24) a He presented a (pseudo) formal analysis.

b He_PPHS1 presented_VVD a_AT (_(pseudo_RG)_) formal_JJ analysis_NN1 [1]

c
```
(T/txt-sc1/---
  (S/np_vp He_PPHS1
   (V/np presented_VVD
    (NP/det_n a_AT
     (N1/ap_n1/-
      (AP/a1
       (T/rmta_a1 (Taph/brack (_( (Tph/a2 (AP/a1 (A1/a pseudo_RG))) )_))
        (A1/a formal_JJ)))
      (N1/n analysis_NN1))))))
```

(25) receives analyses in which the bracketed adjunct is left adjoined to NP or noun. It is possible that an analysis attaching this adjunct as a clausal modifier would be semantically better, but this is not included for reasons of spurious ambiguity mentioned above. (Currently, the adjunct reading of the final PP appears to be blocked for reasons that need investigating but which are connected to the intervening TA.)

(25) a They include free gifts (as they put it) with every purchase.

b They_PPHS2 include_VV0 free_JJ gifts_NN2 (_(as_CSA they_PPHS2 put_VV0 it_PPH1)_) with_IW every_AT1 purchase_NN1 [6]

c
```
(T/txt-sc1/---
 (S/np_vp They_PPHS2
  (V/np_pp include_VV0
   (T/lmta_np
    (NP/plu1 (N1/ap_n1/- (AP/a1 (A1/a free_JJ)) (N1/n gifts_NN2)))
    (Taph/brack (_(
     (Tph/pp
      (PP/p1 (P1/p_s as_CSA (S/np_vp they_PPHS2 (V/np put_VV0 it_PPH1)))))
     )_)))
   (PP/p1 (P1/p_np with_IW (NP/det_n every_AT1 (N1/n purchase_NN1)))))))
```

The examples in (26a,b,c,d) are text ungrammatical and none are assigned any analysis by the grammar. (26a) involves recursion on colon adjuncts which is blocked as the colon adjunct rule prevents the adjunct from itself containing a colon adjunct. (26b,c) are blocked because the text grammar will not license category-initial text units (here delimited by a comma) which are bracketed adjuncts, see (24). (26d) is blocked because multiple left-adjoined clausal modifying medial text adjuncts are blocked. However, the closely related (26e) does parse as multiple left-adjoined medial adjuncts on NP are allowed. However, the correct generalisation is not clear in this instance (see example (17) above).

(26) a *They serve cajun dishes: redfish, gumbo, and unusually: catfish sushi.

b *(Not surprisingly), she left.

c *He told us the news: (not surprisingly), she left.

d *She left – who could blame her – (during the chainsaw scene) and went home.

e *She left him – who could blame her – (during the chainsaw scene) and went home.

In addition, there are many other possible violations of point 'absorption' rules which are correctly predicted to be impossible. Finally, there are several text grammatical examples which do not parse, incorrectly. (27a) does not, because the dashed adjunct cannot attach leftward to AdvP/VP or rightward to S. To correct this, T/rmta_s should probably be relaxed to allow rightward attachment of dashed adjuncts to S. (27b) does not, because the text adjunct introduction rules have not been extended to adjectival/adverbial categories. I have not seen an example of this type in corpus material (yet). (27c) is an example of 'promotion' of comma to semi-colon as a marker of conjuncts in the presence of other commas within the conjunct. I have seen one example of this convention in the Susanne corpus but so far have not implemented the 'rule' (it would involve some effort, though I do not think it would be impossible). (27d) involves, according to Nunberg (1990:60), absorption of the adjunct-final dash by the colon. I find this example somewhat marginal and have not catered for unbalanced dashed adjuncts sentence-medially (though it would be straightforward to add extra rules, at the cost of extra ambiguity).

(27) a Holding the bat firmly – her father (a former leaguer) had given it to her – she awaited the pitch

b The key, obviously – or at least, apparently – has been lost.

c The speakers were Ed; Rachel, a linguist; and Shirley.

d We heard the guns – devastating in their fury: they had been placed on the hill.

There are other examples in Nunberg (1990); however, I think these examples illustrate the main phenomena covered by the text grammar (which does not attempt to include quotation, italicisation, text sentence boundary markers, or word-internal punctuation).

5 Parsing the Susanne and SEC Corpora

The integrated punctuation and POS label sequence grammar has been used to analyse (virtually) all the sentences containing eight or more words drawn from the Susanne and the quite distinct SEC Corpus, which consists of 50,000 words of various registers of transcribed spoken British English, punctuated by the corpus compilers (Taylor & Knowles 1988). Both corpora were retagged with determinate punctuation and POS labels using the Acquilex HMM tagger (Elworthy, 1993, 1994) trained on text tagged with a modified version of CLAWS-II labels.[4]

In these experiments I report a number of measures designed to examine the efficiency and coverage of the grammar and to evaluate the usefulness of punctuation to increase coverage and constrain analyses. Firstly, I report the parse failure rate for the grammar; that is, the number and proportion of sentences for which no analysis was found. This measure is just the inverse of what I refer to as the 'coverage' of the grammar. However, note that coverage here is a weak measure since discovery of one or more global analyses for a sentence does not entail that the correct analysis is contained in this set. Secondly, I report the number and proportion of sentences for which certain numbers of analyses were found. This measure gives an intuitive feel for the degree of ambiguity inherent in the grammar (for a given corpus). In addition, I report the number and proportion of examples which failed as a result of our sentence tokenising or segmentation software and those which timed-out after consuming more than 45secs CPU time on a SUN 4/80 Sparcstation 10 with 64mbytes

[4]This retagging, in fact, increased the error rates reported below by approximately 5% over the use of the manually corrected labels in Susanne3. However, using automatic tagging represents a fairer test of the robustness of the overall technique.

Corpus Name:	Susanne		SEC	
Tokenising Errors:	478	8.24%	251	11.11%
Parse Fails:	1309	22.57%	590	26.11%
1–9 Parses:	1179	20.33%	459	20.31%
10–99 Parses:	1182	20.38%	402	17.79%
100–999 Parses:	787	13.57%	267	11.81%
1K–9.9K Parses:	221	3.81%	82	3.63&
Time-outs:	650	11.21%	205	9.07%
No. of Sentences:	5806		2256	
Average Sent. Lgth:	23.25		25	
Av. S. Lgth - PFs:	21.62		21.44	
Av. S. Lgth - TOs:	31.57		33.16	
Av. Parse Base	1.94		1.72	

Table 1: Results for Susanne and SEC sentences

of physical memory running the ANLT system in Allegro Common Lisp 4.2. These figures are helpful during grammar development to ensure that coverage is increasing but the ambiguity rate is not. However, a more succinct though less intuitive measure of ambiguity rate for a given corpus is Black's average parse base (APB, Black *et al*, 1994:13), which is defined as the geometric mean of the number of parses per word averaged over the entire corpus. Given a sentence n tokens long, this number raised to the nth power gives the average number of analyses assigned by the grammar to a sentence of that length in the corpus. [5] I also report the number of sentences analysed, their average length in tokens and the average length of failures and time-outs.

Table 1 gives these measures for all of the eight or more token sentences from Susanne and SEC. Given that the gramamr was developed with reference to Susanne data, the performance on SEC is surprisingly good. The two corpora differ considerably since the former is drawn from American written text whilst the latter represents British transcribed spoken material and both contain material drawn from widely disparate genres / registers. The 3.4% increase in the parse failure rate suggests that the grammar is relatively unaffected by all these linguistic differences. Despite the slightly greater average sentence length of SEC sentences tested, the figures all suggest a lower level of average ambiguity for this corpus – this might be a consequence of more thorough post-hoc punctuation by the compilers of the corpus, or of properties of spoken material, or both. The APBs of 1.94 and 1.72 contrast with that reported by Black (1994:156) of 1.35 for the IBM grammar of computer manuals applied to a corpus containing sentences between 1 and 17 words in length. The greater increase in APBs for our grammar is probably due to the increased sentence length, the greater heterogeneity of the corpora, and the treatment of the argument / adjunct distinction. Black achieves a coverage of around 95% on this

[5] The APB is computed efficiently by exp(sum(log(no.-of-analyses/length-of-sentence))) over all sentences.

Corpus Name:	SUS Punctuated		SUS Unpunctuated	
Tokenising Errors:	283	7.61%	304	8.17%
Parse Fails:	1020	27.42%	1031	27.72%
1–9 Parses:	511	13.74%	422	11.34%
10–99 Parses:	644	17.31%	615	16.53%
100–999 Parses:	523	14.06%	520	13.98%
1K–9.9K Parses:	212	5.70%	265	7.12&
Time-outs:	532	14.30%	568	15.27%
No. of Sentences:	3725		3725	
Average Sent. Lgth:	27.03		24.98	
Av. S. Lgth - PFs:	23.68		21.22	
Av. S. Lgth - TOs:	33.69		31.55	
Av. Parse Base	2.70		3.92	

Table 2: Results for (Un)Punctuated Susanne

more homogeneous data, as opposed to our coverage rate of 74–78%.

The same measures can be used to evaluate the degree to which punctuation is contributing useful information. In Table 2 I show the results parsing the punctuated portion of Susanne containing sentences with more than eight words and the results parsing the same set of sentences with the same grammar with punctuation removed. Although there is only a slight increase in the parse failure rate for the depunctuated sentences, there is a clear increase in complexity as the APB increases by about 30% from 2.7 to 3.92. Most of the additional complexity is located in the longer sentences, the graph in the second appendix plotting number of parses / word against number of tokens with that degree of ambiguity makes the additional ambiguity of the depunctuated examples clear.

The results reported above do not give any indication of the success of the grammar at retrieving the correct analysis amongst the set returned for a sentence. The difficulty of calculating such a result is that for each sentence of the corpora we need a definition of correct parse and then we need a technique for comparing the output parses against this standard. One popular technique which has been developed by the Grammar Evaluation Interest Group (GEIG, see eg. Harrison *et al*, 1991) involves comparing unlabelled bracketings derived from corpus treebanks with those derived from parses for the same sentences by computing precision, the ratio of matched brackets over all brackets found by the parser, and recall, the ratio of matched brackets over all brackets in the treebank; and 'crossing' brackets, the number of times a bracketed sequence output by the parser overlaps with one from the treebank but neither is properly contained in the other.

Black (1994:7) uses the crossing brackets measure to define a notion of structural consistency, where the structural consistency rate for the grammar is defined as the proportion of sentences for which at least one analysis contains no crossing brackets, and reports a rate of around 95% for his grammar of computer manuals on several tests. The problem with the GEIG scheme and Black's simpler notion of structural consistency is that both are still weak measures of performance (designed to avoid problems of parser/treebank representational compatibility) which lead to unintuitive numbers whose significance still depends heavily on details of the relationship between the representations compared. The crossing bracket measure alone is particularly weak because in general the fewer the brackets inserted by the grammar/parser, the better it will do, whilst certain types of error such as incorrect level of attachment of constituents will not result in crossing brackets. For these reasons, I have not computed such statistics but prefer an analysis based on a simple measure of tree similarity between the 'correct' parse, as defined by the grammar writer, and that returned by the parser. Furthermore, neither structural consistency nor the GEIG scheme in basic form distinguish amongst the set of analyses returned by the parser. There is little point in deriving such statistics for unranked parses independent of some means of ranking them, since in practice most applications will only be able to consider the highest or topmost ranked parses.

5.1 Integrated Probabilistic Analysis

In order to derive a probabilistic LR parser for the POS and punctuation sequence grammar, we utilised a tree similarity measure (very similar to the GEIG scheme) to construct a training corpus from a set of unlabelled bracketings extracted from the Susanne treebank. For each unlabelled bracketed sentence, the parser found the analysis which scored highest on this measure and from this set of analyses for approximately 3500 sentences, LALR transition probabilities were estimated. This use of unlabelled bracketings is, of course, less objectionable (though still not ideal), since it implements an automatic but semi-supervised training regime for the probabilistic LR parser. Training in this fashion is not ideal as some analyses cannot be distinguished in terms of an unlabelled bracketing. However, this approach is more adequate that attempting to estimate probabilities from unlabelled input (e.g. Pereira & Schabes, 1992; Briscoe, 1994). We are at present extending this approach so that probabilities derived from unlabelled bracketings can be re-estimated using adaptive learning, rather than directly estimated from the resulting corpus of (sometimes incorrect) parses.

The trained parser was applied to a further set of 200 test sentences from Susanne all containing a semi-colon and possibly other punctuation marks. The preliminary results of this experiment are given in Table 3. The sentences chosen were some of the most complex and long to be found in Susanne (average length over 35 tokens) and this is reflected in the high parse failure rate. For each sentence for which a parse could be found, I calculated the number of rules applied in the most probable analysis and the number of these rules which were incorrect applied at that point in the derivation. This measure took no account of the capabilities of the grammar or of any POS mistagging which occurred in successfuly-parsed examples, but was based solely on linguistic judgements of what represented the correct textual and syntactic analysis. By this (high)

Tokenising Errors:	4	2.0%
Parse Fails:	86	43.0%
Time-outs:	9	4.5%
0 Error:	9	4.5%
1 Error:	15	7.5%
2 Errors:	19	9.5%
3–4 Errors:	31	15.5%
5–11 Errors:	27	13.5%
No. Applied:	5037	
No. Incorrect:	343	
Perc. Correct:	93.19%	
Average Parse Base:	3.30	
Highest No. of Pses:	3425	
Average Sent. Lgth:	35.61	

Table 3: Results for Susanne 200 Test

standard of evaluation, the most probable analysis was completely correct for only 4.5% of the test sentences (or roughly 9% of those successfully-parsed). This criterion though is considerably stronger and more reliable than the GEIG scheme or even 'perfect match' tests against treebank material, since the latter often underdetermines a complete analysis and often contains errors. When we examine the closeness of the most probable analyses to the correct analysis, the picture looks considerably more promising – only 343 rule applcations out of 5037 applied to the successfully-parsed examples were incorrect. Thus over 93% of rule applications were correct and most of the highest-ranked analyses were close to the correct analysis. Many of the errors (around 50%) involve incorrect attachment of phrases; this is exacerbated by the maintenance of the argument / adjunct distinction in the grammar despite the lack of lexical information to resolve it in the input. However, previous experiments have shown similar effects with more constrained grammars (Briscoe & Carroll, 1993) pointing to the inadequacy of the probabilistic LR model, which is based entirely on the frequency of application of syntactic rules in (LR) parse contexts, for resolving ambiguities involving lexical sense distinctions and collocations.

In order to determine the extent to which punctuation information contributed to the performance summarised in Table 3, I reparsed the 100 successfully analysed examples with punctuation removed using the identical grammar and trained parser. These results are summarised in Table 4. The first point to note is that the depunctuated sentences are considerably more ambiguous than their punctuated equivalents, resulting in an APB three times higher than that for the successfully analysed punctuated sentences; the highest number of analyses returned in the depunctuated case was over 8k (as compared to just over 3k in the punctuated case) and 26 examples which parsed successfully when punctuated were timed-out when depunctuated. this is despite the fact that depunctuated examples are an average of 1.8 tokens shorter than their punctu-

Time-outs:	26	26%
0 Error:	0	0%
1 Error:	9	9%
2 Errors:	10	10%
3–4 Errors:	34	34%
5–9 Errors:	21	21%
No. Applied:	2962	
No. Incorrect:	267	
Perc. Correct:	91%	
Average Parse Base:	4.73	
Highest No. of Pses:	8265	
Average Sent. Lgth:	33.82	

Table 4: Results for 100 Parsed Sentences Unpunctuated

ated equivalents. The percentage of successful rule applications was surprisingly close to that for the punctuated case with only a 2% decline in performance. However, there is still strong evidence that the performance of the parser is degrading significantly; there were no cases of completely correct parses in the depunctuated test, and examination of the analyses around the location of the deleted punctuation marks revealed systematic errors. For instance, the majority of semi-colon deletions resulted in incorrect 'low' and 'local' attachment of the following clause to the leftmost constituent licensed to take a sentential complement. In a few cases, this was avoided because the text sentence had a sufficiently unambiguous coordinate structure. In many cases, of comma deletions, however, the system was able to correctly recover the intended structure as these were optional separators, but in cases where commas functioned as delimiters the correct structure was not recovered. Finally, in some cases the syntactic analysis of the sentence improved over the punctuated equivalent, it is difficult to determine why this might occur, but in some cases it is clear that this is a result of inadequacies in the current text grammar, and elsewhere it may be a function of independence assumptions built into the probabilistic model (so the statistical model only captures very 'local' dependencies, some of which may be made 'non-local' by intervening text grammar rules).

This preliminary experiment suggests that the combined performance of the integrated grammar and trained probabilistic LR parser is promising – though not good enough to be practical as a stand-alone syntactic analyser for subsequent semantic interpretation. We end this section by giving a few examples of successfully parsed examples from the 200 test examples along with their depunctuated analyses and some brief comments.

5.1.1 Illustrative Examples

In the following examples, a) gives the original sentence and b) its tokenised, lemmatised and tagged counterpart whilst c) and d) give the highest ranked analysis of the punctuated and depunctuated versions, respectively. The figures in square brackets represent the total number of analyses found followed by the fraction of correct rule applications in this highest ranked analysis.

Example (28) is typical of many involving semi-colons; the punctuated version is considerably less ambiguous (2 as opposed to 28 analyses). In the punctuated version, the second clause is analysed as a separate text grammatical clause. In the depunctuated version, the second clause is treated as a sentential complement of *mile* and *off* is forced to become an adverbial modifier of the second clause.

(28) a They were about a mile off; under me the ground quivered slightly.

b they_PPHS2 be_VBDR about_II a_AT1 mile_NNU1 off_RP ;_; under_II I_PPIO1 the_AT ground_NN1 quiver_VVD slightly_RR

c
```
T/txt-sc1/+--
 (T/txt-cl1/-
  (S/np_vp they_PPHS2
   (VP/vp_adv
    (V/be_pp/-- be_VBDR
     (PP/p1 (P1/p_np about_II (NP/det_n a_AT1 (N1/n mile_NNU1)))))
    (AP/a1 (A1/a off_RP))))
  ;_;)
 (S/pp_s/- (PP/p1 (P1/p_np under_II I_PPIO1))
  (S/np_vp (NP/det_n the_AT (N1/n ground_NN1))
   (VP/vp_adv (V/0 quiver_VVD) (AP/a1 (A1/a slightly_RR)))))) [2, 20/21]
```

d
```
(T/txt-sc1/---
 (S/np_vp they_PPHS2
  (V/be_pp/-- be_VBDR
   (PP/p1
    (P1/p_np about_II
     (NP/det_n a_AT1
      (N1/n_s4 mile_NNU1
       (S/adv_s/- (AP/a1 (A1/a off_RP))
        (S/pp_s/- (PP/p1 (P1/p_np under_II I_PPIO1))
         (S/np_vp (NP/det_n the_AT (N1/n ground_NN1))
          (VP/vp_adv (V/0 quiver_VVD)
           (AP/a1 (A1/a slightly_RR))))))))))))) [28, 18/20]
```

Example (29) illustrates a case where the analyses recovered are essentially identical (both containing the same error of analysis of *on a par with...*) despite the absence of the semi-colon, because of the unambiguous coordinate structure.

(29) a Its growth continues steadily on a par with past growth; and no limitation is in evidence.

b its_APP$ growth_NN1 continue_VVZ steadily_RR on_II a_AT1 par_NN1 with_IW past_JJ growth_NN1 ;_; and_CC no_AT limitation_NN1 be_VBZ in_II evidence_NN1

c
```
(T/txt-sc1/+--
 (T/txt-cl1/-
  (S/np_vp (NP/det_n its_APP$ (N1/n growth_NN1))
   (VP/vp_pp
    (VP/vp_pp (VP/vp_adv (V/0 continue_VVZ) (AP/a1 (A1/a steadily_RR)))
     (PP/p1 (P1/p_np on_II (NP/det_n a_AT1 (N1/n par_NN1)))))
    (PP/p1
     (P1/p_n1 with_IW
      (N1/ap_n1/- (AP/a1 (A1/a past_JJ)) (N1/n growth_NN1))))))
  ;_;)
 (S/cj_end and_CC
  (S/np_vp (NP/det_n no_AT (N1/n limitation_NN1))
   (V/be_pp/-- be_VBZ (PP/p1 (P1/p_n1 in_II (N1/n evidence_NN1)))))))
[7, 28/29]
```

d
```
(T/txt-sc1/---
 (S/cj_int/-
  (S/np_vp (NP/det_n its_APP$ (N1/n growth_NN1))
   (VP/vp_pp
    (VP/vp_pp (VP/vp_adv (V/0 continue_VVZ) (AP/a1 (A1/a steadily_RR)))
     (PP/p1 (P1/p_np on_II (NP/det_n a_AT1 (N1/n par_NN1)))))
    (PP/p1
     (P1/p_n1 with_IW
      (N1/ap_n1/- (AP/a1 (A1/a past_JJ)) (N1/n growth_NN1))))))
  (S/cj_end and_CC
   (S/np_vp (NP/det_n no_AT (N1/n limitation_NN1))
    (V/be_pp/-- be_VBZ (PP/p1 (P1/p_n1 in_II (N1/n evidence_NN1))))))))
[12, 28/29]
```

(30) is a clear example where an inadequacy of the text grammar leads the system astray in the punctuated case, as it incorrectly licenses the schematic analysis:

(sentence ((s-adverbial, sentence); sentence) pp)

so the system does slightly better in the depunctuated case, correctly attaching the final PP to *think*.

(30) a I do not know yet, it is crazy; I have to think about it.

b I_PPIS1 do_VD0 not_XX know_VV0 yet_RR ,_, it_PPH1 be_VBZ crazy_JJ ;_; I_PPIS1 have_VH0 to_TO think_VV0 about_II it_PPH1

c
```
(T/txt-sc1/---
 (S/np_vp I_PPIS1
  (VP/vp_pp
   (V/do_bse/- do_VD0
    (VP/not_vp not_XX
     (V/tcl/- know_VV0
      (T/txt-sc2/--
       (T/txt-cl1/-
        (S/adv_s/+ (AP/a1 (A1/a yet_RR)) ,_,
         (S/np_vp it_PPH1 (V/be_ap/- be_VBZ (AP/a1 (A1/a crazy_JJ)))))
        ;_;)
       (S/np_vp I_PPIS1
        (V/have_inf have_VH0 (V/to_bse to_TO (V/0 think_VV0))))))))
   (PP/p1 (P1/p_np about_II it_PPH1))))) [7, 16/21]
```

d
```
(T/txt-sc1/---
 (S/np_vp I_PPIS1
  (V/do_bse/- do_VD0
   (VP/not_vp not_XX
    (V/s know_VV0
     (S/np_vp (NP/ap2_np (A1/a yet_RR) it_PPH1)
      (V/be_ap/- be_VBZ
       (AP/a1
        (A1/a_s crazy_JJ
         (S/np_vp I_PPIS1
          (VP/vp_pp
           (V/have_inf have_VH0 (V/to_bse to_TO (V/0 think_VV0)))
           (PP/p1 (P1/p_np about_II it_PPH1)))))))))))))) [10, 15/18]
```

In (31) and Figure 1 we illustrate performance on one of the longer examples for the punctuated cases – the depunctuated example timed-out and could not be parsed. It is clear that these longer examples often become highly ambiguous without punctuation, providing some informal support for Jones' (1994) conjecture that punctuation marks provide anchors of relatively low ambiguity and that overall ambiguity is some function of the length of constituents intervening between (at least some) punctuation marks. In this analysis, the separator commas marking coordination at *active...* are misanalysed as delimiters.

(31) a In the overall process, it is difficult to assign a pure role to each constituent of a build detergent formulation; and, indeed, there is no more reason to separate the interrelated roles of the active, builder, anti-redeposition agent, etc. than there is to assign individual action to each of the numerous isomers making up a given commercial organic active.

b in_II the_AT overall_JB process_NN1 ,_, it_PPH1 be_VBZ difficult_JJto_T assign_VV0 a_AT1 pure_JJ role_NN1 to_II each_DD1 constituent_NN1 of_IO a_AT1 build_VVN detergent_JJ formulation_NN1 ;_; and_CC ,_, indeed_RGA ,_, there_EX be_VBZ no_AT more_DAR reason_NN1 to_TO separate_VV0 the_AT interrelate_VVN role_NN2 of_IO the_AT active_NN1 ,_, builder_NN1 ,_, anti-redeposition_JB agent_NN1 ,_, etc._RA than_CSN there_EX be_VBZ to_TO assign_VV0 individual_JJ action_NN2 to_II each_DD1 of_IO the_AT numerous_J isomer_NN2 make_VVG up_RP a_AT1 give_VVN commercial_JJ organic_JJ active_NN1

6 Applications

There are many potential applications for robust shallow parsing. Some of these are explored in Jensen *et al.* (1994) where applications of a similarly-defined 'syntactic sketch' to style checking, spelling correction, procedural recovery of predicate-argument structure, machine translation, and so forth are discussed. In the context of domain-independent robust syntactic analysis, we are exploring the use of shallow parsing for the extraction of complementation patterns of predicates and its use as a first stage in robust, full syntactic parsing. A second potential application of the grammar is as an intermediate stage in the 'cascaded' parsing of sentences in which loosely-coupled parsing modules take unlabelled bracketings from the previous stage as input to training and/or parsing at the next stage. In what I follows I briefly outline one potentially promising application.

6.1 Extracting Complementation Patterns

There has been considerable recent interest in using statistical corpus analysis techniques to automatically acquire subcategorisation information on the basis of the syntactic and lexical context in which predicates occur. However, the systems which have been developed either rely on simple, robust recognition of context using *local* syntactic and lexical constraints (e.g. Brent 1991, 1993; Manning 1993), or on treebanks constructed manually or with conventional parsers (Grishman & Sterling 1992; Poznanski & Sanfilippo 1993). The disadvantage of the former is that very large corpora of relevant examples will

```
(T/txt-sc1/+--
 (T/txt-cl1/-
  (S/pp_s/+
   (PP/p1 (P1/p_np in_II (NP/det_n the_AT
      (N1/ap_n1/- (AP/a1 (A1/a overall_JB)) (N1/n process_NN1)))))
   ,_,
   (S/np_vp it_PPH1
    (V/be_ap/- be_VBZ (AP/a1 (A1/a_inf difficult_JJ
       (V/to_bse to_TO (V/np_pp assign_VV0
         (NP/det_n a_AT1 (N1/ap_n1/- (AP/a1 (A1/a pure_JJ)) (N1/n role_NN1)))
         (PP/p1 (P1/p_np to_II
           (NP/det_n each_DD1 (N1/n_of constituent_NN1
             (PP/p1 (P1/p_np of_IO (NP/det_n a_AT1 (N1/vppt_n1 build_VVN
                 (N1/ap_n1/- (AP/a1 (A1/a detergent_JJ))
                  (N1/n formulation_NN1))))))))))))))))))
  ;_;)
 (S/cj_end (T/lmta_conj and_CC
   (Taph/comma+/- ,_, (Tph/a2 (AP/a1 (A1/a indeed_RGA))) ,_,))
  (S/np_vp there_EX (V/be_np/--- be_VBZ
    (NP/det_n no_AT (N1/ap_n1/- (AP/a1 (A1/a more_DAR))
      (N1/n_inf reason_NN1
       (V/to_bse to_TO (V/np_np separate_VV0
         (NP/det_n the_AT
          (N1/n1_pp2 (N1/vppt_n1 interrelate_VVN
            (N1/n_of role_NN2 (PP/p1 (P1/p_np of_IO
               (T/lmta_np (NP/det_n the_AT (N1/n active_NN1))
                (Taph/comma+/- ,_,
                 (Tph/n1+ (N1/co_n1 (N1/n builder_NN1) ,_,)
                  (N1/ap_n1/- (AP/a1 (A1/a anti-redeposition_JB))
                   (N1/n agent_NN1))) ,_,))))))
           (PP/adv_p1 (AP/a1 (A1/a etc._RA))
            (P1/p_s than_CSN (S/np_vp there_EX (V/be_inf/- be_VBZ
               (V/to_bse to_TO (V/np_ing/- assign_VV0
                 (NP/plu1 (N1/n1_pp1
                   (N1/ap_n1/- (AP/a1 (A1/a individual_JJ)) (N1/n action_NN2))
                   (PP/p1 (P1/p_np to_II (NP/pro_pp each_DD1
                      (PP/p1 (P1/p_np of_IO (NP/det_n the_AT
                         (N1/ap_n1/- (AP/a1 (A1/a numerous_JJ))
                          (N1/n isomer_NN2)))))))))))
                 (V/0 make_VVG))))))))
         (NP/ap2_np (A1/a up_RP) (NP/det_n a_AT1 (N1/vppt_n1 give_VVN
            (N1/ap_n1/- (AP/a1 (A1/a commercial_JJ))
             (N1/ap_n1/- (AP/a1 (A1/a organic_JJ))
              (N1/n active_NN1)))))))))))))) [93/99]
```

Figure 1: Best Analysis for (31b)

be required to achieve reliable results given the degree of noise in the information extracted, whilst the latter requires intensive manual analysis or the prior existence of robust, full parsers (which require accurate subcategorisation information in the first place).

We have implemented a simple heuristic system that extracts syntactic and lexical information out of sub-analyses from a parse forest, together with information about the contribution of the sub-analysis to the set of analyses found. For example, for (5) above, if we ask for information for *provide* we obtain the following patterns:

```
(18
((|this_DD1| "V/0" |provide_VV0|) . 8)
((|this_DD1| "V/pp_pp" |provide_VV0||for_IF||for_IF|).2)
((|this_DD1| "V/pp" |provide_VV0| |for_IF|) . 8))
```

In 8 of the 18 analyses found, *provide* was either analysed as intransitive or taking a PP[for] complement, and in 2, as taking two such complements. We can weight this information by the relative probability of the different analyses, but in this case a more heuristic approach to acquiring subcategorisation information would suffice in which the acquisition program simply knew that intransitivity is the 'null' hypothesis and verbs do not take complements introduced by identical prepositions. Such patterns can be surprisingly determinate; for instance, (6) has 161 analyses, however, the syntactic context around *provide* restricts the range of possible complementation patterns contributing to a global analysis to:

```
(161 ((|legislature_NNJ1| "V/np" |provide_VV0| |fund_NN2|) . 161))
```

This pattern also illustrates the potential usefulness of extracting lexical heads from surrounding putative arguments to building up semantic selectional information. Our hypothesis is that extracting such patterns for many citations of a single verb should allow us to construct an accurate account of its complementational (and perhaps selectional) properties using simple statistical and heuristic filtering of the patterns extracted. This approach will yield better data that approaches based solely on local constraints, but be more automatic and practical than those based on full syntactic parses. Furthermore, in the case where a global analysis cannot be found by our system, we are still able to extract patterns from sub-analyses; for instance, the following fragment from an unparseable sentence resulted in the pattern below:

```
... the senate is expected to approve ...
(0 (PASSIVE |senate_NNJ1| "V/inf" |expect_VVN| *))
```

It is immediately clear that this pattern contains less information, as we have no idea whether it would have contributed to any global syntactic analysis of the sentence involved. However, such patterns can still be exploited by the acquisition system, albeit with lower confidence. Finally, the acquisition process can be made more efficient and focussed by restricting the extraction of patterns to the *n*-best analyses returned by the probabilistic LR parser, where n is determined experimentally to ensure that the correct analysis has an appropriately high chance of being found in each set considered.

The resulting patternsets for sentences containing a target predicate extracted using this system are input to a patternset evaluator which attempts to create lexical entries encoding valency, alternations, control of predicative complements and semantic selectional preferences. Further details of this system and some preliminary results evaluating its performance are given in Briscoe & Carroll (1994).

7 Conclusions and Further Work

Our preliminary results suggest that the LR probabilistic model can be applied with some success to grammars of English words, of punctuation, and of English POS sequences. The exploitation of global syntactic constraints on POS and punctuation sequences leads to a robust 'shallow' parser capable of analysing naturally-occurring and complex examples from a wide variety of sources. However, increasing coverage towards 100% and increasing the percentage of most probable analyses which are correct remain hard and unsolved problems. The utility of the type of grammar described here is limited by its overly liberal approach to valency / subcategorisation. Greater accuracy in analyses will probably only be achieved by a more direct application of valency constraints acquired using the techniques outlined in the last section, either through more refined and lexical probabilistic modelling, and/or through direct incorporation of accurate valency information into the lexicon/grammar.

The text grammar requires further development in order to reach optimal performance and coverage. Nevertheless, these experiments suggest that a systematic approach to text grammar and punctuation can considerably facilitate robust parsing. Further work should address more directly the correctness of some hard constraints incorporated into the text grammar; for example, Sampson (1993) argues that colon expansions can be followed by wide scoping comma-introduced material, citing several examples like the following from the LOB corpus: *The answer: simplicity, good sense and an instinct for dietetic values, and what more could one ask?*. Such constraints can be modelled as preferences, if appropriate, via probabilities. However, this option is only attractive, since it introduces further ambiguity, if such cases turn out to be sufficiently frequent.

From a more theoretical perspective, the declarative approach to text grammar developed here could be incorporated into a more complete syntactic grammar of English incorporating a rule-to-rule compositional semantic component (such as the full ANLT grammar of English, Grover *et al.*, 1993). Full integration would involve the development of semantic rules for text unit and text adjunct combination. I think this should be feasible within the rule-to-rule framework, since some text adjuncts reduce to syntactic modifiers or conjunts of the text unit over which they scope, whilst others stand in a discourse relationship to each other. These latter would involve the incorporation of discourse relations (and some truth-conditional interpretation of these) into the compositional semantic framework (see e.g. Asher, 1993). For instance, a semi-colon typically cues a discourse embedding relation between two text units, such as explanation or elaboration.

Acknowledgements

Geoff Nunberg provided encouragement and much advice on the analysis of punctuation. Greg Grefenstette undertook the original corpus tokenisation and segmentation. John Carroll helped with subsequent corpus tokenisation, segmentation and automatic tagging and also implemented and maintained the ANLT software and probabilistic extensions. Bernie Jones and Kiku Ribas made helpful comments on an earlier draft. I am responsible for all the mistakes.

References

Asher, N. 1993. *Reference to Abstract Objects in Discourse.* Kluwer Academic, Dordrecht.

Black, E. (ed.) 1994. *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach.* Rodopi, Amsterdam.

Brent, M. 1991. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 209–214. Berkeley, CA.

Brent, M. 1993. From grammar to lexicon: unsupervised learning of lexical syntax. *Computational Linguistics* 19.3: 243–262.

Briscoe, E. 1994. Prospects for practical parsing of unrestricted text: robust statistical parsing techniques. In Oostdijk, N & de Haan, P. eds. *Corpus-based Research into Language.* Rodopi, Amsterdam: 97–120.

Briscoe, E. and Carroll, J. 1993. Generalised probabilistic LR parsing for unification-based grammars. *Computational Linguistics* 19.1: 25–60.

Briscoe, E. and Carroll, J. 1994. *Argument Structure Extraction from Corpora.* RXRC, Grenoble, ms..

Briscoe, E., Grover, C., Boguraev, B. and Carroll, J. 1987. A formalism and environment for the development of a large grammar of English. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 703–708. Milan, Italy.

Carroll, J. 1993. *Practical unification-based parsing of natural language.* Cambridge University, Computer Laboratory, TR-314.

Carroll, J. and Briscoe, E. 1992. Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, 33–38. Cambridge, MA.

Carroll, J. and Grover, C. 1989. The derivation of a large computational lexicon for English from LDOCE. In Boguraev, B. and Briscoe, E. eds. *Computational Lexicography for Natural Language Processing.* Longman, London: 117–134.

Church, K 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conf. on Applied Natural Language Processing*, 136–143. Austin, Texas.

Elworthy, D. 1993. *Part-of-Speech Tagging and Phrasal Tagging.* Acquilex-II Working Paper 10, Cambridge University Computer Laboratory†.

Elworthy, D. 1994. Does Baum-Welch re-estimation help taggers?. In *Proceedings of the 4th Conf. Applied NLP.* Stuttgart, Germany.

Grishman, R. & Sterling, J. 1992. Acquisition of selectional patterns. In *Proceedings of the Coling-92*, 658–664. Nantes, France.

Grover, C., Carroll, J. and Briscoe, E. 1993. *The Alvey Natural Language Tools Grammar (4th Release).* Cambridge University Computer Laboratory, TR-284.

Harrison, P., Abney, S., Black, E., Flickenger, D., Gdaniec, C., Grishman, R., Hindle, D., Ingria, B., Marcus, M., Santorini, B. and Strzalkowski, T. 1991. Evaluating syntax performance of parser/grammars of English. In *Proceedings of the Proc. of the Workshop on Evaluating Natural Language Processing Systems.* ACL.

Hindle, D. 1989. Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 118–25. Vancouver, Canada.

Hindle, D., and Rooth, M. 1991. Structural ambiguity and lexical relations. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 229–236. Berkeley, CA.

Jackendoff, R 1977. *X-bar Syntax.* MIT Press; Cambridge, MA..

Jensen, K. Kluwer, Dordrecht. A broad-coverage natural language analysis system. In *Current Issues in Parsing Technology.* Tomita, M. (ed.). 1991

Jensen, K., Heidorn, G. & Richardson, S. (eds.) 1994. *Natural Language Processing: The PLNLP Approach.* Kluwer, Dordrecht.

Jones, B 1994. *Can punctuation help parsing?.* Acquilex-II Working Paper , Cambridge University Computer Laboratory†.

Jones, B 1994b. Can punctuation help parsing?. In *Proceedings of the Coling94.* Kyoto, Japan.

Magerman, D. & M. Marcus 1990. *Parsing a Natural Language Using Mutual Information Statistics.* Pennsylvania University, CIS Dept., Ms..

Manning, C. 1993. Automatic acquisition of a large subcategorisation dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 235–242. Columbus, Ohio.

de Marcken, C. 1990. Parsing the LOB corpus. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 243–251. New York.

Nunberg, G. 1990. *The linguistics of punctuation.* CSLI Lecture Notes 18, Stanford, CA.

Pereira, F. and Warren, D. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13.3: 231–278.

Pereira, F., and Schabes, Y. 1992. Inside-outside re-estimation for partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 128–135. .

Poznanski, V. & Sanfilippo, A. 1993. Detecting dependencies between semantic verb subclasses and subcategorization frames in text corpora. In *Proceedings of the SIGLEX Workshop on Acquisition of Lexical Knowledge.* Boguraev, B. & Pustejovsky, J..

Procter, P. (ed.) 1978. *The Longman Dictionary of Contemporary English.* London, UK: Longman.

Taylor, L., Grover, C. and Briscoe, E. 1989. The syntactic regularity of English noun phrases. In *Proceedings of the 4th European Meeting of the Association for Computational Linguistics*, 256–263. UMIST, Manchester.

Sampson, G. 1993. Review of *Nunberg, G. The Linguistics of Punctuation. Linguistics* 99?: 467–475.

Sampson, G. 1994. Susanne: a Doomsday book of English grammar. In Oostdijk, N & de Haan, P. eds. *Corpus-based Research into Language.* Rodopi, Amsterdam: 169–188.

Taylor, L. and Knowles, G. 1988. *Manual of information to accompany the SEC corpus: the machine-readable corpus of spoken English.* University of Lancaster, UK, Ms..

Tomita, M. 1987. An efficient augmented-context-free parsing algorithm. *Computational Linguistics* 13.1: 31–46.

Whittemore, G., Ferrara, K. & Brunner, H. 1990. Empirical study of simple attachment schemes for post-modifier prepositional phrases. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 23–30. New York.

(Acquilex Working Papers can be obtained from cide@cup.cam.ac.uk)

Appendix 1 – The integrated Grammar

```
; Grammar for Tag Sequences (CLAWS2, SEC/Susanne)
;
; Copyright Ted Briscoe / Rank Xerox, June, 1994
; Use only for research purposes

FEATURE N{+, -}
FEATURE V{+, -}
FEATURE BAR{0, 1, 2}
FEATURE H{+, -}
FEATURE MINOR{DET, COMP, DEG, PRT, CONJ, INTERJ, NOT}
FEATURE PLU{+, -}
FEATURE POSS{+, -}
FEATURE NTYPE{PRO, REFL, NAME, NORM, PART, THERE, NUM, DIR, TIT, PRETIT,
   POSTTIT, TEMP, MEAS, POSS, PLACE}
FEATURE ATYPE{ATT, PRD, POST, CAT, NUM, XCOMP, HOW, DG, PLACE}
FEATURE AFORM{ER, EST, NONE}
FEATURE SUBCAT{SFIN, SCOMP, SINF, NP, PP, VPINF, VPING, VPPRT, NONE}
FEATURE PFORM{FOR, FORS, OF, WITH, WITHOUT, WH, THAN, AS, PREP, PREPS}
FEATURE ADJ{+, -}
FEATURE CJTYPE{BEGIN, END}
FEATURE WH{+, -}
FEATURE ADV{+, -}
FEATURE MOD{+, -}
FEATURE AUX{TO, BE, HAVE, DO, MODAL, CAT, -}
FEATURE VFORM{BSE, INF, ING, PPART, PAST, PRES}
FEATURE INV{+, -}
FEATURE FIN{+, -}
FEATURE CONJ{+, -}
FEATURE T{+}
FEATURE PUNCT{comma, semicolon, colon, period, exmark, qumark, quote,
dash, lbrack, rbrack}
FEATURE SCOLON{+,-}
FEATURE COLON{+,-}
FEATURE DASH{+,-}
FEATURE TA{+,-}
FEATURE BAL{+, B, E}
FEATURE BRACK{+,-}
FEATURE COMMA{+,-}
FEATURE TXTCAT{UNIT, ADJUNCT}
FEATURE TXT{CL, PH}

SET VHEAD = {N, V, PLU, VFORM, AUX, FIN, CONJ}
SET NHEAD = {N, V, PLU, NTYPE, CONJ, POSS}
SET PHEAD = {N, V, PFORM, CONJ, ADJ}
```

```
SET AHEAD = {N, V, AFORM, ATYPE, ADV, CONJ, MOD}
SET PUNCTS = {SCOLON, COLON, DASH, COMMA, BRACK}

ALIAS V2 = [V +, N -, BAR 2, TXTCAT UNIT, TXT CL].
ALIAS V1 = [V +, N -, BAR 1].
ALIAS V0 = [V +, N -, BAR 0].
ALIAS N2 = [V -, N +, BAR 2, TXTCAT UNIT, TXT PH].
ALIAS N1 = [V -, N +, BAR 1, TXTCAT UNIT, TXT PH].
ALIAS N0 = [V -, N +, BAR 0].
ALIAS P2 = [V -, N -, BAR 2].
ALIAS P1 = [V -, N -, BAR 1].
ALIAS P0 = [V -, N -, BAR 0].
ALIAS A2 = [V +, N +, BAR 2].
ALIAS A1 = [V +, N +, BAR 1].
ALIAS A0 = [V +, N +, BAR 0].
ALIAS H2 = [H +, BAR 2].
ALIAS H1 = [H +, BAR 1].
ALIAS H0 = [H +, BAR 0].
ALIAS DT = [MINOR DET].
ALIAS CP = [MINOR COMP].
ALIAS DG = [MINOR DEG].
ALIAS PT = [MINOR PRT].
ALIAS CJ = [MINOR CONJ].
ALIAS IJ = [MINOR INTERJ].
ALIAS NG = [MINOR NOT].
ALIAS Tp = [T +].
ALIAS Tu = [TXTCAT UNIT].
ALIAS Tph = [TXTCAT UNIT, TXT PH].
ALIAS Ta = [TXTCAT ADJUNCT].
ALIAS +psc = [PUNCT semicolon].
ALIAS +pco = [PUNCT comma].
ALIAS +pcl = [PUNCT colon].
ALIAS +pqu = [PUNCT qumark].
ALIAS +pex = [PUNCT exmark].
ALIAS +pda = [PUNCT dash].
ALIAS +plb = [PUNCT lbrack].
ALIAS +prb = [PUNCT rbrack].

ALIAS +sc = [SCOLON +].
ALIAS -sc = [SCOLON -].
ALIAS -cl = [COLON -].
ALIAS +cl = [COLON +].
ALIAS -da = [DASH -].
ALIAS +da = [DASH +].
ALIAS -co = [COMMA -].
ALIAS +co = [COMMA +].
ALIAS -br = [BRACK -].
```

```
ALIAS +br = [BRACK +].
ALIAS nopunct = [-sc, -cl, -da, -co].

ALIAS +bal = [BAL +].
ALIAS +end = [BAL E].
ALIAS +beg = [BAL B].
ALIAS +ta = [TA +].
ALIAS -ta = [TA -].

CATEGORY TXT : [TXTCAT] => {TA, BAL, SCOLON, TXT, COLON, DASH, COMMA, BRACK, CONJ}.
CATEGORY CN : [N +, V -] => {PLU, NTYPE, BAR, WH, POSS, CONJ, MOD}.
CATEGORY CNw : [N +, V -, BAR 0] => {TA}.
CATEGORY CV : [N -, V +] => {PLU, BAR, VFORM, AUX, FIN, CONJ}.
CATEGORY CV1 : [N -, V +, BAR 1] => {MOD, SCOLON}.
CATEGORY CV2 : [N -, V +, BAR 2] => {INV, WH}.
CATEGORY CVw : [N -, V +, BAR 0] => {TA}.
CATEGORY CP : [N -, V -] => {BAR, PFORM, ADJ, CONJ, MOD}.
CATEGORY CPb : [N -, V -, BAR 0] => {SUBCAT, TA}.
CATEGORY CPw1 : [N -, V -, BAR 1] => {WH}.
CATEGORY CPw2 : [N -, V -, BAR 2] => {WH, SCOLON}.
CATEGORY CA : [N +, V +] => {BAR, AFORM, ATYPE, ADV, CONJ, MOD}.
CATEGORY CA2 : [N +, V +, BAR 2] => {SCOLON}.
CATEGORY CDET : [MINOR DET] => {PLU, POSS, WH}.
CATEGORY CCOMP : [MINOR COMP] => {TA}.
CATEGORY CDEG : [MINOR DEG] => {}.
CATEGORY CPRT : [MINOR PRT] => {}.
CATEGORY CNOT : [MINOR NOT] => {}.
CATEGORY CINTERJ : [MINOR INTERJ] => {}.
CATEGORY CCJ : [MINOR CONJ] => {CJTYPE, TA}.

; Text grammar

; TOP Tp, N2.
TOP Tp.

; top-level (semi-colon / sentence-final punct rules) + recursion thru
; colon adjunct

PSRULE T/txt-sc1 : Tp --> (V2[+sc, +cl, FIN +])* V2[-sc, FIN +] (+pqu) (+pex).
PSRULE T/txt-sc2 : V2[+sc, -cl] --> (V2[+sc, +cl, FIN +])+ V2[-sc, FIN +] (+pqu) (+pex).
PSRULE T/s_qu_s : Tp --> V2[FIN +, -sc] +pqu (+pco) V2[FIN +, nopunct].
PSRULE T/s_ex_s : Tp --> V2[FIN +, -sc] +pex (+pco) V2[FIN +, nopunct].
PSRULE T/txt-cl1 : V2[+sc, +cl] --> H2[-sc] (+pqu) +psc.
PSRULE T/txt-cl2 : V2[+sc, +cl] --> H2[-sc] +pex +psc.
PSRULE T/txt-cl3 : V2[+sc, +cl] --> H2[-sc] +pqu +pda.

; text adjunct introduction
```

```
; middle, function word left attach

PSRULE T/lmta_conj :  CJ[+ta] --> CJ[-ta] Ta[-cl, +bal].
PSRULE T/lmta_comp :  CP[+ta] -->  CP[-ta] Ta[-cl, +bal].
PSRULE T/lmta_prep :  P0[+ta, SUBCAT @x] -->  H0[-ta, SUBCAT @x] Ta[-cl, +bal].
PSRULE T/lmta_auxv :  V0[+ta, INV @y] -->  H0[-ta, INV @y] Ta[-cl, +bal].

; middle, N left attach

PSRULE T/lmta_np : N2[+ta, WH @x] --> H2[WH @x, MOD +] Ta[-cl, +bal].
PSRULE T/lmta_n1 : N1[+ta, WH @x] --> H1[WH @x, MOD +] Ta[-cl, +bal].
PSRULE T/lmta_n : N0[+ta, WH @x] --> H0[WH @x] Ta[-cl, +bal].

; middle, V left attach

PSRULE T/lmta_v :  V0[+ta] -->  H0[-ta] Ta[-cl, +bal].

; end, left attach

PSRULE T/leta_s : V2[FIN +, +ta, WH @x] --> H2[-cl, -da, WH @x] Ta[+end].

; begin/middle right attach

PSRULE T/rmta_n1 :  N1[+ta, WH @x] --> Ta[+br] H1[WH @x, MOD -].
PSRULE T/rmta_a1 :  A1[+ta, WH @x] --> Ta[+br] H1[WH @x, MOD -].
PSRULE T/rmta_s : V2[FIN +, +ta, WH @x] --> Ta[+br] H2[-cl, -da, WH @x].

; textual adjuncts (same rules for Tph/V2)

PSRULE Taph/colon : Ta[+cl, +end, TXT PH] --> +pcl Tph.
PSRULE Taph/dash+ : Ta[+da, +bal, TXT PH] --> +pda Tph[-da] +pda.
PSRULE Taph/dash- : Ta[+da, +end, TXT PH] --> +pda Tph[-da].
PSRULE Taph/comma+ :
Ta[+co, +bal, -ta, TXT PH] --> +pco Tph[CONJ -] +pco (Ta[+co, +beg, TXT PH]).
PSRULE Taph/comma-b : Ta[+co, +beg, TXT PH] --> Tph[CONJ -] +pco.
PSRULE Taph/brack : Ta[+br, TXT PH] --> +plb Tph[-br] +prb.

PSRULE Tacl/colon1 : Ta[+cl, TXT CL] --> +pcl V2[-cl, -sc].
PSRULE Tacl/colon2 : Ta[+cl, TXT CL] --> +pcl V2[-cl, +sc].
PSRULE Tacl/dash+ : Ta[+da, +bal, TXT CL] --> +pda V2[-da] +pda.
PSRULE Tacl/dash- : Ta[+da, +end, TXT CL] --> +pda V2[-da].
PSRULE Tacl/comma+ : Ta[+co, +bal, TXT CL] --> +pco V2[-sc, -co, CONJ -] +pco.
PSRULE Tacl/comma-e : Ta[+co, +end, TXT CL] --> +pco V2[-sc, -co, CONJ -].
PSRULE Tacl/comma-b : Ta[+co, +beg, TXT CL] --> +pco V2[-sc, -co, CONJ -].
PSRULE Tacl/brack : Ta[+br, TXT CL] --> +plb V2[-sc, -br] +prb.
```

```
; can't do Tph generally because of diff features in diff cats
; 'list' comma coord of Tphs is like semi-colon coord of Tacls

PSRULE Tph/np+ : Tph --> (N2[+sc])+ N2.
PSRULE Tph/n1+ : Tph --> (N1[+sc])+ N1.
PSRULE Tph/pp+ : Tph --> (P2[+sc])+ P2.
PSRULE Tph/a2+ : Tph --> (A2[+sc])+ A2.
PSRULE Tph/vp+ : Tph --> (V1[+sc])+ V1.

PSRULE Tph/np : Tph[CONJ @x] -->  N2[CONJ @x].
PSRULE Tph/n1 : Tph[CONJ @x] -->  N1[CONJ @x].
PSRULE Tph/pp : Tph[CONJ @x] -->  P2[CONJ @x].
PSRULE Tph/a2 : Tph[CONJ @x] -->  A2[CONJ @x].
PSRULE Tph/vp : Tph[CONJ @x] -->  V1[CONJ @x].

PSRULE NP/co_np : N2[+sc] --> H2[-sc, -ta] +pco.
PSRULE N1/co_n1 : N1[+sc] --> H1[-sc, -ta] +pco.
PSRULE PP/co_pp : P2[+sc] --> H2[-sc] +pco.
PSRULE AP/co_pp : A2[+sc] --> H2[-sc] +pco.
PSRULE VP/co_vp : V1[+sc] --> H1[-sc] +pco.

; integrated rules

; dates (2) Aug (2), 1913

PSRULE NP/date : N2[NTYPE MEAS] --> (N2[NTYPE NUM]) H2 (N2[NTYPE NUM]) +pco N2[NTYPE NUM]

; special comma case

PSRULE Tcl/interj_tph : V2[-sc, +co] --> IJ +pco Tph.

; special qu marks

PSRULE T/npq : N2 --> H2[WH +] +pqu.
PSRULE T/npqcj : N2 --> H2[CONJ +, WH +] +pqu.

; complex AP premods can be sep by commas

PSRULE N1/ap_n1 : N1[WH @x] --> A2[ADV -, MOD -] (+pco) H1[WH @x].

; sentence-level modifiers with optional/oblig commas

PSRULE S/pp_s : V2[WH -, INV -] --> P2[WH -, ADJ -, CONJ @c] (+pco) H2[FIN +, -ta].
PSRULE S/vppt_s : V2[WH -, INV -] --> V1[VFORM PPART] (+pco) H2[WH -, FIN +, -ta].
PSRULE S/vping_s : V2[WH -, INV -] --> V1[VFORM ING] (+pco) H2[WH -, FIN +, -ta].
PSRULE S/npadv1_s : V2[WH @y, INV @x] --> N2[NTYPE TEMP, -ta] (+pco)
                                          H2[WH @y, FIN +, -ta, INV @x].
```

```
PSRULE S/npadv2_s : V2[WH @y, INV @x] --> N2[NTYPE MEAS, -ta] (+pco)
                                          H2[WH @y, FIN +, -ta, INV @x].
PSRULE S/npfr_s : V2[WH -, INV -] --> N2[WH +] (+pco) H2[FIN +, -ta].
PSRULE S/np_co_s1 : V2[WH -, INV -] --> N2[NTYPE NORM] +pco H2[FIN +, -ta].
PSRULE S/np_co_s2 : V2[WH -, INV -] --> N2[NTYPE PRO] +pco H2[FIN +, -ta].
PSRULE S/adv_s : V2[WH @x, INV @y] --> A2[ADV +] (+pco) H2[WH @x, INV @y, FIN +, -ta].

PSRULE S/s_pp : V2[WH -, INV -] --> H2[FIN +, -ta] +pco  P2[WH -, ADJ -, CONJ @c].
PSRULE S/s_vppt : V2[WH -, INV -] --> H2[WH -, FIN +, -ta] +pco V1[VFORM PPART].
PSRULE S/s_vping : V2[WH -, INV -] --> H2[WH -, FIN +, -ta] +pco V1[VFORM ING].
PSRULE S/s_npadv1 : V2[WH @x, INV @y] --> H2[WH @x, INV @y, FIN +, -ta] +pco
                                          N2[NTYPE TEMP, -ta].
PSRULE S/s_npadv2 : V2[WH @x, INV @y] --> H2[WH @x, INV @y, FIN +, -ta] +pco
                                          N2[NTYPE MEAS, -ta].
PSRULE S/s_npfr : V2[WH -, INV -] --> H2[FIN +, -ta] (+pco)  N2[WH +].
PSRULE S/s_co_np1 : V2[WH -, INV -] --> H2[FIN +, -ta]  +pco N2[NTYPE NORM].
PSRULE S/s_co_np2 : V2[WH -, INV -] --> H2[FIN +, -ta]  +pco N2[NTYPE PRO].
PSRULE S/s_adv : V2[WH @x, INV @y] --> H2[WH @x, INV @y, FIN +, -ta] +pco A2[ADV +].
PSRULE S/s_a1 : V2[WH @x, INV @y] --> H2[WH @x, INV @y, FIN +, -ta] +pco A1[ADV -].

; vocatives (except NTYPE NORM...)

PSRULE S/s_npvoc1 : V2 --> H2[FIN +] (+pco) N2[NTYPE NAME].
PSRULE S/s_npvoc2 : V2 --> H2[FIN +] +pco N2[NTYPE TIT].
PSRULE S/s_npvoc3 : V2 --> H2[FIN +] +pco N1[NTYPE NORM, PLU -].
PSRULE S/npvoc_s1 : V2 --> N2[NTYPE NAME] +pco H2[FIN +].
PSRULE S/npvoc_s2 : V2 --> N2[NTYPE TIT] +pco H2[FIN +].
PSRULE S/npvoc_s3 : V2 --> N1[NTYPE NORM, PLU -] +pco H2[FIN +].

; tag qus + S, declared Bill.

PSRULE S/s_tag-qu : V2 --> H2[FIN +] +pco V0[INV @x] (NG) N2[NTYPE PRO] +pqu.
PSRULE S/s_vb-name : V2 --> H2[FIN +] +pco V0[AUX -] N2[NTYPE NAME].
PSRULE S/s_vb-pro : V2 --> H2[FIN +] +pco V0[AUX -] N2[NTYPE PRO].

; POS tag sequence grammar below includes commas in coord rules and nothing else

PSRULE S/np_vp : V2[WH -, INV -] --> N2 H1[FIN +].
PSRULE S/n1_vp : V2[WH -, INV -] --> N1[NTYPE NORM, PLU -] H1[FIN +].
PSRULE S/poss_vp : V2[WH -, INV -] --> N2[NTYPE PRO, POSS +] H1[FIN +].
PSRULE S/np_vping : V2[WH -, INV -] --> N2 H1[FIN -, VFORM ING].
PSRULE S/n1_vping : V2[WH -, INV -] --> N1[NTYPE NORM, PLU -] H1[FIN -, VFORM ING].
PSRULE S/s_vp : V2[WH -, INV -] --> CP V2[WH -, FIN +, -ta] H1[FIN +].
PSRULE S/sing_vp : V2[WH -, INV -] --> V2[WH -, FIN -, VFORM ING, CONJ @x] H1[FIN +].
PSRULE S/vping_vp : V2[WH -, INV -] --> V1[FIN -, VFORM ING, CONJ @x] H1[FIN +].
PSRULE S/s_be_s : V2[WH -, INV -] --> V2[WH -, INV -] V0[INV @x, AUX
BE] (NG)
```

```
                                         V2[WH -, INV -].
PSRULE S/sai : V2[INV +, FIN +] --> V0[INV @x] (NG) V2[WH -, INV -, -ta].
PSRULE S/sai-be1 : V2[INV +] --> V0[INV @x, AUX BE] (NG) N2 (NG) N2.
PSRULE S/sai-be2 : V2[INV +] --> V0[INV @x, AUX BE] (NG) N2 (NG) P2.
PSRULE S/sai-be3 : V2[INV +] --> V0[INV @x, AUX BE] (NG) N2 (NG) A2.
PSRULE S/sai-be4 : V2[INV +] --> V0[INV @x, AUX BE] (NG) N2 (NG) PT.
PSRULE S/comp : V2[WH -, INV -] --> CP H2.
PSRULE S/whnp : V2[WH +, INV @x] --> N2[WH +] H2[FIN +, INV @x].
PSRULE S/how : V2[WH +, INV @x] --> A1[ADV +, ATYPE HOW] H2[WH -, FIN +, INV @x].
PSRULE S/whpp : V2[WH +, INV @x] --> P2[WH +] H2[WH -, FIN +, INV @x].
PSRULE S/wh : V2[WH +, INV -] --> N2[WH +] H2[WH -, FIN +].
PSRULE S/wh-aux : V2[WH +, INV -, FIN @y] --> N2[WH +] N2 V0[INV @x, FIN @y].
PSRULE S/whpp-aux : V2[WH +, INV -] --> P2[WH +] N2 V0[INV @x].
PSRULE S/imp : V2[INV +] --> H1[VFORM BSE, FIN +].
PSRULE S/imp_neg : V2[INV +] --> V0[AUX DO] NG H1[VFORM BSE, FIN +].

PSRULE S/cj_beg : V2[WH @x, CONJ -] --> CJ[CJTYPE BEGIN] H2[WH @x, CONJ +].
PSRULE S/cj_int : V2[WH @y, CONJ @x] --> H2[WH @y, CONJ -, -ta] (+pco)
                                         V2[WH @y, CONJ +, -ta].
PSRULE S/cj_end : V2[WH @x, CONJ +] --> CJ[CJTYPE END]  H2[WH @x, CONJ -].

PSRULE V/0_p : V0 --> H0 PT.
PSRULE V/0 : V1 --> H0.
PSRULE V/np : V1 --> H0 N2.
PSRULE V/n1 : V1 --> H0 N1[NTYPE NORM, WH -, POSS -, PLU -].
PSRULE V/n1b : V1 --> H0 N1[NTYPE TIT, WH -, POSS -, PLU -].
PSRULE V/np_p :  V1 --> H0 N2 PT.
PSRULE V/pp : V1 --> H0 P2[WH -, ADJ -, MOD -].
PSRULE V/inf : V1 --> H0 V1[FIN -, VFORM INF].
PSRULE V/ing : V1 --> H0 V1[FIN -, VFORM ING].
PSRULE V/ppart : V1 --> H0 V1[VFORM PPART].
PSRULE V/bse : V1 --> H0 V1[VFORM BSE].
PSRULE V/s : V1 --> H0 V2[nopunct].
PSRULE V/tcl : V1 --> H0 (CP) V2[+sc, -cl].
PSRULE V/ap : V1 --> H0 A2[ADV -].
PSRULE V/np_np : V1 --> H0 N2 N2.
PSRULE V/np_n1 : V1 --> H0 N2 N1[NTYPE NORM, WH -, POSS -, PLU -].
PSRULE V/np_inf : V1 --> H0 N2 (PT) V1[FIN -, VFORM INF].
PSRULE V/np_ing : V1 --> H0 N2 (PT) V1[FIN -, VFORM ING].
PSRULE V/np_pp : V1 --> H0 N2 P2[WH -, ADJ -, MOD -].
PSRULE V/pp_pp : V1 --> H0 P2[WH -, ADJ -, MOD -] P2[WH -, ADJ -, MOD -].
PSRULE V/np_pp_pp : V1 --> H0 N2 P2[WH -, ADJ -, MOD -] P2[WH -, ADJ -, MOD -].
PSRULE V/np_s : V1 --> H0 N2 (PT) V2[WH -, INV -, -sc, -cl].
PSRULE V/np_ap : V1 --> H0 N2 A2[ADV -].
PSRULE V/np_np_s : V1 --> H0 N2 N2[NTYPE MEAS] V2[WH -, INV -, -sc, -cl].
PSRULE VP/adv_vp : V1[MOD +] --> A2[ADV +] H1.
PSRULE VP/vp_adv : V1[MOD +] --> H1 A2[ADV +].
```

```
PSRULE VP/vp_pp : V1[MOD +] --> H1 P2[ADJ -, WH -].
PSRULE VP/vp_npadv1 : V1[MOD +] --> H1 N2[NTYPE TEMP, -ta].
PSRULE VP/vp_npadv2 : V1[MOD +] --> H1 N2[NTYPE MEAS, -ta].
PSRULE VP/npadv1_vp : V1[MOD -] --> N2[NTYPE TEMP, -ta] H1.
PSRULE VP/npadv2_vp : V1[MOD -] --> N2[NTYPE MEAS, -ta] H1.
PSRULE VP/not_vp : V1[MOD @x] --> NG H1[MOD @x].
PSRULE VP/cj_beg : V1[CONJ -, MOD @x] --> CJ[CJTYPE BEGIN] H1[CONJ +, MOD @x].
PSRULE VP/cj_int : V1[CONJ @x, MOD @z] --> H1[CONJ -, VFORM @y, MOD @z] (+pco)
                   V1[CONJ +, VFORM @y, MOD @z].
PSRULE VP/cj_end : V1[CONJ +] --> CJ[CJTYPE END] H1[CONJ -].
PSRULE V/cj_beg : V0[CONJ -, MOD @x] --> CJ[CJTYPE BEGIN] H0[CONJ +, MOD @x].
PSRULE V/cj_int : V0[CONJ @x] --> H0[CONJ -] (+pco) H0[CONJ +].
PSRULE V/cj_end : V0[CONJ +] --> CJ[CJTYPE END] H0[CONJ -].
PSRULE V/be_ppart : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) V1[MOD -, VFORM PPART].
PSRULE V/be_pas : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) V1[MOD -, VFORM PAST].
PSRULE V/be_ing : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) V1[MOD -, VFORM ING, FIN -].
PSRULE V/be_inf : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) V1[MOD -, VFORM INF, FIN -].
PSRULE V/be_pp : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) (NG) P2[ADJ -, WH -].
PSRULE V/be_prt : V1 --> H0[INV @x, AUX BE] (NG) PT.
PSRULE V/be_ap : V1 --> H0[INV @x, AUX BE] (NG) A2.
PSRULE V/be_np : V1 --> H0[INV @x, AUX BE] (A2[ADV +]) (NG) N2 (PT).
PSRULE V/be_n1 : V1 --> H0[INV @x, AUX BE]  (NG) N1[WH -, POSS -, PLU -] (PT).
PSRULE V/be_s : V1 --> H0[INV @x, AUX BE] V2[nopunct].
PSRULE V/be_ta : V1 --> H0[INV @x, AUX BE]  Ta[+cl].
PSRULE V/do_bse : V1 --> H0[INV @x, AUX DO] (A2[ADV +]) V1[MOD -, VFORM BSE].
PSRULE V/do_np : V1 --> H0[INV @x, AUX DO] N2.
PSRULE V/do_ppart : V1 --> H0[INV @x, AUX DO, VFORM PPART].
PSRULE V/do_pas : V1 --> H0[INV @x, AUX DO, VFORM PAST].
PSRULE Vdo_as : V1 --> H0[INV @x, AUX DO] P2[PFORM AS].
PSRULE V/to_bse : V1 --> H0[AUX TO] V1[MOD -, VFORM BSE].
PSRULE V/have_ppart : V1 --> H0[INV @x, AUX HAVE] (A2[ADV +]) V1[MOD -, VFORM PPART].
PSRULE V/have_pas : V1 --> H0[INV @x, AUX HAVE] (A2[ADV +]) V1[MOD -, VFORM PAST].
PSRULE V/have_np_ppt : V1 --> H0[INV @x, AUX HAVE] N2 V1[VFORM PPART].
PSRULE V/have_np_bse : V1 --> H0[INV @x, AUX HAVE] N2 V1[VFORM BSE].
PSRULE V/have_np : V1 --> H0[INV @x, AUX HAVE] N2.
PSRULE V/have_n1 : V1 --> H0[INV @x, AUX HAVE] N1[NTYPE NORM, WH -, POSS -, PLU -].
PSRULE V/have_inf : V1 --> H0[INV @x, AUX HAVE] V1[MOD -, VFORM INF, FIN -].
PSRULE V/modal_bse : V1 --> H0[INV @x, AUX MODAL] (A2[ADV +]) V1[VFORM BSE].
PSRULE V/cat_inf : V1 --> H0[INV @x, AUX CAT] (A2[ADV +]) V1[VFORM INF, FIN -].
PSRULE V/aux_cj_beg : V0[INV @x, CONJ -] --> CJ[CJTYPE BEGIN] H0[INV @x, CONJ +].
PSRULE V/aux_cj_int : V0[INV @x, CONJ @x] --> H0[INV @x, CONJ -] (+pco) H0[INV @x, CONJ +
PSRULE V/aux_cj_end : V0[INV @x, CONJ +] --> CJ[CJTYPE END] H0[INV @x, CONJ -].

PSRULE NP/plu1 : N2[WH @x, POSS -, NTYPE NORM] --> H1[WH @x, PLU +].
PSRULE NP/plu2 : N2[WH @x, POSS -, NTYPE TEMP] --> H1[WH @x, PLU +].
PSRULE NP/plu3 : N2[WH @x, POSS -, NTYPE MEAS] --> H1[WH @x, PLU +].
PSRULE NP/plu4 : N2[WH @x, POSS -, NTYPE PART] --> H1[WH @x, PLU +].
```

```
PSRULE NP/num : N2[WH @x, POSS -, NTYPE NUM] --> H1[WH @x].
PSRULE NP/place : N2[WH @x, POSS -] --> H1[WH @x, NTYPE PLACE].
PSRULE NP/det_n : N2[WH @x, POSS -] --> DT[WH @x] H1[WH -].
PSRULE NP/det_whn : N2[WH +, POSS -] --> DT H1[WH +].
PSRULE NP/det_det_n : N2[WH @x, POSS -] --> DT[WH @x] DT H1[WH -].

PSRULE NP/part_np : N2[POSS -, WH -] -->  N0[NTYPE PART, WH -] H2.
PSRULE NP/part_of : N2[WH -, POSS -] --> DT DT P2[PFORM OF].
PSRULE NP/num_of : N2[POSS -, MOD +] --> H2[NTYPE NUM] P2[PFORM OF].
PSRULE NP/name_of : N2[POSS -, MOD +] --> H2[NTYPE NAME] P2[PFORM OF].
PSRULE NP/pro_pp : N2[WH @x, POSS -, MOD +] --> H2[NTYPE PRO, WH @x] P2[ADJ +].
PSRULE NP/pro_inf : N2[WH @x, POSS -, MOD +] --> H2[NTYPE PRO, WH @x]
                                                 V1[VFORM INF, FIN -].
PSRULE NP/pro_s : N2[WH -, POSS -, MOD +] --> H2[NTYPE PRO, WH -]
                                              V2[nopunct, INV -, FIN +].
PSRULE NP/pro_ap : N2[WH @x, POSS -, MOD +] --> H2[NTYPE PRO, WH @x] A2[ATYPE POST].

PSRULE NP/npt_ap : N2[POSS -, WH -, MOD +] --> H2[NTYPE TEMP] A1[ATYPE POST, MOD -].
PSRULE NP/npl_ap : N2[POSS -, WH -, MOD +] --> H2[NTYPE PLACE] A1[ATYPE POST, MOD -].
PSRULE NP/nms_ap : N2[POSS -, WH -, MOD +] --> H2[NTYPE MEAS] A1[ATYPE POST, MOD -].
PSRULE NP/np_ap : N2[POSS -, WH -, MOD +] --> H2 A1[ATYPE POST, MOD +].
PSRULE NP/ap1_np : N2[POSS -, WH -] --> A1[ADV +, ATYPE DG] H2[MOD -].
PSRULE NP/ap2_np : N2[POSS -, WH -] --> A1[ADV +, ATYPE CAT] H2[MOD -].
PSRULE NP/np_refl : N2[POSS -, MOD +] --> H2[PLU @x] N2[NTYPE REFL, PLU @x].
PSRULE NP/poss_n1 : N2[POSS -, WH -] --> N2[POSS +] H1[POSS -].
PSRULE NP/n2_poss : N2[POSS +, WH -] --> H2[POSS -] N0[NTYPE POSS].
PSRULE NP/adv1 : N2[POSS -, WH -, PLU -] --> H1[NTYPE TEMP, -ta].
PSRULE NP/adv2 : N2[POSS -, WH -, PLU -] --> H1[NTYPE MEAS, -ta].
PSRULE NP/adv3 : N2[POSS -, WH -, PLU -] --> H1[NTYPE DIR, -ta].
PSRULE NP/interj : N2[NTYPE NORM, POSS -, WH -] --> IJ.

; names
PSRULE NP/name_n1 : N2[WH -, POSS -] --> N2[NTYPE NAME, POSS -, MOD -] H1[NTYPE NORM].
PSRULE NP/n1_name : N2[WH -, POSS -] --> (N1[NTYPE NORM, MOD -])
                                         H1[NTYPE NAME, POSS -, MOD -].
PSRULE NP/name_n2 : N2[WH -, POSS -] -->  N2[NTYPE NAME, POSS -, MOD -] H2[NTYPE NORM].
PSRULE NP/n2_name : N2[WH -, POSS -] --> H2[NTYPE NORM] N2[NTYPE NAME, POSS -, MOD -].
PSRULE NP/tit_name : N2[WH -, POSS -] --> N0[NTYPE TIT, POSS -] H2[NTYPE NAME, MOD -].
PSRULE NP/pretit_n1 : N2[WH -, POSS -] --> N0[NTYPE PRETIT, POSS -] H1[NTYPE NAME, MOD -]
PSRULE NP/n1_posttit : N2[WH -, POSS -] -->  H1[NTYPE NAME, MOD -]
                                             N0[NTYPE POSTTIT, POSS -].
PSRULE NP/dir_name : N2[WH -, POSS -] --> N0[NTYPE DIR, POSS -] H2[NTYPE NAME, MOD -].
PSRULE NP/ap_name : N2[WH -, POSS -] --> A2[ADV -, MOD -] H2[NTYPE NAME, MOD -].
PSRULE NP/num_name2 : N2[WH -, POSS -] --> N2[NTYPE NUM, POSS -] H2[NTYPE NAME, MOD -].
PSRULE NP/name2_num : N2[POSS -] --> H2[NTYPE NAME, POSS -, MOD -] N2[NTYPE NUM, POSS -].
PSRULE NP/temp_name2 : N2[WH -, POSS -] --> N2[NTYPE TEMP, POSS -] H2[NTYPE NAME, MOD -].
```

```
; compounds
PSRULE N1/n1_nm : N1[POSS -] --> N0[POSS -, NTYPE NORM] H1[NTYPE NORM, -ta].
PSRULE N1/n1_npl : N1[POSS -, NTYPE NORM] --> N0[POSS -, NTYPE NORM] H1[NTYPE PLACE].
PSRULE N1/n1_dir : N1[POSS -, NTYPE NORM] --> N0[POSS -, NTYPE NORM] H1[NTYPE DIR].
PSRULE N1/n1_nms : N1[POSS -, NTYPE NORM] --> N0[POSS -, NTYPE NORM] H1[NTYPE MEAS].
PSRULE N1/n1_nt : N1[POSS -, NTYPE NORM] --> N0[POSS -, NTYPE NORM] H1[NTYPE TEMP].

PSRULE N1/nt_nt : N1[POSS -] -->  N0[NTYPE TEMP, POSS -] H1[NTYPE TEMP].
PSRULE N1/nms_nms : N1[POSS -] -->  N0[NTYPE MEAS, POSS -] H1[NTYPE MEAS].

PSRULE N1/num2_n1 : N1[POSS -, PLU @x] --> N2[NTYPE NUM, POSS -, PLU @x, MOD -]
                                           H1[NTYPE NORM].
PSRULE N1/num2_nt : N1[POSS -, PLU @x] --> N2[NTYPE NUM, POSS -, PLU @x, MOD -]
                                           H1[NTYPE TEMP].
PSRULE N1/num2_nms : N1[POSS -, PLU @x] --> N2[NTYPE NUM, POSS -, PLU @x, MOD -]
                                            H1[NTYPE MEAS].
PSRULE N1/nms_num2 : N1[POSS -] --> H0[NTYPE MEAS] N2[NTYPE NUM, POSS -, MOD -].
PSRULE N1/nm_num2 : N1[POSS -] --> H0[NTYPE NORM] N2[NTYPE NUM, POSS -, MOD -].

PSRULE N/name_npl : N0[POSS -] --> N2[NTYPE NAME, POSS -, MOD -] H0[NTYPE PLACE].
PSRULE N/npl_name : N0[POSS -] --> H0[NTYPE PLACE] N2[NTYPE NAME, POSS -, MOD -].
PSRULE N/nt_nm : N0[POSS -] --> N1[NTYPE TEMP, POSS -, MOD -] H0[NTYPE NORM].
PSRULE N/nms_nm : N0[POSS -] --> N1[NTYPE MEAS, POSS -, MOD -] H0[NTYPE NORM].
PSRULE N/npl_nm : N0[POSS -] --> N0[NTYPE PLACE, POSS -] H0[NTYPE NORM].
PSRULE N/npl_dir : N0[POSS -] --> H0[NTYPE PLACE] N0[NTYPE DIR, POSS -].
PSRULE N/dir_nm : N0[POSS -] --> N0[NTYPE DIR, POSS -] H0[NTYPE NORM].
PSRULE N/dir_npl : N0[POSS -] --> N0[NTYPE DIR, POSS -] H0[NTYPE PLACE].

; gerunds...!
PSRULE NP/det_ger : N2[WH @x, POSS -, PLU -] --> DT[WH @x] V0[VFORM ING, FIN -].
PSRULE N1/ap_ger : N1[POSS -, PLU -] --> A2[ADV -, MOD -] V0[VFORM ING, FIN -].
PSRULE N1/ger_of : N1[POSS -, PLU -] --> V0[VFORM ING, FIN -] P2[PFORM OF].

; any less a fool / more or less a fool
PSRULE N1/ap_np : N1[POSS -] --> A2[ADV +, AFORM ER] H2.

; the fiercest of / deepest brown...
PSRULE NP/dt_est : N2[NTYPE NORM, POSS -] --> DT A1[AFORM EST] (A1[ADV -]).

; no/nothing/something more ...
PSRULE NP/pro_er : N2[POSS -, NTYPE PRO] --> H2 A1[AFORM ER].

; the other/poor under
PSRULE NP/ap_pp : N2[POSS -, WH @x] --> DT[WH @x]
                                        A2[ADV -, AFORM NONE, ATYPE ATT] P2[ADJ +].

PSRULE N1/name+ : N1[POSS -] --> N0[NTYPE NAME] H1[NTYPE NAME, MOD -].
```

```
PSRULE N1/poss_n1 : N1[POSS -] --> N1[POSS +] H1[POSS -, MOD -].
PSRULE N1/n1_poss : N1[POSS +] --> H1[POSS -] N0[NTYPE POSS].
PSRULE N1/n : N1[POSS -, MOD -] --> H0.
PSRULE N1/n_s1 : N1[POSS -, MOD +] --> H0[NTYPE NORM] V2[INV -, FIN +, nopunct].
PSRULE N1/n_s2 : N1[POSS -, MOD +] --> H0[NTYPE TEMP] V2[INV -, FIN +, nopunct].
PSRULE N1/n_s3 : N1[POSS -, MOD +] --> H0[NTYPE PLACE] V2[INV -, FIN +, nopunct].
PSRULE N1/n_s4 : N1[POSS -, MOD +] --> H0[NTYPE MEAS] V2[INV -, FIN +, nopunct].
PSRULE N1/n_rel : N1[POSS -, MOD +] --> H0  CP V1[FIN +].
PSRULE N1/n_inf : N1[POSS -, MOD +] --> H0 V1[VFORM INF, FIN -].
PSRULE N1/n_pprt : N1[POSS -, MOD +] --> H0 V1[VFORM PPART].
PSRULE N1/n_ing : N1[POSS -, MOD +] --> H0[NTYPE NORM] V1[VFORM ING, FIN -].
PSRULE N1/n_of : N1[POSS -, MOD +] --> H0 P2[PFORM OF].
PSRULE N1/n1_pp1 : N1[WH @x, MOD +] --> H1 P2[ADJ +, PFORM PREP, WH @x].
PSRULE N1/n1_pp2 : N1[WH @x, MOD +] --> H1 P2[PFORM THAN].
PSRULE N1/n1_pp3 : N1[WH @x, MOD +] --> H1 P2[ADJ +, PFORM WITH, WH @x].
PSRULE N1/n1_pp4 : N1[WH @x, MOD +] --> H1 P2[ADJ +, PFORM WITHOUT, WH @x].
PSRULE N1/n1_pp5 : N1[WH @x, MOD +] --> H1 P2[ADJ +, PFORM FOR, WH @x].
PSRULE N1/n1_pp6 : N1[WH @x, MOD +] --> H1 P2[PFORM AS, WH @x].
PSRULE N1/vppt_n1 : N1[WH @x, MOD +] --> V0[VFORM PPART] H1[NTYPE NORM].
PSRULE N1/vping_n1 : N1[WH @x, MOD +] --> V0[VFORM ING, FIN -] H1[NTYPE NORM].

PSRULE NP/cj_beg : N2[CONJ -, WH @y] --> CJ[CJTYPE BEGIN] H2[CONJ +, WH @y, -ta].
PSRULE NP/cj_int1 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE @z, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE @z, CONJ +, -ta].
PSRULE NP/cj_int2 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE NAME, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE NORM, CONJ +, -ta].
PSRULE NP/cj_int3 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE TEMP, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE PRO, CONJ +, -ta].
PSRULE NP/cj_int4 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE NORM, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE NUM, CONJ +, -ta].
PSRULE NP/cj_int5 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE NORM, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE PLACE, CONJ +, -ta].
PSRULE NP/cj_int6 : N2[CONJ @c, WH @y] -->
H2[POSS @x, WH @y, NTYPE PLACE, CONJ -, -ta] (+pco)
N2[POSS @x, WH @y, NTYPE NORM, CONJ +, -ta].

PSRULE NP/cj_end : N2[CONJ +, WH @y] --> CJ[CJTYPE END] (A2[ADV +]) (+pco)
                                         H2[CONJ -, WH @y].
PSRULE N1/cj_beg : N1[CONJ -, WH @y] --> CJ[CJTYPE BEGIN] H1[CONJ +, WH @y].
PSRULE N1/cj_int1 : N1[CONJ @c, WH @y] -->
H1[POSS @x, WH @y, NTYPE @z, CONJ -, -ta] (+pco)
N1[POSS @x, WH @y, NTYPE @z, CONJ +, -ta].
```

```
PSRULE N1/cj_end : N1[CONJ +, WH @y] --> CJ[CJTYPE END] H1[CONJ -, WH @y].
PSRULE N/cj_int : N0[CONJ @x, WH @y] -->  H0[CONJ -] (+pco) H0[CONJ +, WH @y].
PSRULE N/cj_end : N0[CONJ +, WH @y] --> CJ[CJTYPE END] H0[CONJ -, WH @y].

PSRULE P1/p_np : P1[WH @x] --> H0[SUBCAT NP] N2[WH @x, +ta].
PSRULE P1/p_n1 : P1[WH -] --> H0[SUBCAT NP] N1[NTYPE NORM, WH -, POSS -, PLU -].
PSRULE P1/p_poss : P1[WH -] --> H0[SUBCAT NP] N2[NTYPE PRO, WH -, POSS +].
PSRULE P1/p_npl : P1[WH -] --> H0[SUBCAT NP] A2[ATYPE PLACE].
PSRULE P1/p_s : P1[WH -] --> H0[SUBCAT SFIN] V2[WH -, INV -, FIN +, -sc, -cl].
PSRULE P1/p_pp : P1[WH -] --> H0[SUBCAT PP] P2[WH -, MOD -].
PSRULE P1/p_sinf : P1[WH -] --> H0[SUBCAT SINF] N2[WH -] V1[VFORM INF, FIN -].
PSRULE P1/p_ing : P1[WH -] --> H0[SUBCAT VPING] V1[VFORM ING, FIN -].
PSRULE P1/p_inf : P1[WH -] --> H0[SUBCAT VPINF] V1[VFORM INF, FIN -].
PSRULE P1/p_ppart : P1[WH -] --> H0[SUBCAT VPPRT] V1[VFORM PPART].
PSRULE P1/as_a2 : P1[WH -] --> H0[PFORM AS] A2.
PSRULE P1/than_a2 : P1[WH -] --> H0[PFORM THAN] A2.
PSRULE P1/p : P1[ADJ -, WH -] --> H0[SUBCAT NONE, PFORM PREP].
PSRULE P1/pwh : P1[ADJ -, WH +] --> H0[SUBCAT NONE, PFORM WH].
PSRULE PP/adv_p1 : P2[ADJ -, WH @x, MOD -] --> A2[ADV +] H1[WH @x].
PSRULE PP/p1 : P2[WH @x] --> H1[WH @x].
PSRULE PP/cj_beg : P2[CONJ -, WH @x] --> CJ[CJTYPE BEGIN] H2[CONJ +, WH @x].
PSRULE PP/cj_int : P2[CONJ @x, WH @x] --> H2[CONJ -, WH @x1] (+pco) H2[CONJ +, WH @x].
PSRULE PP/cj_end : P2[CONJ +, WH @x] --> CJ[CJTYPE END] H2[CONJ -, WH @x].
PSRULE P/cj_int : P0[CONJ @x] --> H0[CONJ -] H0[CONJ +].
PSRULE P/cj_end : P0[CONJ +] --> CJ[CJTYPE END] H0[CONJ -].


PSRULE A1/a : A1[MOD -] --> H0.
PSRULE A1/dg_a : A1 --> DG H1.
PSRULE A1/a_of : A1[ADV -, MOD +] --> H0 P2[PFORM OF].
PSRULE A1/a_for : A1[ADV -, MOD +] --> H0 P2[PFORM FOR].
PSRULE A1/a_pp1 : A1[ADV -, MOD +] --> H0 P2[PFORM PREP].
PSRULE A1/a_pp2 : A1[ADV -, MOD +] --> H0 P2[PFORM THAN].
PSRULE A1/a_pp3 : A1[ADV -, MOD +] --> H0 P2[PFORM WITH].
PSRULE A1/a_pp4 : A1[ADV -, MOD +] --> H0 P2[PFORM WITHOUT].
PSRULE A1/a1_pp1 : A1[ADV -, MOD +] --> H1 P2[PFORM PREP].
PSRULE A1/a1_pp2 : A1[ADV -, MOD +] --> H1 P2[PFORM WITH].
PSRULE A1/a1_pp3 : A1[ADV -, MOD +] --> H1 P2[PFORM WITHOUT].
PSRULE A1/a_s : A1[ADV -, MOD +] --> H0 V2[INV -, FIN +, -sc, -cl].
PSRULE A1/a_inf : A1[ADV -, MOD +] --> H0 V1[VFORM INF, FIN -].
PSRULE A1/a_sinf : A1[ADV -, MOD +] --> H0 P2[PFORM OF] V1[VFORM INF, FIN -].
PSRULE AP/cj_beg : A2[CONJ -] --> CJ[CJTYPE BEGIN] H2[CONJ +].
PSRULE AP/cj_int : A2[CONJ @x] --> H2[CONJ -] (+pco) H2[CONJ +].
PSRULE AP/cj_end : A2[CONJ +] --> CJ[CJTYPE END] H2[CONJ -].
PSRULE A/cj_int1 : A0[CONJ @x] --> H0[CONJ -] H0[CONJ +].
PSRULE A/cj_int2 : A0[CONJ @x] --> H0[CONJ -] V0[CONJ +, VFORM PPART].
PSRULE A/cj_int3 : A0[CONJ @x] --> V0[CONJ -, VFORM PPART] H0[CONJ +].
```

```
PSRULE A/cj_end : A0[CONJ +] --> CJ[CJTYPE END] H0[CONJ -].

PSRULE A1/adv_a1 : A1[ADV -] --> A2[ADV +] H1.
PSRULE A1/adv_er : A1[ADV +] --> A0[ADV +] H0[AFORM ER].
PSRULE A1/no_er : A1[ADV +] --> DT H0[AFORM ER].
PSRULE A1/er_adv : A1[ADV -] --> H0[AFORM ER] A0[ADV +].

PSRULE AP/a1 : A2 --> H1.

; no doubt, at least etc RR RR idioms, not even XX RR

PSRULE AP/a_a : A2 --> H0[ATYPE @x, ADV +] A0[ATYPE @x, ADV +].
PSRULE AP/a_apl : A2 --> H0[ATYPE CAT, ADV +] A0[ATYPE PLACE, ADV +].
PSRULE AP/not_a : A2 --> NG H0[ATYPE CAT, ADV +].
PSRULE AP/adv_of : A2 --> H0[ATYPE CAT, ADV +] P2[PFORM OF].

; cross-cat hacks...!

PSRULE AP/dg_ppt : A2 --> DG V0[VFORM PPART].
PSRULE AP/dg_ing : A2 --> DG V0[VFORM ING, FIN -].
PSRULE A1/adv_ppt : A1 --> A2[ADV +] V0[VFORM PPART].
PSRULE A1/adv_ing : A1 --> A2[ADV +] V0[VFORM ING, FIN -].

PROPRULE HFC_V : [V +, N -] --> [H +], U. F(0) = F(1), F in VHEAD.
PROPRULE HFC_N : [V -, N +] --> [H +], U. F(0) = F(1), F in NHEAD.
PROPRULE HFC_P : [V -, N -] --> [H +], U. F(0) = F(1), F in PHEAD.
PROPRULE HFC_A : [V +, N +] --> [H +], U. F(0) = F(1), F in AHEAD.
PROPRULE AGR1 : V2 --> [N +, V -], [H +, BAR 1]. PLU(1) = PLU(2).
PROPRULE AGR2 : N2 --> [MINOR DET], [H +]. PLU(1) = PLU(2).
PROPRULE WH1 : N1 --> [H +], U. WH(0) = WH(1).
PROPRULE INV : V2 --> H2, U. INV(0) = INV(1).
PROPRULE TXTCAT1 : V2 --> H2, U. TXTCAT(0) = TXTCAT(1).
PROPRULE TXTCAT2 : N2 --> [H +, BAR(1,2)], U. TXTCAT(0) = TXTCAT(1).
PROPRULE TXTCAT3 : N1 --> H1, U. TXTCAT(0) = TXTCAT(1).
PROPRULE PUNCT : [TA +, BAR(1, 2)] --> [H +], [TXTCAT ADJUNCT].
F(0) = F(2), F in PUNCTS.

DEFRULE CONJ1 : [N, V, ~COLON, ~TA] --> [H +, CONJ @], U.  CONJ(1) = -.
DEFRULE CONJ2 : [N, V, ~COLON, ~TA] --> [N, V, ~CONJ], U. CONJ(1) = -.
DEFRULE AUX : V1 --> [H0], U. AUX(1) = -.
DEFRULE POSS1 : N1 --> U. POSS(0) = -.
DEFRULE POSS2 : [] --> [N2], U. POSS(1) = -.
DEFRULE ADJUNCT : [TXTCAT ADJUNCT] --> U. F(0) = -, F in PUNCTS.
DEFRULE COLON : V2 --> H2, U. COLON(1) = -.
DEFRULE SCOLON1 : [] --> [BAR 2], U. SCOLON(1) = -.
DEFRULE SCOLON2 : [] --> H2, U. SCOLON(1) = -.
DEFRULE SCOLON3 : [BAR 2] -->  U. SCOLON(0) = -.
```

```
DEFRULE SCOLON4 : N1 -->  U. SCOLON(0) = -.
DEFRULE SCOLON5 : V1 -->  U. SCOLON(0) = -.
DEFRULE SCOLON6 : [N +, V -] -->  H1, U. SCOLON(1) = -.
DEFRULE SCOLON7 : [N -, V +] -->  H1, U. SCOLON(1) = -.
DEFRULE SCOLON8 : [] --> N1, U. SCOLON(1) = -.
DEFRULE SCOLON9 : [] --> V1, U. SCOLON(1) = -.
DEFRULE TA1 : [N +, V -] --> [H +, BAR 1], U. TA(1) = -.
DEFRULE TA2 : N2 --> [H +], [N +, V -, BAR(1,2)], U. TA(2) = -.
DEFRULE TA3 : N2 --> [H +], V2, U. TA(2) = -.

WORD APP$ : DT[WH -, PLU @x, POSS +].
WORD AT : DT[PLU @x, POSS -, WH -].
WORD AT1 : DT[PLU -, POSS -, WH -].
WORD BCS : P0[SUBCAT SCOMP, PFORM PREPS].
WORD BTO : P0[SUBCAT VPINF, PFORM PREPS], P0[SUBCAT PP, PFORM PREPS].
WORD CC : CJ[CJTYPE END].
WORD CCB : CJ[CJTYPE END].
WORD CF : P0[SUBCAT SFIN, PFORM PREPS, CONJ -], P0[SUBCAT NONE, PFORM PREPS, CONJ -].
WORD CS : P0[SUBCAT SFIN, PFORM PREPS, CONJ -],
          P0[SUBCAT VPPRT, PFORM PREPS, CONJ -],
          P0[SUBCAT VPING, PFORM PREPS, CONJ -],
          P0[SUBCAT PP, PFORM PREPS, CONJ -].
WORD CSA : P0[SUBCAT @y, PFORM AS, CONJ -].
WORD CSN : P0[SUBCAT @y, PFORM THAN, CONJ -].
WORD CST : CP.
WORD CSW : CP.
WORD DA : A0[ADV -, ATYPE ATT, AFORM NONE, CONJ -],
   N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -], DT[POSS -, WH -, PLU @x].
WORD DA1 : A0[ADV @x, ATYPE ATT, AFORM NONE, CONJ -],
   N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -], DT[POSS -, WH -, PLU -].
WORD DA2 : A0[ADV @x, ATYPE ATT, AFORM NONE, CONJ -],
   N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD DA2R : A0[ADV -, ATYPE @x, AFORM ER, CONJ -],
   N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD DAR : A0[ADV @x, ATYPE CAT, AFORM ER, CONJ -],
   N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -].
WORD DAT : A0[ADV -, ATYPE @x, AFORM EST, CONJ -],
   N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -].
WORD DB : N0[NTYPE PART, PLU @x, POSS -, WH -, CONJ -],
   N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -],
   A0[AFORM NONE, ATYPE CAT, ADV +, CONJ -].
WORD DB2 : N0[NTYPE PART, PLU +, POSS -, WH -, CONJ -].
WORD DD : DT[PLU @x, WH -, POSS -],
   N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -].
WORD DD1  : DT[PLU -, WH -, POSS -],
   N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD DD2 : DT[PLU +, WH -, POSS -],
```

```
   N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD DDQ : DT[WH +, PLU @x, POSS -],
   N2[NTYPE PRO, PLU @x, POSS -, WH +, CONJ -].
WORD DDQ$ : DT[WH +, POSS -, PLU @x],
   N2[NTYPE PRO, PLU @x, POSS -, WH +, CONJ -].
WORD DDQV : DT[WH +, PLU @x, POSS -],
   N2[NTYPE PRO, PLU @x, POSS -, WH +, CONJ -].
WORD EX : N2[NTYPE THERE, PLU @x, POSS -, WH -, CONJ -].
WORD ICS : P0[SUBCAT SFIN, PFORM PREPS],
    P0[SUBCAT VPING, PFORM PREPS, CONJ -],
    P0[SUBCAT NP, PFORM PREP],  P0[SUBCAT PP, PFORM PREP].
WORD IF : P0[SUBCAT SINF, PFORM FORS, CONJ -],
   P0[SUBCAT NP, PFORM FOR, CONJ -], P0[SUBCAT VPING, PFORM FOR, CONJ -].
WORD II : P0[SUBCAT @x, PFORM PREP, CONJ -].
WORD IO : P0[SUBCAT NP, PFORM OF, ADJ +, CONJ -],
   P0[SUBCAT VPING, PFORM OF, ADJ +, CONJ -], PT.
WORD IW : P0[SUBCAT NP, PFORM WITH, CONJ -],
   P0[SUBCAT VPING, PFORM WITHOUT, CONJ -].
WORD JA : A0[ADV -, ATYPE PRD, AFORM NONE, CONJ -].
WORD JB : A0[ADV -, ATYPE ATT, AFORM NONE, CONJ -].
WORD JBR : A0[ADV -, ATYPE ATT, AFORM ER, CONJ -].
WORD JBT : A0[ADV -, ATYPE ATT, AFORM EST, CONJ -].
WORD JJ : A0[ADV -, ATYPE @x, AFORM NONE, CONJ -].
WORD JJR : A0[ADV -, ATYPE @x, AFORM ER, CONJ -].
WORD JJT : A0[ADV -, ATYPE @x, AFORM EST, CONJ -].
WORD JK : A0[ADV -, ATYPE CAT, AFORM NONE, CONJ -].
WORD LE : CJ[CJTYPE BEGIN].
WORD MC : N0[NTYPE NUM, PLU @x, POSS -, WH -, CONJ -].
WORD MC$ : N2[NTYPE NUM, PLU @x, POSS +, WH -, CONJ -].
WORD MC-MC : N2[NTYPE NUM, PLU @x, POSS -, WH -, CONJ -].
WORD MC1 : N0[NTYPE NUM, PLU -, POSS -, WH -, CONJ -],
   DT[POSS +, PLU -, WH -].
WORD MC2 : N0[NTYPE NUM, PLU @x, POSS -, WH -, CONJ -].
WORD MD : A0[ADV @x, ATYPE NUM, AFORM NONE, CONJ -],
   N0[NTYPE NUM, PLU -, POSS -, WH -, CONJ -].
WORD MF : N2[NTYPE NUM, PLU @x, POSS -, WH -, CONJ -].
WORD NC2 : N0[NTYPE NORM, PLU +, POSS -, WH -, CONJ -].
WORD ND1 : N0[NTYPE DIR, PLU -, POSS -, WH -, CONJ -].
WORD NN : N0[NTYPE NORM, PLU @x, POSS -, WH -, CONJ -].
WORD NN1 : N0[NTYPE NORM, PLU -, POSS -, WH -, CONJ -].
WORD NN1$ : N0[NTYPE NORM, PLU +, POSS +, WH -, CONJ -].
WORD NN2 : N0[NTYPE NORM, PLU +, POSS -, WH -, CONJ -].
WORD FW : N0[NTYPE NORM, PLU @x, POSS -, WH -, CONJ -].
WORD NNJ : N0[NTYPE NORM, PLU @x, POSS -, WH -, CONJ -].
WORD NNJ1 : N0[NTYPE NORM, PLU -, POSS -, WH -, CONJ -].
WORD NNJ2 : N0[NTYPE NORM, PLU +, POSS -, WH -, CONJ -].
WORD NNL :  N0[NTYPE PLACE, PLU @x, POSS -, WH -, CONJ -].
```

```
WORD NNL1 : NO[NTYPE PLACE, PLU -, POSS -, WH -, CONJ -].
WORD NNL2 : NO[NTYPE PLACE, PLU +, POSS -, WH -, CONJ -].
WORD NNO : NO[NTYPE NORM, PLU @x, POSS -, WH -, CONJ -].
WORD NNO1 : NO[NTYPE NORM, PLU -, POSS -, WH -, CONJ -].
WORD NNO2 : NO[NTYPE NORM, PLU +, POSS -, WH -, CONJ -].
WORD NNS : NO[NTYPE TIT, PLU @x, POSS -, WH -, CONJ -],
           NO[NTYPE NORM, PLU @x, POSS -, WH -, CONJ -].
WORD NNS1 : NO[NTYPE TIT, PLU -, POSS -, WH -, CONJ -],
           NO[NTYPE NORM, PLU -, POSS -, WH -, CONJ -].
WORD NNS2 : NO[NTYPE TIT, PLU +, POSS -, WH -, CONJ -],
           NO[NTYPE NORM, PLU +, POSS -, WH -, CONJ -].
WORD NNSA1 : NO[NTYPE POSTTIT, PLU -, POSS -, WH -, CONJ -].
WORD NNSA2 : NO[NTYPE POSTTIT, PLU +, POSS -, WH -, CONJ -].
WORD NNSB : NO[NTYPE PRETIT, PLU @x, POSS -, WH -, CONJ -].
WORD NNSB1 : NO[NTYPE PRETIT, PLU -, POSS -, WH -, CONJ -].
 WORD NNSB2 : NO[NTYPE PRETIT, PLU +, POSS -, WH -, CONJ -].
WORD NNT : NO[NTYPE TEMP, PLU @x, POSS -, WH -, CONJ -].
WORD NNT1 : NO[NTYPE TEMP, PLU -, POSS -, WH -, CONJ -].
WORD NNT2 : NO[NTYPE TEMP, PLU +, POSS -, WH -, CONJ -].
WORD NNU : NO[NTYPE MEAS, PLU @x, POSS -, WH -, CONJ -].
WORD NNU1 : NO[NTYPE MEAS, PLU -, POSS -, WH -, CONJ -].
WORD NNU2 : NO[NTYPE MEAS, PLU +, POSS -, WH -, CONJ -].
WORD NP : NO[NTYPE NAME, PLU @x, POSS -, WH -, CONJ -].
WORD NP1 : NO[NTYPE NAME, PLU -, POSS -, WH -, CONJ -].
WORD NP2 : NO[NTYPE NAME, PLU +, POSS -, WH -, CONJ -].
WORD NPD1 : NO[NTYPE TEMP, PLU -, POSS -, WH -, CONJ -],
   N2[NTYPE TEMP, PLU -, POSS -, WH -, CONJ -].
WORD NPD2 : NO[NTYPE TEMP, PLU +, POSS -, WH -, CONJ -].
WORD NPM1 : NO[NTYPE MEAS, PLU -, POSS -, WH -, CONJ -],
   N2[NTYPE MEAS, PLU -, POSS -, WH -, CONJ -].
WORD NPM2 : NO[NTYPE MEAS, PLU +, POSS -, WH -, CONJ -].
WORD PN : N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -].
WORD PN1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PNQO : N2[NTYPE PRO, PLU -, POSS -, WH +, CONJ -].
WORD PNQS : N2[NTYPE PRO, PLU -, POSS -, WH +, CONJ -].
WORD PNQV$ : N2[NTYPE PRO, PLU -, POSS -, WH +, CONJ -].
WORD PNQVO : N2[NTYPE PRO, PLU -, POSS -, WH +, CONJ -].
WORD PNQVS : N2[NTYPE PRO, PLU -, POSS -, WH +, CONJ -].
WORD PNX1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PP$ : N2[NTYPE PRO, PLU -, POSS +, WH -, CONJ -].
WORD PPH1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PPHO1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PPHO2 : N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD PPHS1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PPHS2 : N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD PPIO1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PPIO2 : N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
```

```
WORD PPIS1 : N2[NTYPE PRO, PLU -, POSS -, WH -, CONJ -].
WORD PPIS2 : N2[NTYPE PRO, PLU +, POSS -, WH -, CONJ -].
WORD PPX1 : N2[NTYPE REFL, PLU -, POSS -, WH -, CONJ -].
WORD PPX2 : N2[NTYPE REFL, PLU +, POSS -, WH -, CONJ -].
WORD PPY : N2[NTYPE PRO, PLU @x, POSS -, WH -, CONJ -].
WORD RA : A0[AFORM NONE, ATYPE POST, ADV +, CONJ -].
WORD REX : A0[AFORM NONE, ATYPE XCOMP, ADV +, CONJ -].
WORD RG :  A0[AFORM NONE, ATYPE DG, ADV +, CONJ -].
WORD RGA : A0[AFORM NONE, ATYPE POST, ADV +, CONJ -].
WORD RGQ : A0[AFORM NONE, ATYPE HOW, ADV +, CONJ -].
WORD RGQV : A0[AFORM NONE, ATYPE HOW, ADV +, CONJ -].
WORD RGR : A0[AFORM ER, ATYPE @x, ADV +, CONJ -].
WORD RGT : A0[AFORM EST, ATYPE @x, ADV +, CONJ -].
WORD RL : A0[AFORM NONE, ATYPE PLACE, ADV +, CONJ -], PT.
WORD RP : A0[AFORM NONE, ATYPE CAT, ADV +, CONJ -], PT.
WORD RPK : A0[AFORM NONE, ATYPE CAT, ADV +, CONJ -].
WORD RR : A0[AFORM NONE, ATYPE CAT, ADV +, CONJ -].
WORD RRQ : P0[PFORM WH, SUBCAT NONE, CONJ -].
WORD RRQV : P0[PFORM WH, SUBCAT NONE, CONJ -].
WORD RRR : A0[AFORM ER, ATYPE CAT, ADV +, CONJ -].
WORD RRT : A0[AFORM EST, ATYPE CAT, ADV +, CONJ -].
WORD RT : N2[NTYPE TEMP, POSS -, PLU -, WH -, CONJ -].
WORD TO : V0[FIN -, AUX TO, VFORM INF, CONJ -].
WORD UH : IJ.
WORD VB0 : V0[FIN +, INV @y, AUX BE, VFORM BSE, PLU @x, CONJ -].
WORD VBDR : V0[FIN +, INV @y, AUX BE, VFORM PAST, PLU @x, CONJ -].
WORD VBDZ : V0[FIN +, INV @y, AUX BE, VFORM PAST, PLU -, CONJ -].
WORD VBG : V0[FIN -, INV @y, AUX BE, VFORM ING, PLU @x, CONJ -].
WORD VBM : V0[FIN +, INV @y, AUX BE, VFORM PRES, PLU -, CONJ -].
WORD VBN : V0[FIN +, INV @y, AUX BE, VFORM PPART, PLU @x, CONJ -].
WORD VBR : V0[FIN +, INV @y, AUX BE, VFORM PRES, PLU @x, CONJ -].
WORD VBZ : V0[FIN +, INV @y, AUX BE, VFORM PRES, PLU -, CONJ -].
WORD VD0 : V0[FIN +, INV @y, AUX DO, VFORM BSE, PLU @x, CONJ -].
WORD VDD : V0[FIN +, INV @y, AUX DO, VFORM PAST, PLU @x, CONJ -].
WORD VDG : V0[FIN -, INV @y, AUX DO, VFORM ING, PLU @x, CONJ -].
WORD VDN : V0[FIN +, INV @y, AUX DO, VFORM PPART, PLU @x, CONJ -].
WORD VDZ : V0[FIN +, INV @y, AUX DO, VFORM PRES, PLU -, CONJ -].
WORD VH0 : V0[FIN +, INV @y, AUX HAVE, VFORM BSE, PLU @x, CONJ -].
WORD VHD : V0[FIN +, INV @y, AUX HAVE, VFORM PAST, PLU @x, CONJ -].
WORD VHG : V0[FIN -, INV @y, AUX HAVE, VFORM ING, PLU @x, CONJ -].
WORD VHN : V0[FIN +, INV @y, AUX HAVE, VFORM PPART, PLU @x, CONJ -].
WORD VHZ : V0[FIN +, INV @y, AUX HAVE, VFORM PRES, PLU -, CONJ -].
WORD VM : V0[FIN +, INV @y, AUX MODAL, VFORM PRES, PLU @x, CONJ -].
WORD VMK : V0[FIN +, INV @y, AUX CAT, VFORM PRES, PLU @x, CONJ -].
WORD VVD : V0[FIN +, AUX -, VFORM PAST, PLU @x, CONJ -].
WORD VVG : V0[FIN -, AUX -, VFORM ING, PLU @x, CONJ -].
WORD VVN : V0[FIN +, AUX -, VFORM PPART, PLU @x, CONJ -].
```

```
WORD VVZ : V0[FIN +, AUX -, VFORM PRES, PLU -, CONJ -].
WORD VVGK : V0[FIN -, AUX -, VFORM ING, PLU @x, CONJ -].
WORD VVNK : V0[FIN +, AUX -, VFORM PPART, PLU @x, CONJ -].
WORD VV0 : V0[FIN +, AUX -, VFORM BSE, PLU @x, CONJ -].
WORD XX : NG.
WORD ZZ1 : N0[PLU -, NTYPE NAME, POSS -, WH -, CONJ -],
   N2[PLU -, NTYPE NAME, POSS -, WH -, CONJ -].
WORD ZZ2 : N0[PLU +, NTYPE NAME, POSS -, WH -, CONJ -].
WORD $ : N0[PLU @x, NTYPE POSS, POSS +, WH -, CONJ -].

WORD \, : [PUNCT comma]. ; punctuation shld be tagged ,_, etc.
WORD \; : [PUNCT semicolon].
WORD \: : [PUNCT colon].
WORD \! : [PUNCT exmark].
WORD \? : [PUNCT qumark].
WORD \- : [PUNCT dash].
WORD \( : [PUNCT lbrack].
WORD \) : [PUNCT rbrack].
```

Appendix 2 – Plot of (Un)Punctuated Susanne Data