SDP - Spoken Dialogue Parser

David McKelvie

September 3, 1998

Summary

This report describes work done on part of speech tagging and parsing the MAPTASK Corpus in the "Robust Parsing and Part-of-Speech Tagging of Transcribed Speech Corpora" project, funded by the ESRC (project R000236800). This report concentrates on the implementation of the software developed in the project and the format of the SGML annotation of the parse trees. An overview of the project's aims and results can be found in [McKelvie 98b] and an analysis of the speech disfluencies found while parsing the corpus can be found in [McKelvie 98a].

Contents

1 Segmentation **5**

1.1 Generating Corpus for tagging 5

1.2 Conversion to SGML 5

1.3 Normalisation of tokens 6

1.4 Re-segmentation of words 6

1.5 How it works 7

2 Part of Speech Tag Set used for MT corpus **8**

2.1 Major Word Classes 8

2.1.1 MAIN VERBS 8

2.1.2 NOUNS 8

2.1.3 ADJECTIVES 9

2.1.4 ADVERBS 9

2.2 Minor Word Classes 10

2.2.1 AUXILIARY VERBS 10

2.2.2 DETERMINERS 11

2.2.3 PRONOUNS 11

2.2.4 PREPOSITIONS 12

2.2.5 CONJUNCTIONS 12

2.2.6 INTERJECTIONS 12

2.2.7 PUNCTUATION 13

2.3 Some problematic areas 13

2.4 Example of Tag Set Usage 13

3 Part of Speech Tagging **18**

3.1 Introduction 18

3.1.1 Rule-based post tagging 19

3.2 Hand-Correction 19

3.3 Evaluation 20

4 Parsing **22**

4.1 Introduction 22

4.2 Chart implementation 24

4.3 Evaluation 24

5 User Manual **25**

5.1 Introduction 25

5.2 To Run SDP 25

5.3 Modifying the behaviour of the Parser 27

5.4 Format of SGML input/output files 28

5.5 Format of the grammar rule file 28

5.6 Chart Parser 30

6 Grammar for MapTask Corpus **31**

6.1 Introduction 31

6.2 Lexical introduction Rules 31

6.2.1 Lexical Disfluencies 31

6.2.2 Normal Lexical Introduction Rules 32

6.3 Sequencing of Grammar rules 35

6.4 Disfluency Rules 36

6.5 Edit Signals 36

6.6 Adjective phrases 37

6.7 Adverbial Phrases 37

6.8 Determiners 38

6.9 Noun Phrases . . . 38
6.9.1 Relative noun phrases . . . 39
6.9.2 Aborted Noun phrases . . . 40
6.10 Prepositional Phrases . . . 40
6.10.1 Aborted PP's . . . 42
6.11 Verb phrases . . . 42
6.11.1 Subcategorisation frames . . . 42
6.11.2 Other VP rules . . . 44
6.11.3 Aborted Verb phrases . . . 44
6.12 Clauses . . . 44
6.12.1 Aborted sentences . . . 45
6.13 Utterances . . . 45
6.14 Defaults . . . 46
6.15 Lexicon . . . 47

7 Description of the SGML markup of the parsed corpus 49
7.1 Top level elements . . . 49
7.2 The structure of PARA . . . 50
7.3 Phrase level elements . . . 50
7.3.1 Phrase structure . . . 50
7.3.2 Definitions of phrasal attributes . . . 52
7.4 The structure of the summary . . . 54
7.5 Etc . . . 55

8 Availability 57
8.1 Availability of software, data, documentation . . . 57
8.2 WWW demonstration page . . . 57

Chapter 1

Segmentation

1.1 Generating Corpus for tagging

The starting point for the tagging of the MapTask Corpus is the per-speaker word-level transcriptions produced by the ESRC funded "Restructuring the HCRC Map Task Corpus for Wide Distribution" project. These transcripts contain a sequence of tokens spoken by each speaker in the conversations. Each token has its duration marked. A token is a word, a pause or some other noise. The description of the format of these transcripts can be found in "Coding Rubric for HCRC Map Task Word-level Transcription (Draft Version 0.1)", `http://www.cogsci.ed.ac.uk/~ht/rubric.ps` .

The first stage is to convert the corpus data into a format which can be used by the POS tagger. This involves some manipulation of the format of the files and some re-segmentation of the words.

1.2 Conversion to SGML

The program `make-input` converts a word-transcript X label file into an SGML form (described by the Document Type Description `conversation.dtd`). This DTD is as follows:

```
<!-- Document Type Description for MapTask Tagging/Parsing project -->

<!ELEMENT conversation - - (W)*>
<!ATTLIST conversation
        id      ID              #REQUIRED -- conversation id --
        speaker (Follower|Giver) #REQUIRED
        date    CDATA           #REQUIRED -- Date/time when this file made -->
<!ELEMENT w - - (#PCDATA)>
<!ATTLIST w
        start   NUTOKEN         #IMPLIED -- start time in seconds --
        dur     NUTOKEN         #IMPLIED -- duration in seconds --
        utt     NUMBER          #IMPLIED -- second field of X-label file --
        pos     CDATA           #IMPLIED -- Part of Speech --
        type    (normal|clitic)  normal >
```

1.3 Normalisation of tokens

The program `prepare-words` reads the SGML file produced by `make-input` and tidies up the word tokens into a form more suited for tagging. This consists mainly of the normalisation of microtags and the splitting off of clitics (e.g. John's → John +'s, he'll → he +'ll).

In detail, the actions taken are:

1. Initial partial tokens (IP microtag).
   {ip|til=until} → 'til,
   {ip|cause=because} → 'cause,
   otherwise {ip|XXX=YYY} → YYY.
2. Corrected forms: In BR, IP and PH microtags, where there is a correct form given (after a =), then we replace the microtag with the corrected form. E.g. {br|XXX=YYY} → YYY.
3. One word microtags: FG, GG, FP, CI are replaced by the word in the microtag.
4. Aborted words: AB, RP are replaced by word fragments. {ab|XXX} → XXX–, {rp|XXX} → –XXX,
5. Letters pronounced as such: LE microtags are quoted, e.g. {le|u s} → "u" "s".
6. Noises: PH microtags are replaced with the word &noise.
7. Clitics: Word final 's, s', 'll, 'd, 've, 're, 'm are split off as separate <W TYPE=CLITIC> elements. These don't have START or DUR attributes marked. Note that word final n't is not treated as a clitic.

1.4 Re-segmentation of words

Some short sequences of words have proven difficult to tag as separate words, and may be considered as multi-word lexicon units. The program `prepare-words-2` takes the output of `prepare-words` and joins up such word sequences.

Presently the word sequences treated in this way are:

more or less	→ more_or_less[1]
sort of	→ sort_of
kind of	→ kind_of
instead of	→ instead_of
because of	→ because_of
in between	→ in_between
crest falls	→ crest_falls[2]
diamond mine	→ diamond_mine[2]
gold mine	→ gold_mine[2]
each other	→ each_other

Notes:

1. 'More or less' is tagged as one unit, since we did not want to have to complicate the grammar to allow conjunctions inside predeterminers.
2. These names of map landmarks have been treated as one unit due to problems in tagging them correctly. 'Falls' was tagged as a verb, and 'mine' was tagged as a personal pronoun.

1.5 How it works

The sequence of UNIX commands will create a SGML word file from an X-label word transcript file which is suitable as input to the tagger.

```
make-input wlab/q1ec1.g.words | \
prepare-words | \
prepare-words-2 > tag/q1ec1.g.words
```

Chapter 2

Part of Speech Tag Set used for MT corpus

The part-of-speech tag set used is a modified version of the Brown Corpus tag set. The corpus was initially tagged using the Brown tag set and the tag set modified to improve the tagging performance. Some tags have been removed because they did not occur often enough in the MapTask Corpus to train effectively. A few have been added in order to model filled pauses, fragmentary words, and to make some finer distinctions among adverbs.

2.1 Major Word Classes

2.1.1 MAIN VERBS

VB	VERB BASE FORM
VBD	VERB PAST FINITE FORM eg saw, looked, went
VBG	VERB PRESENT PARTICIPLE (-ing) FORM
VBN	VERB PAST PARTICIPLE eg burnt, gone
VBZ	VERB 3rd PERSON SINGULAR FORM (+s)

Note: The verb tags used are those defined in the Brown tagset. These tags are used for main verbs (i.e. not auxiliary verbs). See also section 2.2.1.

2.1.2 NOUNS

NN	COMMON NOUN SINGULAR OR MASS
NNS	COMMON NOUN PLURAL
NP	PROPER NOUN SINGULAR

There are some differences in our noun tags from the Brown tagset. Firstly, we split genitive clitics off from nouns and treat +'s as a separate token with its own tag, thus removing the need for the Brown tags (NN$, NNS$, NP$, NPS$, and NR$). Secondly, there are no plural proper nouns or plural adverbial nouns in the MapTask Corpus, so we don't use the Brown NPS and NRS tags. Finally, we don't use the Brown tag NR for 'adverbial nouns' (e.g. east, home, monday, etc), instead we take these as either (NN noun, RB adverb or RP (see below)).

2.1.3 ADJECTIVES

JJ	ADJECTIVES
JJR	COMPARATIVE ADJECTIVE
JJT	SUPERLATIVE ADJECTIVE with or without -est morphology

As for nouns, we have no need for Brown's JJ$ tag (e.g. great's), and we do not make the distinction between Brown's JJS (superlative adjectives with -est morphology) and JJT (superlative adjectives without -est morphology e.g. innermost or chief) tags, mapping both to JJT.

2.1.4 ADVERBS

Adverbs are a real mixed bag, so we attempt to subclassify them.

QL	QUALIFIER ADVERB - premodifies adjective or adverbial particles, prepositional adverbs or prepositions
QLDT	QUALIFIER ADVERB - premodifies pronouns or determiners,
QLP	enough - when it postmodifies JJ or RB
RB	ADVERB
RBR	COMPARATIVE ADVERBS +ER
WQL	how (as qualifier as in 'how many')
WRB	how, when, whenever, where, whereabout, whereabouts, why,
NOT	not

This is basically the same as the Brown tagset for adverbs, except that we use NOT instead of *. We don't use RBT for superlative adverbs, since there are none in the MapTask Corpus lexicon. And we introduce a new tag QLDT for adverbs premodifying determiners as in 'nearly two metres'. Finally, we don't use a special tag (RN) for the word 'afar'.

2.2 Minor Word Classes

2.2.1 AUXILIARY VERBS

In comparison with the Brown tagset, we do not distinguish between positive and negative forms of auxilary verbs. So that tags BED*, etc are not used. Also some rarer forms of auxilary verbs are tagged as main verbs to avoid having rarely used tags.

TO	to (as verbal particle)
	FORMS OF THE VERB be
BE	be
BEM	am, +'m, was
BER	are, +'re, were
BEZ	is, +'s, was

BEG(being) is tagged as VBG as it is so rare in corpus.
BED(were) is tagged as BER due to rarity in corpus.
BEDZ(was) is tagged as BEM,BEZ due to rarity in corpus.
BEN(been) is tagged as VBN due to rarity in corpus.

	FORMS OF THE VERB do
DO	do, did
DOZ	does, did

Other lexical forms of DO are tagged as main verbs. DOD(did) is tagged as DO or DOZ due to rarity.

	FORMS OF THE VERB have
HV	have, +'ve, had
HVZ	has, +'s, had

HVG 'having' is tagged as VBG as it is so rare in corpus. 'had' (tagged HVD and HVN) has been retagged as HV,HVZ, or VBN due to rarity.

	MODAL VERBS
MD	MODAL +'d, +'ll, can, could, may, might, must, need, ought, shall, should, will, would PLUS negative forms of the above

2.2.2 DETERMINERS

The Brown tags for predeterminers (ABL, ABN, ABX) have been collapsed to one tag DPR. Similarly, the Brown tags for demonstrative determiners (DT, DT$, DTI, DTS and DTX) have been collapsed to DT. PP$ has been renamed PPG, and a new tag GEN has been defined for genitive clitics.

	PREDETERMINERS (can occur before a CENTRAL DETERMINER)
DPR	all, both, double, half, just, quarter, quite, such
	CENTRAL DETERMINERS
AT	ARTICLES: a, an, no, the
DT	SING DEMONSTRATIVE: another, each, that, this QUANTIFIERS: any, some, either PLUR DEMONSTRATIVE: these, those
PPG	POSSESSIVES: her, his, its, my, our, their, your
WDT	INTERROGATIVE: what, which
	POST DETERMINERS (can appear after a central determiner)
AP	few, further, final, last, least, less, little, many more, most, much, next, only, other, same, single very
CD	CARDINAL NUMBERS e.g. one, two, three, etc
OD	ORDINAL NUMBER e.g. first, second, third
GEN	+'s Genitive clitic

2.2.3 PRONOUNS

EX	there (existential subject)
PD	DEMONSTRATIVE: this, that, these, those
WPS	INTERROGATIVE SUBJ: who, what, whatever, which
WPO	INTERROGATIVE OBJ: who, what, whatever, which
PPS	PERSONAL SUBJECT 3RD SING: he, she, it
PPSS	PERSONAL SUBJECT NON-3RD SING: I, we, you, they
PPO	PERSONAL OBJECT: it, us, you, me, him, her, +'s, them
PPL	PERSONAL REFL: herself, himself, itself, myself, yourself ourselves, yourselves, themselves
PPG2	PERSONAL POSS: hers, his, mine, mines, ours, theirs, yours

PR	RELATIVE: who, which, that
PN	OTHER PRONOUNS: any, anything, anywhere(pro-pp?) everything, everywhere none, nothing, nowhere, neither some, something, somewhere 'them both', both, all, one RECIPROCAL 'each other' occurs twice - treat as a single token enough, ordinal numbers

2.2.4 PREPOSITIONS

IN	PREPOSITIONS (which take NP as complement)
RP	either ADVERBIAL PREPOSITIONS (prepositions which take NO complement) or VERBAL PARTICLES (prepositions which form part of verbs) also included here are some uses of directions e.g. north

2.2.5 CONJUNCTIONS

CC	and, but, either, neither, nor, or, though, yet
CS	'cause, 'til, after, as, because, before, if, like, once, since so, than, that, though, unless, until, whereas, whether, while

CS means a clause initial element. CC is used for clause internal conjunctions.

2.2.6 INTERJECTIONS

AFF	POSITIVE: right, okay, okey-dokey, mmhmm, uh-huh, yeah, yes, aye, fine, correct, rightee-ho, right-o, mm-mm, uh-uh, NEUTRAL: now, well NEGATIVE: no, nope
	What Quirk et.al call reaction signals/initiators
FP	FILLED PAUSE: eh, ehm, er, erm, hmm, mm, uh, um
UH	INTERJECTION ah, aha, oh, bang, christ, dear, fine, god, gosh, ha, hell, hurrah, jeez, jesus, my, och, oo, oops, phew, please, say, smashing, sorry, splat, super, ugh, whoa, whoops, why, wow

now also includes what used to be FW (FOREIGN WORD)
alles, culpa, es, fini, finito, gemacht, mea,
tu, verstanden

NOI other NOISEs made by the speaker: &noise, &laugh, &indecipherablespeech

PAU PAUSE: ...

FRAG FRAGMENTED WORDS (aborted words)

The MAPTASK Corpus microtags FP, GG, FG have been distributed among the tags AFF, FP and UH. AFF tend to appear at the beginning or end of utterances (sometimes as utterances on their own). FP tend to signal speech disfluencies. UH occur inside utterances.

Aborted words are tagged with a special tag FRAG. An alternative is to allow them to be any tag and let the tagger try and assign the most plausible tag to them.

2.2.7 PUNCTUATION

We don't have tags for punctuation, as they are stripped out before part-of-speech tagging. SENT is used to tag those pauses which are used to segment the speech stream into units for tagging. Currently all pauses are used in this way. In the future, once accurate pause durations have been determined, only pauses over some duration threshhold will be used as tagging unit separators.

SENT TAGGING UNIT SEPARATOR – NOT IN LEXICON –

2.3 Some problematic areas

There are a number of problems with the tag set, which require further thought. These are discussed here:

1. The word 'like'. As Jim Miller has noted, this word is used in a number of different ways and it is difficult to decide on what its POS tag should be.
2. Adjectives which subcategorise for noun phrases or prepositional phrases, such as 'like' + NP, or 'near' + NP/PP, occur rarely in the MAPTASK Corpus and are at present not well tagged.
3. Non-verbal uses of the word 'say', such as "does that take you up to the top right-hand corner say of the ... the ruined monastery".

2.4 Example of Tag Set Usage

The following is the Giver's speech from the MAPTASK Corpus conversation *q1ec1*, tagged using the above tag set.

Notes: Items in square brackets are pauses, with their durations in seconds. The actual durations are only rather rough estimates at the moment. Items starting with + are clitics which have been split off from the previous word. A few multi-word items have been tagged as one unit, for example *sort_of* . Tags may be followed by a slash and a number; the number shows the number of different tags this word has in total. Tags in capitals have been corrected by hand, the others were tagged using the Xerox automatic tagger. A few tags have a question mark attached; I find these problematic.

q1ec1.g.tag

```
[0.0000] okay  starting off  we   are above a  caravan park [0.9795] we   are
         aff/2 vbg/2    rp/2 ppss ber in/2  at nn      nn           ppss ber

going to   go due  south straight south and  then we   +'re going to   g--
vbg   to/3 vb ql/2 rp/4  ql/3     rp/4  cc/2 rb   ppss ber  vbg   TO/3 frag

turn straight back round and  head north past an old mill on   the right-hand
vb/2 ql/3     rp/3 rp/4  cc/2 vb/2 rp/4  in/3 at jj  nn   in/2 at  jj/2

side [3.1460] yeah south and  then straight back up   again with an old mill
nn            aff  rp/4  cc/2 rb   ql/3     rp/3 RP/2 rb    in/2 at jj  nn

on   the right and  you    +'re going to   pass on   the left-hand side of
in/2 at  nn/6  cc/2 ppss/2 ber  vbg   to/3 vb/2 in/2 at  jj/2      nn   in/2

the mill [1.5463] okay  and  then we   +'re going to   turn east [0.9900]
at  nn            aff/2 cc/2 rb   ppss ber  vbg   to/3 vb/2 rp/4

d--  not straight east slightly sort\_of northeast [1.4554] slightly slightly
frag not ql/3     rp/4 QL/2     QL/4     rp/4               RB/2     rb/2

yeah very slightly and  we   +'re going to   continue straight along erm
aff  ql/2 rb/2     cs/2 ppss ber  vbg   to/3 vb       ql/3     rp/2  fp

quite a  wee dis-- a  wee distance erm quite a  wee distance right we   +'re
dpr/3 at jj  frag  at jj  nn       fp  dpr/3 at jj  nn       aff/6 ppss ber

gonna continue along on   that course and  then we   +'re going to   turn
vbg   vb       rp/2  in/2 dt/4 nn     cc/2 rb   ppss ber  vbg   to/3 vb/2

north again [4.1689] and  immediat-- well  a  distance below that turning
rp/4  rb             cc/2 frag       aff/4 at nn       in/3  dt/4 jj/3

point there +'s   a  fenced meadow but  you    should be avoiding that by
nn    ex/4  bez/3 at jj     nn     cs/4 ppss/2 md     be vbg      pd/4 in/2

quite a  distance [1.8191] okay  so   we   +'ve turned and  we   +'re going
dpr/3 at nn                aff/2 cs/3 ppss hv   vbn/2  cs/2 ppss ber  vbg

up   north again [0.4305] continue straight up   north [0.4305] and  then
rp/2 rp/4  rb             vb       ql/3     rp/2 rp/4           cc/2 rb

we   +'re going to   turn to   the west on [0.8610] a  curvature right
ppss ber  vbg   to/3 vb/2 in/3 at  nn/4 IN/2        at nn        AFF/6

sort\_of "s"-bend [1.2765] and  immediately below that bend there is  an
jj/4     nn                cc/2 rb/2        in/3  dt/4 nn/2 ex/4  bez at

abandoned cottage [0.9253] and  we   +'re passing above the top  of   that
jj        nn               cs/2 ppss ber  vbg     in/2  at  nn/2 in/2 pd/4
```

```
we   +'re going to   continue in   that sort\_of "s" shape a  big wide
ppss ber  vbg   to/3 vb       in/2 dt/4 jj/4     nn  nn    at jj  jj

"s" [1.5535] and  on   the sort\_of mmhmm top   erm left of   that again below
nn           cc/2 in/2 at  jj/4     aff   jjt/2 fp  nn/6 in/2 pd/4 rb    in/3

it    there +'s   a  fenced meadow but  you    +'re passing on   the top
ppo/2 ex/4  bez/3 at jj     nn     cs/4 ppss/2 ber  vbg     in/2 at  nn/2

of   that okay [0.9991] right okay  we   +'ve gone from the abandoned cottage
in/2 pd/4 aff/2         aff/6 aff/2 ppss hv   vbn  in/2 at  jj        nn

right and  we   +'re on   the sort\_of "s" shape yeah [0.9198] right and  then
AFF/6 cc/2 ppss ber  in/2 at  jj/4     nn  nn    aff           aff/6 cc/2 rb

at   the top  of   the "s" we   +'re turning north [1.1845] okay  we   +'re
in/2 at  nn/2 in/2 at  nn  ppss ber  vbg/3   rp/4           aff/2 ppss ber

going straight due  north at   the top   there there +'s   a  west
vbg   ql/3     ql/2 rp/4  in/2 at  NN/2  pn?/4 ex/4  bez/3 at jj/4

lake [1.0063] which we   +'re going to   pass on   the south erm
nn            wpo/4 ppss ber  vbg   to/3 vb/2 in/2 at  jj/4  fp

southeast [1.8785] side and  we   +'re gonna do that in   a  curve almost a
JJ/4               nn   cc/2 ppss ber  vbg   do pd/4 in/2 at nn/2  qldt/3 at

half "u" shape [1.3290] yeah [0.9769] yeah [1.7651] the southeast and
nn/6 nn  nn             aff           aff           at  nn/4      cc/2

continue up   north slightly [1.2342] but  not quite to   the tip of   that
vb       rp/2 rp/4  rb/2              cc/4 not ql/3  in/3 at  nn  in/2 dt/4

lake [0.9569] and  then we   +'re going to   turn down ove-- above a  trick
nn            cc/2 rb   ppss ber  vbg   to/3 vb/2 rp/4 frag  in/2  at jj

point and  we   +'re going to   turn immediately to   your right and
nn    cc/2 ppss ber  vbg   to/3 vb/2 ql/2        in/3 ppg  nn/6  cc/2

straight down at   an angle of   forty-five [2.8604] okay  and  gonna
ql/3     rp/4 in/2 at nn    in/2 pn/3                aff/2 cc/2 vbg

continue that wee distance down and  at   the point [0.4503] at   the end
vb       dt/4 jj  nn       rp/4 cc/2 in/2 at  nn             in/2 at  nn/2

of   that it    should be near  to   the abandoned cottage where we   went
in/2 pd/4 pps/2 md     be JJ?/4 in/3 at  jj        nn      wrb   ppss vbd/2

past miles away but  if not just carry on   and  then continue down in   that
in/3 nns   rp/2 cs/4 cs not rb/3 vb    rp/2 cc/2 rb   vb       rp/4 in/2 dt/4
```

```
forty-five degree [0.9006] and  turn round by   a  monument on   the outside
cd/3       nn              cc/2 vb/2 rp/4  in/2 at nn       in/2 at  nn/3

of   the monument [1.6423] yeah and  then a  very slight turning up   again
in/2 at  nn                aff  cc/2 rb   at ap/2 jj     nn/3    in/2 rb

north sort\_of northwest [1.1762] very slight curve sort\_of very slight
rp/4  QL/4     rp/4               ap/2 jj     nn/2  rb?/4    ap/2 jj

"s"-shaped just  a  slight curve and  then gonna proceed up   north again
jj         dpr/3 at jj     nn/2  cc/2 rb   vbg   vb      rp/2 rp/4  rb

and  on   the right-hand side there +'s   a  nuclear test site before right
cc/2 in/2 at  jj/2       nn   ex/4  bez/3 at jj      nn   nn   cs/3   QL/6

before reaching the top  of   that northbound and  then you    +'re going
cs/3   vbg      at  nn/2 in/2 dt/4 nn/2       cc/2 rb   ppss/2 ber  vbg

to   turn back west and  above that there +'s   an east lake [1.7361]
to/3 vb/2 rp/3 rp/4 cc/2 in/2  pd/4 ex/4  bez/3 at jj/4 nn

yeah [1.5388] and  that +'s   the finish
aff           cc/2 pd/4 bez/3 at  nn/2
```

Chapter 3

Part of Speech Tagging

3.1 Introduction

The entire corpus was part-of-speech tagged using the Xerox HMM Tagger(version 0.9) [Cutting et al. 92]. A lexicon was used which contained all the words in the corpus, so the problem of unknown words was avoided. Word frgaments were handled by a rule which tagged each of them as 'FRAG', rather than attempting to expand word fragments into the intended complete word. An HMM tagging model was trained on the entire corpus, since we were interested in tagging this corpus as well as possible rather than evaluate the tagger. We did not use HMM models trained on other corpora (for example newspaper text) as we didnt want to make any assumptions about the kind of language used in the corpus. Nevertheless, it would be interesting to tag the MapTask Corpus corpus using such an HMM model and compare performance.

The following code was used to train and tag the corpus (see tagger/TAGGER.lisp) using the XEROX HMM Tagger.

```
cd tagger
lisp
(load "TAGGER.lisp")                      # need to type in 0 twice
(tag-trainer:train-on-files file-list)    # TRAINING
(dolist (f ff) (format t "Processing ~S~%" f)(my-tag-file f)) # TAGGING
```

One eighth of the MapTask Corpus(the first quadruple, consisting of 26942 words) has had its POS tags hand-corrected. According to this hand-corrected corpus, the first version of the tagger achieves an accuracy of 97.39% (i.e. there are 703 tagging errors).

David Elworthy [Elworthy 93], claims that a better measure of tagger accuracy is the proportion of *ambiguous* words which are given the correct tag, where by ambiguous we mean that the word has more than one possible POS tag. According to this measure, the first quadruple of the MapTask Corpus has 11673 ambiguous words and the first version of the tagger achieves an accuracy of 93.98% on these ambiguous words.

3.1.1 Rule-based post tagging

After a number of experiments had optimised the tagging performance on the hand-corrected section, there were still a number of errors which seemed to fall into a number of patterns. A rule-based POS tag editor program was written `tag-rewrite.perl` which rewrote POS tags if they occured in certain patterns. For example, the rule

vbd $\longrightarrow$ vbn / hvz pps _

means that the POS tag 'vbd' is changed to 'vbn' if it occurs after the POS tag sequence 'hvz pps', i.e. this would change 'have/hvz you/pps looked/vbd' to 'have/hvz you/pps looked/vbn'.

The rules used were:

vbd $\longrightarrow$ vbn / hvz pps _
vbd $\longrightarrow$ vbn / hv ppss _
pn $\longrightarrow$ cs / _ sent
to $\longrightarrow$ in / _ ppg
vbd $\longrightarrow$ jj / ppo _ sent
rp $\longrightarrow$ in / _ sent at
rp $\longrightarrow$ in / _ sent nn
rp $\longrightarrow$ ql / vb _ in
pd $\longrightarrow$ dt / _ sent nn
pn $\longrightarrow$ rp / nn _
pn $\longrightarrow$ rp / nns _
pn $\longrightarrow$ rp / pn _ sent
pn $\longrightarrow$ rp / vb _ sent
pn $\longrightarrow$ rp / vbn _ sent
pn $\longrightarrow$ rp / vbg _ sent
rb $\longrightarrow$ ql / _ rb sent
rb $\longrightarrow$ aff / _ vb
aff $\longrightarrow$ jj / bez _
nn $\longrightarrow$ aff / nn _ cc
nn $\longrightarrow$ aff / nn _ cs
nn $\longrightarrow$ aff / nn _ in
aff $\longrightarrow$ jj / cs _ sent
in $\longrightarrow$ rp / rp _ rb
rp $\longrightarrow$ ql / ql _ in
pr $\longrightarrow$ cs / pn _ ppss
in $\longrightarrow$ rp / nn _ in

The rules were found by looking for common patterns of tags around error tags.

3.2 Hand-Correction

The entire POS tagged corpus was then hand-corrected and the automatically tagged corpus evaluated against the hand-corrected version. The process of hand-correction was made much easier by the existance of the automatically tagged corpus as a starting point (although its existance may

have biased the correction phase in ambiguous cases), and by a specialised POS tag editor program written by Henry Thompson. This program `tagdisamb/disamb` uses the LTNSL SGML API to read/write the SGML version of the tagged corpus, and Python and TCL/Tk for its user interface. The editor only allows one to modify POS tags, and ensures that all new POS tags are allowed by the lexicon.

3.3 Evaluation

The evaluation of the automatic corpus against the hand-corrected version gave the results on the complete corpus:

Error rate	= 2.27%
Ambiguous error rate	= 5.04%

Let us define a *stupid tagger* as being a tagger which always assigns the same tag to each word, regardless of context. The tag which it assigns to a word is the most probable tag for this word, as occurring in some tagged corpus. If we train a *stupid tagger* on the hand-corrected quadruple of the MapTask Corpus, and then use the *stupid tagger* to tag this quadruple, we find a tagging accuracy of 91.62% (i.e. there are 2257 tagging errors out of 26942 words). The corresponding figure for ambiguous words only is much lower at 80.66%.

What is the average POS ambiguity in the MapTask Corpus, i.e. how difficult is the job of tagging it? One commonly used measure of this is the average number of POS tags per word, weighted by the frequency of occurance of the words in a test corpus. If we calculate this average POS ambiguity over the MapTask Corpus, we get a figure of 1.87 tags per word. For comparison [Feldweg 95] claims a figure of 1.51 tags per word for a two million word sample of German newspaper text. The figure for the British National Corpus [BNC 97] gives a much higher figure of 3.96.

The accuracy of the POS tagger on this corpus is not particularly good in comparison to other reported figures. This is probably due to the rather small size of the MapTask Corpuscorpus as compared to the BNC or Wall Street Journal corpora. The rather disjointed nature of the spontaneous speech may also have been a contributatory factor. However, it was good enough for our purposes, as it meant that the hand correction phase was feasible (about three person weeks).

What did we learn from this exercise? Firstly, a lot of work was needed to fine tune the initial transition probabilities in order to get reasonable results (the Xerox tagger uses an initial set of hints about likely tag transitions in the corpus). We didn't use the latest tagging technology, e.g. Claws or later versions of HMM taggers, e.g. Mikheev, Cutting, Gilbert, some of which require less work on initial hints.

Secondly, we didnt use a standard, written language HMM or tag set (i.e. trained on a large corpus of written language). This was deliberate, as we didn't want to make any assumptions about the similarity of written language and spontaneous speech. For example, we found that we had to modify the tag set. The disadvantage however was that the corpus was probably too small for use as training data.

The hand correction of the POS tags was done by a single person and not a team, so we have no idea of how reliable it was.

Finally, we didnt investigate whether/to what extent disfluency disrupted the sequence of POS tags (cf Heeman [Heeman 94]). In practise, however, we found that disfluency did not introduce tagging errors.

Chapter 4

Parsing

This chapter describes the parsing algorithm used in the SDP system. Chapter 5 describes how to use the parser, chapter 6 describes the grammar and lexicon used to parse the MapTask Corpus corpus, and chapter 7 describes the SGML format of the parsed corpus.

4.1 Introduction

The SDP parser is implemented in SICStus Prolog. This means that it is relatively slow, but meant that it was simpler to develop and experiment with alternative implementations.

The parser uses phrase structure rules (extended by Kleene iteration operations) to describe the grammar. Grammatical constituents are described by a major category e.g. NP and a list of attribute/value features. An example of such a rule is:

```
rule(main(s03),
     s([vform=fin,inv=not_inv,dum=D|_]),
     [ np([num=N,case=subj,pers=P,dum=D|_]),
       vp([vform=fin,num=N,pers=P|_]) ] ).
```

Each rule has an identifier, a description of the left hand side of the rule, and a list of the daughters (the right hand side of the rule). So the above rule says that an S can consist of a NP and a VP. Capitalised words in the rule are Prolog variables (e.g. N). Unification is used to ensure that the num and pers attributes of the NP and the VP are the same.

The parser uses a layered grammar, it is bottom up and deterministic.

- Layered grammar: The grammar is divided into an ordered number of layers of rules. For each layer in order, each rule in the layer (in order) is applied to the input utterance at each position. Once no more rules in this layer can fire, the parser moves on to the next layer. The layers in the MapTask Corpus grammar consist of a number of basic layers which find non-recursive phrases (e.g. determiners, adjective phrases, and simplex noun phrases); a higher level layer

which describes recursive phrase structure rules e.g. NP → NP PP; a final layer which describes utterances in terms of sequences of phrases and clauses; and a layer for handling disfluency which is handled specially by the parser.

- Disfluency: The grammar contains a small number of general rules for parsing disfluent phrases and utterances. The theory behind this approach is based on the idea that self-repairs can be described in a similar way to conjunctions and is described in detail in [McKelvie 98a]. The disfluency (meta-)rules are implemented as another layer of the grammar, but this layer is checked to see if any rules fire immediately after any other rule in one of the other layers has fired. So, if this parser was to be seen as a model of the human sentence processor (not one of its declared aims), then this could be seen as postulating a separate process that looks out for disfluencies.

- Bottom up: The decision to use a bottom-up parser was driven by two factors. Firstly, we wanted to make as few initial assumptions about the structure of these spoken dialogues as possible, i.e. we didn't want to assume that there was a common syntactic description of all utterances at the onset. In fact we didn't want to assume that we could parse all utterances into complete sentences. This approach seemed to match the somewhat fragmentary form of some of the corpus utterances. Secondly, a bottom up parser best fitted the incremental approach taken to grammar development, where e.g. rules for simplex noun phrases were developed first.

- Deterministic: The decision to build a deterministic parser was taken to avoid the issue of selecting the correct parse from among several and in general to try to avoid the issue of ambiguity as far as possible. This is difficult, although less difficult than it would be for written language. Problems with ambiguity were avoided by (a) not making distinctions that introduced ambiguity (b) only committing to parses when we were fairly sure that we could make the correct choice. Various techniques were used. Prolog constraints were used to avoid back-tracking, for example the selection of sub categorisation frames for verbs was delayed until phrases to the right of the verb had been detected. Parses were left fragmentary when unsure. Right context checking was introduced into the rule formalism.

- Use of right context: Some of the rules of the grammar include a fourth field which describes the right context which must exist if the rule is to fire, for example:

```
rule(main(abort(s02)),
     s([abort=abort,conj=C|_]),
     [ cs([conj=C|_]) ],
     [ RC ]):-
     when(nonvar(RC),(RC=ed(_))).
```

  which says that if the parser sees a sentence conjunction followed by an editing phrase, then it will parse the conjunction as an aborted sentence, but won't include the editing phrase in the aborted sentence. The use of this right context is mainly used as a way of making the deterministic parser more palatable by preventing rules firing if they have a certain right context. The use of similar left context has not yet seemed to be necessary.

4.2 Chart implementation

A chart parsing version of the parser was developed which used the same grammar but which produces all possible parses for a given utterance. This version has not been extensively tested and no means of ranking parses has been developed.

The introduction of chart parsing has a number of implications for the parser. Firstly, if we store chart edges as Prolog predicates in the database (as is commonly done), then we lose the under-specified nature of our edges - we have to spell it out. Secondly, in the chart parser version, the disfluency rules produce a lot of stupid edges e.g. N[NP [NP(ab) the] [NP cat]], so that the nature of the disfluency handling rules need to be rethought. Possibly to the extent of no longer needing them. This is rather disturbing to the analysis of disfluencies presented in [McKelvie 98a], although we still think that there is a gain in descriptive clarity in describing disfluencies as in some way parallel to coordination.

4.3 Evaluation

The bottom up approach that we used was good for grammar development and evaluation. For example, the development of a lexicon including sub categorisation frames for verbs was done after the basic grammar had been developed. In general, parsing based largely on POS tags rather than on words has the advantage that a complete lexicon is not needed for parsing.

The approach taken here is similar to the 'chunking' approach described by Steve Abney [Abney 96], but differs in that we attempted to go beyond chunking, by attempting to parse some recursive phrasal constituents where possible. Finite state technology was not used although it could have been, at least for the lower layers of the grammar. Finite state technology would have been more efficient.

The parser treated disfluency as a syntactic process and not just as random noise or as parser failure. It achieved reasonable accuracy although by no means perfect.

There is a difficulty in deterministic parsers where the parser behaviour depends on rule orderings. As the grammar grows, modifying it become more difficult, due to unforseen interactions between the rules. Splitting the grammar into layers helps to some extent. This is however a problem which affects all grammar development, see for example [Moshier 97] for a critique of HPSG along these lines.

It could be argued that the grammar developed is somewhat ad hoc, in that it is not clearly based on a grammatical theory such as HPSG. For example, it doesn't follow Jackendorff's X-bar theory, and has rules such as NP $\Rightarrow$ NP, PP rather than N' $\Rightarrow$ N', PP. This analysis is however arguably correct for the chunking style of parsing which is advocated here.

Future work should compare the approach taken here with one which clearly separates FSA chunking from recursive phrasal parsing attachment decisions. This would lead to a modular three stage parsing algorithm: viz, a part of speech tagger, a finite state chunker, and a probabilistic non-deterministic phrasal attacher. On the basis of this project, it seems that disambiguation (i.e. phrase attachment) of complex, spontaneous speech probably requires some understanding of what the utterance means.

Chapter 5

User Manual

5.1 Introduction

The Spoken Dialogue Parser (SDP) is a parser designed for bottom-up parsing of spoken dialogues such as the Maptask or the Trains corpora.

5.2 To Run SDP

1. Start Prolog

   SDP runs under SICStus Prolog (tested under versions 2.1 or 3). It makes use of the delay (coroutining) mechanisms of SICStus Prolog. When processing large files, it is advisable to run this on a fast machine.

2. Load the parser file (optionally compiling it).

   ```
   :- ['~dmck/MT/grammar.pl'].
   :- compile('~dmck/MT/grammar.pl').
   ```

3. To parse a sentence

   ```
   :- parse([the/at,cat/nn,sat/vbd,on/in,the/at,mat/nn,'.'/sent]).
   ```

   which will give something like (depending on the grammar and the output format parameters)

   ```
   s(vform=fin)
      np(num=sing, pers=3, case=subj)
         detp()
            centdet() the/at
         n2(num=sing, pers=3, case=subj)
            n(num=sing, pers=3, case=subj) cat/nn
   ```

```
   vp(vform=fin, num=sing, pers=3)
      v(vform=fin, subcat=3, num=sing, pers=3) sat/vbd
      pp(prep=on)
         p(prep=on) on/in
         np(num=sing, pers=3, case=obj)
            detp()
               centdet() the/at
            n2(num=sing, pers=3, case=obj)
               n(num=sing, pers=3, case=obj) mat/nn
sentmark() ./sent
>>>ok [s,sentmark]

-----------------------------------------------
```

4. To parse a file of sentences:

   ```
   :- t(FileName).
   ```

   This predicate is a non-interactive version of the parser. This assumes that FileName is the name of a file (as a Prolog atom). The input file will be parsed and the parsed sentences will be written to a file of with a similar name, based on the following ordered rules:

   - Non-SGML input
     - *.retag → *.syn
     - * → *.syn
   - SGML input
     - *.tag.sgm → *.syn.sgm
     - *.sgm → *.syn.sgm
     - * → *.syn

   The format of the input file depends on the setting of the `read_sgml` predicate. If `read_sgml(false)` then it is a sequence of Word/Tag pairs, one per line. Words and tags can be arbitrary strings (not containing /) which will be converted to Prolog atoms. The file is separated into sentences by Words with tag 'sent'. The file must also start with a 'sent' tagged word. If `read_sgml(true)` then the input file is an SGML file ...

5. `:- g(FileName).`

   is the same as above, but it is interactive and writes parses to the terminal not to a file. After each sentence, it allows the user the choice of stopping the parsing, redoing the parse (after editing the grammar file) and toggling the tracing of rules.

6. `:- test.`

   This predicate calls 'g/1' on the test file defined by the test_file/1 predicate.

5.3 Modifying the behaviour of the Parser

The behaviour of the parser can be controlled by a number of Prolog predicates, which are in grammar.pl and can be changed.

Modify how parse trees are printed.

```
print_attributes(yes/no).
```

Controls the printing of attributes when printing parse trees. Default value is yes.

```
print_all_nodes(false/true).
```

If false, then we dont print a non-terminal node if it (a) is an only child and (b) has only one non-terminal child. If true then all non-terminal nodes are printed. Default value is false.

```
print_trivial_sentences(false/true).
```

Controls whether or not 'trivial' sentences are printed. Default value is false. A 'trivial' utterance is one that contains only words tagged with sent, noise, aff, pau, fp, or uh (or as defined by the predicate 'trivial_sentence/1').

```
print_disfluencies(true/false).
```

If true then disfluent utterances are printed completely. If false then disfluent parse trees are pruned to remove disfluencies. Default value is true.

```
remove_commas(true/false).
```

If true then remove commas from text before parsing. Default is false.

```
add_rule_ids(true/false).
```

If true then all grammatical categories have a new attribute 'rule' showing which rule created them. If true then there a certain amount of hackery and the rule attribute is NOT unified in the normal way, it is ignored. So that np([case=obj,rule=r1—_]) and np([case=obj,rule=r2—_]) DO unify. Default value is true.

```
read_sgml(true/false).
```

If true read from sgml file. Default value is true.

```
write_sgml(true).
```

If true write output in sgml format. Default value is true.

```
output_parse(full/phrasal).
```

If full then all parse nodes are printed, if phrasal only phrases are printed. Default value is phrasal.

```
add_constituent_durations(true/false).
```

If true (the default) then two new attributes are added to each grammatical category, i.e. 'dur' and 'start'. These are calculated from the 'dur' and 'start' attributes of the children.

Modify files

```
test_file(TestFile).
```

TestFile is a file containing example sentences which is read by the 'test/0' predicate.

```
grammar_file(GrammarFile).
```

GrammarFile is the name of the prolog source file containing the rules of the grammar.

5.4 Format of SGML input/output files

When write_sgml(true) then the output file is written as SGML conforming to the PARSEDCHANNEL or PHRASALCHANNEL DTDs, described in chapter ???.

5.5 Format of the grammar rule file

The file of grammar rules is a Prolog source file. It must contain definitions for the following predicates:

tracing(RuleId,TracingInfo) This predicate says for each rule identifier whether or not it will be traced. Possible values for TracingInfo are 'no', 'yes' or 'full'. If this is 'no' the rule will not be traced, if it is 'yes' the rule will be traced after it has successfully fired, if 'full' it will be traced whenever it is being considered for firing. This predicate should be defined as dynamic. For example, the default definition for tracing/2 is

```
:- dynamic tracing/2.
tracing(_,no)
```

rule(LHS,List) Lexical introduction rules are represented by the rule/2 predicate. LHS is a grammatical category and List is a list of Word/Tag pairs, for example:

```
rule(adj([deg=pos|_]),[_/jj]).
```

says that all words tagged with the JJ tag are converted into pre-terminal nodes of category ADJ with the DEG attribute set to POS.

rule_groups(List_of_rule_groups) The order of application of groups of rules is specified by the rule_groups/1 predicate.

rule/3 or rule/4 Phrase structure rules are represented by rule/3 or rule/4.

```
rule(RuleId,LHS,RHS)
```

is equivalent to rule(RuleId,LHS,RHS,[]), i.e. to rule/4 with empty right context. Rule/3 predicates are converted to rule/4 predicates by some user:term_expansion/2 rules, so that rule/3 and rule/4 predicates can be interspersed in the grammar rule file.

```
rule(RuleId,LHS,RHS,RC)
```

means that occurances of RHS are replaced by LHS, provided that RC occurs immediately to the right of RHS in the sequence.

For example, the rule for declarative sentences:

```
rule(main(s03),s([vform=fin,inv=f,dum=D|_]),
     [np([num=N,case=subj,pers=P,dum=D|_]),
      vp([vform=fin,num=N,pers=P|_])]).
```

says that a sentence (s) consists of a subject noun phrase (np) and a finite verb phrase (vp).

ruledefault(Cat) Default values for attributes can be specified with the ruledefault/1 predicate. For example,

```
ruledefault(np([abort=f|_])).
```

which says that all noun phrases which are created during parsing will get the value 'f' for their abort attribute, unless they already have a different value for this attribute.

default(Cat) Final default values for attributes can be specified with the default/1 predicate, for example

```
default(vp([vform=fin|_])).
```

which says that, at the end of parsing, any verb phrase with an unspecified value for the 'vform' attribute, will get a value of 'fin'.

dont_want_to_print(Attribute=Value) Control over the amount of attribute information is printed is given by the dont_want_to_print/1 predicate. Any Attribute-Value pair which unifies with this will not be printed. This is useful for attribute/value pairs which are SGML default values.

5.6 Chart Parser

The parser can also be run as a chart parser which outputs all maximal spanning parses. This can be run by:

```
['~dmck/MT/grammar.pl', '~dmck/MT/chart.pl'].
```

and continuing as described above.

Chapter 6

Grammar for MapTask Corpus

6.1 Introduction

This chapter gives a complete listing of the grammar and the lexicon used to parse the corpus. These are in the form of Prolog predicates.

6.2 Lexical introduction Rules

The first step in the parsing process is to move from tagged words to grammatical categories for the words (or in some cases sequences of words). This is handled by the `rule/2` predicate, which are of the form rule(Category,List of tagged words). For example, the clause

```
rule(n([num=sing,pers=3,sem=measure|_]),[inch/nn]).
```

means that the word 'inch' when tagged as NN, will be parsed into the category N with attributes num=sing, pers=3, and sem=measure.

6.2.1 Lexical Disfluencies

1. Repeated words, except for tag=aff or ql, are assumed to be a disfluency.

   ```
   rule(X,[Word/_,Word/Tag]):-
           rule(X,[Word/Tag]), not(X=aff(_)), not(X=ql(_)).
   ```

2. Modified repeated words. This rule should perhaps be extended to cover other clitic verbs as well.

   ```
   rule(X,['+''s'/Tag,is/Tag]):- rule(X,[is/Tag]).
   ```

3. Aborted words Case 1: where fragment is replaced by just the next word.

```
rule(X,[Frag/frag,Word/Tag]):-
        when((nonvar(Frag),nonvar(Word)),
             ( name(Frag,FragName), append(PartName,"--",FragName),
               name(Word,WordName), append(PartName,_,WordName),
               rule(X,[Word/Tag]))).
```

4. Aborted words Case 2: where there is a pause between fragment and replacement word.

```
rule(X,[Frag/frag,'...'/pau,Word/Tag]):-
        when((nonvar(Frag),nonvar(Word)),
             ( name(Frag,FragName), append(PartName,"--",FragName),
               name(Word,WordName), append(PartName,_,WordName),
               rule(X,[Word/Tag]))).
```

6.2.2 Normal Lexical Introduction Rules

Adjectives

```
rule(adj([deg=pos|_]),[_/jj]).
rule(adj([deg=comp|_]),[_/jjr]).
rule(adj([deg=sup|_]),[_/jjt]).
```

Determiners

```
rule(qldt(_),[_/qldt]).
rule(postdet(_),[couple/nn,of/in]).
rule(postdet(_),[_/ap]).
rule(postdet(_),[_/od]).
rule(cd(_),[_/cd]).
rule(predet(_),[_/dpr]).
rule(centdet(_),[_/at]).
rule(centdet(_),[_/dt]).
rule(centdet(_),[_/ppg]).
rule(centdet([wh=t|_]),[_/wdt]).
rule(gen(_),[_/gen]).
```

Nouns and pronouns

Nouns which represent units of measurement are marked especially, since measure noun phrases modify prepositional phrases, and behave differently from non-measure noun phrases.

```
rule(nprop([pers=3,type=prop,sem=any|_]),[_/np]).
rule(n([num=sing,pers=3,sem=measure|_]),[inch/nn]).
rule(n([num=plur,pers=3,sem=measure|_]),[inches/nns]).
```

```
rule(n([num=sing,pers=3,sem=measure|_]),[centimetre/nn]).
rule(n([num=plur,pers=3,sem=measure|_]),[centimetres/nns]).
rule(n([num=sing,pers=3,sem=measure|_]),[bit/nn]).
rule(n([num=sing,pers=3,sem=measure|_]),[half/nn]).
rule(n([num=sing,pers=3,sem=measure|_]),[quarter/nn]).
rule(n([num=plur,pers=3,sem=measure|_]),[quarters/nns]).
rule(n([num=plur,pers=3,sem=measure|_]),[degrees/nns]).
rule(n([num=sing,pers=3,sem=measure|_]),[couple/nn]).
rule(n([num=sing,pers=3,sem=measure|_]),[lot/nn]).

rule(uh(_),[oh/uh,shit/nn]).
rule(uh(_),[oh/uh,heck/nn]).

rule(n([num=sing,pers=3,sem=any|_]),[_/nn]).
rule(n([num=plur,pers=3,sem=any|_]),[_/nns]).

rule(prmod(_),[else/rb]).

rule(pro([num=sing,case=subj,pers=3,sem=any,type=pro|_]),      [he/pps]).
rule(pro([num=sing,case=subj,pers=3,sem=any,dum=t,type=pro|_]),[it/pps]).
rule(pro([num=plur,case=subj,pers=3,sem=any,type=pro|_]),      [they/pps]).

rule(pro([num=sing,case=subj,pers=1,sem=any,type=pro|_]),[i/ppss]).
rule(pro([num=sing,case=subj,pers=1,sem=any,type=pro|_]),['I'/ppss]).
rule(pro([num=plur,case=subj,pers=1,sem=any,type=pro|_]),[we/ppss]).
rule(pro([case=subj,pers=2,sem=any,type=pro|_]),         [you/ppss]).
rule(pro([case=subj,pers=3,num=plur,sem=any,type=pro|_]),[they/ppss]).

rule(pro([case=subj,pers=3,sem=any,dum=t,type=pro|_]),[_/ex]).

rule(pro([case=subj,pers=3,sem=any,type=pro,wh=t|_]),[_/wps]).

rule(pro([num=sing,pers=3,sem=any,type=pro|_]),[_/pd]).

rule(wpo([wh=t|_]),[_/wpo]).

rule(pro([case=obj,sem=any,type=pro|_]),[_/ppo]).

rule(pro([case=obj,sem=any,type=pro|_]),[_/ppl]).
rule(pro([sem=any,type=pro|_]),[_/ppg2]).
/* _/pn are not type=pro as they allow PP modifiers */
/* They are unspecified for sem as we want to allow eg "one and a half" */
/*rule(pro([sem=any|_]),[_/pn]).*/
rule(pro(_),[_/pn]).

rule(p([prep=X|_]),[X/in]).
```

Verbs

Subcategorisation information for verbs is retrieved from the `lexicon/3` predicate. Verb subcat information was gathered from an analysis of the corpus. A default set of allowed subcat frames is allowed for unseen verbs.

```
rule(v([vform=fin,num=sing,pers=3,clitic=C,subcat=S|_]),[V/bez]):-
        lexicon(be,vb,S),
        ( V='+''s', C=t ; not(V='+''s'), C=f ).
rule(v([vform=fin,num=sing,pers=1,clitic=C,subcat=S|_]),[V/bem]):-
        lexicon(be,vb,S),
        ( V='+''m', C=t ; not(V='+''m'), C=f ).
rule(v([vform=fin,num=N,   pers=P,clitic=C,subcat=S|_]),[V/ber]):-
    lexicon(be,vb,S),
    ( V='+''re', C=t ; not(V='+''re'), C=f ),
    when(nonvar(N), ( N=plur ; ( N=sing, P=2 ))).
rule(v([vform=bse,aux=f,subcat=S|_]),[_/be]):-
        lexicon(be,vb,S).

rule(v([vform=fin,aux=t,mod=t,subcat=S|_]),[_/md]):-
        when(ground(S),member(S,[vp(bse),intrans])).

rule(v([vform=VF,aux=f,num=N,pers=P,subcat=S|_]),[Word/vb]):-
     lexicon(Word,vb,S),
     when(nonvar(VF), ( VF=bse ; VF=fin)),
     when(nonvar(N), ( VF=fin, ( N= plur ; N=sing, not(P=3)))).

rule(v([vform=fin,aux=f,num=sing,pers=3,subcat=S|_]),[Word/vbz]):-
        lexicon(Word,vb,S).
rule(v([vform=fin,aux=f,subcat=S,tns=past|_]),[Word/vbd]):-
        lexicon(Word,vb,S).
rule(v([vform=ing,aux=f,subcat=S|_]),[Word/vbg]):-
        lexicon(Word,vb,S).
rule(v([vform=en, aux=f,subcat=S|_]),[Word/vbn]):-
        lexicon(Word,vb,S).
rule(v([vform=to, aux=f,subcat=vp(bse)|_]),[_/to]).
rule(v([vform=VF,aux=A,num=N,pers=P,subcat=S|_]),[_/do]):-
     lexicon(do,vb,S),
     when(nonvar(VF), ( VF=bse, A=f ; VF=fin)),
     when(nonvar(N), ( VF=fin, ( N= plur ; N=sing, not(P=3)))).
rule(v([vform=fin,subcat=S|_]),[_/doz]):-
        lexicon(do,vb,S).
rule(v([vform=VF,aux=A,num=N,pers=P,subcat=S,clitic=C|_]),[V/hv]):-
     lexicon(have,vb,S),
     ( V='+''d',  C=t, VF=fin
     ; V='+''ve', C=t, VF=fin
     ; not(V='+''d'),not(V='+''ve'), C=f ),
     when(nonvar(VF), ( VF=bse, A=f ; VF=fin)),
     when(nonvar(N), ( VF=fin, ( N= plur ; N=sing, not(P=3)))).
```

```
rule(v([vform=fin,subcat=S,clitic=C|_]),[V/hvz]):-
     ( V='+''d', C=t ; not(V='+''d'), C=f ),
     lexicon(have,vb,S).
```

Other Minor categories

```
rule(qlp(_),[_/qlp]).

rule(not(_),[_/not]).

rule(ql([ql=W|_]),[W/ql]).

rule(cc([conj=and_then|_]),[and/cc,then/rb]).
rule(cc([conj=X|_]),[X/cc]).

rule(cs([conj=X|_]),[X/cs]).

rule(sentmark(_),[_/sent]).
rule(sentmark(_),[_/cm]).

rule(adv([wh=f,deg=pos|_]),[_/rb]).
rule(adv([wh=f,deg=comp|_]),[_/rbr]).
rule(rpt([prep=X|_]),[X/rp]).

rule(wrb(_),[_/wrb]).
rule(wql(_),[_/wql]).

rule(relpro(_),[_/pr]).

rule(aff(_),[_/aff]).
rule(pau(_),[_/pau]).
rule(fp(_),[_/fp]).
rule(frag(_),[_/frag]).
rule(uh(_),[_/uh]).
rule(noise(_),[_/noi]).
```

6.3 Sequencing of Grammar rules

Rule groups are fired in the following order.

```
rule_groups([ed(_),detp(_),ap(_),advp(_),np(_),rp(_),main(_),utt(_)]).
```

6.4 Disfluency Rules

1. Ignore edit signals.
   Rule is X → X ed with context Y ... If we can fire a rule Z → X Y ...
   We dont want to fire this with rules which introduce aborts

```
rule(disf(RuleId),X,[X,ed(_),aff(_)?],RC1):-
        not(RuleId=ed(_)),
        not(RuleId=disf(_)),
        not(RuleId=main(abort(_))),
        rule(RuleId,_,RHS,RC),
        make_disf_rule(RHS,RC,X,RC1).
/*      when(nonvar(X),(copy_term(X,X1),make_disf_rule(RHS,RC,X1,RC1))). */
/* without this change means we will have problems with intrans vp */
```

2. X → X[abort=t] X

```
rule(disf(2),Y,[X,aff(_)?,Y]):-
        when((nonvar(X),nonvar(Y)),
             ( X=.. [Cat,A1],
               Y=.. [Cat,A2],
               unify_attr_lists(A1,[abort=t|_]),
               ( Cat=np,!,
                 unify_attr_lists(A1,[type=Type,case=Case|_]),
                 unify_attr_lists(A2,[type=Type,case=Case|_])
               ; true ))).
```

3. X → X ed X

```
rule(disf(3),Y,[X,ed(_),aff(_)?,Y]):-
        when((nonvar(X),nonvar(Y)),
             ( X=.. [Cat,A1],
               Y=.. [Cat,A2],
               ( Cat=np,!,
                 unify_attr_lists(A1,[type=Type|_]),
                 unify_attr_lists(A2,[type=Type|_])
               ; true ))).
```

6.5 Edit Signals

```
rule(ed(1),ed1(_),[fp(_)]).
rule(ed(2),ed1(_),[fg(_)]).
rule(ed(3),ed1(_),[pau(_)]).
rule(ed(4),ed1(_),[noise(_)]).
rule(ed(5),ed1(_),[uh(_)]).

rule(ed(a),ed(_),[frag(_)+,ed1(_)*]).
rule(ed(b),ed(_),[ed1(_)+]).
```

6.6 Adjective phrases

```
rule(ap(a),a2(X),[not(_)?,ql(_)*,adj(_)*,adj(X),qlp(_)]).
rule(ap(a),a2(X),[not(_)?,ql(_)*,adj(_)*,adj(X)]).

rule(main(ap01),a2(X),[ql([ql=as|_]),ap(X),cs([conj=as|_]),np(_)]).

/* any bigger than that */
rule(main(ap02),a2(X),[detp(_),ap(X),cs([conj=than|_]),np(_)]):-
        X=[deg=comp|_].

/* bigger than that */
rule(main(ap03),a2(X),[ap(X),cs([conj=than|_]),np(_)]):-
        X=[deg=comp|_].

rule(main(ap12),ap([wh=t|X]),[wql(_),a2([wh=f|X])]).
rule(main(ap11),ap(X),[a2(X)]).

/* e.g. level with it */
rule(main(ap13),ap(X),[ap(X),pp(_)]).

/* e.g. difficult to explain */
rule(main(ap14),ap(X),[ap(X),vp([vform=to|_])]).

/* e.g. three inches wide */
rule(main(ap15),ap(X),[np([sem=measure|_]),ap(X)]).
```

6.7 Adverbial Phrases

```
rule(advp(a),adv1(X),[ql(_)*,adv(_)*,adv(X),qlp(_)]).
rule(advp(a),adv1(X),[ql(_)*,adv(_)*,adv(X)]).
rule(advp(adv1_03a),adv1([wh=t|X]),[wql(_),adv1([wh=f|X])]).
rule(advp(adv1_03a),adv1([wh=t|X]),[wrb(X)]).

rule(main(adv1_04),adv1(X),[ql([ql=as|_]),advp(X),cs([conj=as|_]),np(_)]).
rule(main(adv1_05),adv1(X),[cs([conj=as|_]),advp(X),cs([conj=as|_]),np(_)]).
/* further than that */
rule(main(ap03),adv1(X),[advp(X),cs([conj=than|_]),np(_)]):-
        X=[deg=comp|_].

rule(main(advp01),advp(X),[not(_),adv1(X)]).
rule(main(advp02),advp(X),[adv1(X)]).
rule(main(advp03),advp(_),[advp(_),cc(_),advp(_)]).
/* A hack to catch tagger errors */
rule(main(advp05),advp(X),[advp(X),np([sem=measure|_])]).
```

6.8 Determiners

```
/* two to three */
rule(detp(dp01),postdet(_),[cd(_), v([vform=to|_]), cd(_)]).
rule(detp(dp01),postdet(_),[cd(_), p([prep=to|_]), cd(_)]).
/* two three    */
rule(detp(dp02),postdet(_),[cd(_)?,cd(_)]).

rule(detp(dp04),postdg(_),[ql(_)*,qldt(_)?,postdet(_)]).

rule(detp(dp06),detp1(X),[predet(_),centdet(X)]).
rule(detp(dp06),detp1(X),[predet(X)]).
rule(detp(dp08),detp1(X),[centdet(X)]).
rule(detp(dp09),detp1(X),[detp1(_)?,postdg(X)]).

rule(detp(a),detp0(X),[not(_)?,ql(_)*,qldt(_)*,detp1(X)]).

rule(detp(qldtab),postdg([abort=t|_]),[qldt(_)]).

rule(detp(dp14),detp(_),[detp0(_),cc(_),detp0(_)]).
rule(detp(dp15),detp(X),[detp0(X)]).

rule(main(dp16),detp(_),[np(_),gen(_)]).
```

6.9 Noun Phrases

```
rule(np(n1_1),n1(X),[n(_)*,n(X)]).
rule(main(n2_1),n2(X),[ap(_)?,n1(X)]).
rule(main(n2_3),n2(X),[n2(X),cc(_),n2(X)]).

rule(main(np01),np([wh=W|X]),[detp([wh=W|_]),n2([wh=f|X])]).
/* Zero det in plural NP */
rule(main(np02),np(X),[n2(X)]):- X=[num=plur|_].
/* Zero det in singular NP is sometimes possible (over general?) */
rule(main(np6catch),np(X),[n2(X)]):- X=[num=sing|_].

rule(main(np03),np(X),[nprop(_)?,nprop(X)]).
rule(main(np05),np(X),[pro(X),prmod(_)]).
rule(main(np05),np(X),[pro(X)]).
rule(main(np11),np([num=plur|X]),
        [np([num=_|X]),cc([conj=and|_]),np([num=_|X])],[RC]):-
    not(RC=v(_)), not(RC=vp(_)).
/* "the engine E1" */
rule(main(np11a),np(X),[np(Y),np(X)]):- X=[type=prop|_], Y=[type=norm|_].

rule(main(np12),np(X),[np(X),cc([conj=C|_]),np(X)],[RC]):-
```

```
    not(C=and), not(RC=v(_)), not(RC=vp(_)).
```

Attach 'of' and 'than' PPs quickly

```
rule(main(np13),np(X),[np(X),pp([prep=of|_])]).
rule(main(np13),np(X),[np(X),pp([prep=than|_])]).
```

Attach other PPs more slowly (if followed by a sentence)

```
rule(main(np14),np(X),[np(X),pp([conj=none|_])],[RC]):-
        X=[sem=any,type=norm|_],
        member(RC,[s(_),vp(_),sentmark(_),aff(_),advp(_)]).
```

6.9.1 Relative noun phrases

(1) head NP is subject of VP e.g. "the man who liked cakes"

```
rule(main(npr01),np([num=N,pers=P|X]),
     [np([num=N,pers=P|X]), relpro(_), vp([num=N,pers=P|_])]).
```

(2) head NP is object of S although case is not necessarily obj e.g. "what we need"

```
rule(main(npr02),np(X),[wpo(X),s([inv=f|_])]).
/* rule(main(npr02),np(X),[np(X),s([inv=f|_])]):- X=[wh=t|_]. */
```

(3) head NP is object of S although case is not necessarily obj e.g. "a lake which we pass" e.g. "the buffalo that is black"

```
rule(main(npr03),np(X),[np(X),relpro(_),s(_)]).
```

(4) Adverbial e.g. "where we have to go"

```
rule(main(npr04),np(_),[advp([wh=t|_]),s([vform=_,inv=f|_])]).
```

(5) "the way you +ll be going" - missing relpro

```
rule(main(npr6),np(X),[np(X),s([vform=fin,inv=f,dum=f,conj=none|_])]):-
        X=[type=norm|_].
```

(6) handle "what about the pan?"

```
rule(main(npr6),np(X),[wpo(X),pp(_)]).
/* rule(main(npr6),np(X),[np(X),pp(_)]):- X=[wh=t|_]. */
```

(7) "A man called horse" Rule not used as it causes problems with questions

```
/* rule(main(npr7),np(X),[np(X),vp([vform=en|_])]). */
```

(8) "what to do now"

```
rule(main(npr08),np(X),[wpo(X),vp([vform=to|_])]).
/* rule(main(npr08),np(X),[np(X),vp([vform=to|_])]):- X=[wh=t|_].*/
```

(9) "the best thing to do now"

```
rule(main(npr09),np(X),[np(X),vp([vform=to|_])]).
```

(10) A hack - because tagger is not good at tagging 'about'

```
rule(main(hack1),np([sem=measure|X]),[p([prep=about|_]),np([sem=measure|X])]).
```

(11) Force wpo(_) to become np([wh=t|_])

```
rule(main(npwpo),np(X),[wpo(X)],[RC]):-
        when(nonvar(RC),member(RC,[sentmark(_)])).
```

6.9.2 Aborted Noun phrases

```
rule(main(abort(np01)),np([abort=t|_]),[detp(_)],[RC]):-
        not(RC=ap(_)),not(RC=n2(_)).
rule(main(abort(np02)),np([abort=t|_]),[detp(_),ap(_)],[RC]):-
        not(RC=n1(_)).
/* NB cc(_) implies that we dont allow AP CC AP -> AP in this grammar */
```

6.10 Prepositional Phrases

Particle phrases are treated as intransitive PPs.

```
rule(rp(a),r2(X),[not(_)?,wql(_)?,ql(_)*,rpt(X),qlp(_)]).
rule(rp(a),r2(X),[not(_)?,wql(_)?,ql(_)*,rpt(X)]).

rule(main(rp07),pp(X),[r2(X),pp(_)]).
rule(main(rp07),pp(X),[r2(X)],[RC]):- not(RC=p(_)).

rule(main(pp01),pp([prep=between|X]),
```

```
        [p([prep=between|X]),
         np([case=obj|_]),
         cc([conj=and|_]),
         np([case=obj|_])],
        [RC]):- not(RC=p(_)).
```

Pronomial and Proper NPs almost never have attached PPs or Relclauses.

```
rule(main(pp02), pp(X),[p(X),np([type=pro,case=obj|_])]).
rule(main(pp02), pp(X),[p(X),np([type=prop,case=obj|_])]).

rule(main(pp02a),pp(X),[p(X),np([case=obj|_])],[RC]):-
        when(nonvar(RC),
             ( (\+ member(RC,[p(_),pp(_),relpro(_)])),
               ( RC=v(Attr), unify_attr_lists(Attr,[vform=VF|_]),not(VF=to)
               ; not(RC=v(_)) ))).

rule(main(pp03),pp([wh=t|X]),[p(X),advp([wh=t|_])]).
```

/* Although why do we do these differently from rp ? */

```
rule(main(pp07), pp(X),[ql(_),pp(X)]).
rule(main(pp07a),pp(X),[wql(_),pp(X)]).
rule(main(pp08), pp(X),[not(_),pp(X)]).
```

Handles expressions like "about three inches above the cooker".

```
rule(main(pp09),pp(X),[np([sem=measure|_]),pp(X)]).
```

"down about three inches".

```
rule(main(pp10),pp(X),[pp(X),np([sem=measure|_])]).

rule(main(pp05),pp([prep=conj|_]),[pp(_),cc(_),pp(_)],[RC]):-
        not(RC=relpro(_)).
```

"to the right until you reach the tree".

```
rule(main(pp11),pp(X),[pp(X),s([conj=C|_])]):-
        when(nonvar(C),member(C,[until,'''til'])).
```

Force construction of 'of' PPs.

```
rule(main(pp12),pp(X),[p(X),np([case=obj|_])]):- X=[prep=of|_].
```

6.10.1 Aborted PP's

```
rule(main(abort(pp01)),pp([abort=t|X]),[p(X),np([abort=t|_])]).

rule(main(abort(pp02)),pp([abort=t|X]),[p(X)],[RC]):-
        when(nonvar(RC),member(RC,[pp(_),ed(_),aff(_),s(_),sentmark(_)])).
```

6.11 Verb phrases

```
rule(main(vp01),v([vform=VF,aux=t|X]),[v([vform=VF,aux=t|X]),not(_)]).
rule(main(vp02),vp(X),[not(_),vp(X)]).
rule(main(vpadv),v(X),[v(X),advp([wh=f|_])]).

/* Final Adverbs */

rule(main(vp32),vp(X),[vp(X),advp(_)]).
```

6.11.1 Subcategorisation frames

The following verb subcategorisation frames are allowed in the grammar.

pp_pp	[pp(_),pp(_)]
np_pp	[np([case=obj|_]),pp(_)]
pp	[pp(_)]
np_vp(VF)	[np([case=obj|_]),vp([vform=VF|_])]
np_np	[np([case=obj|_]),np([case=obj|_])]
np	[np([case=obj|_])]
ap	[ap(_)]
vp(VF)	[vp([vform=VF|_])]
np_aff	[np([case=subj|_]),aff(_)]
aff	[aff(_)]
s	[s([inv=f|_])]
sbar(C)	[s([conj=C|_])]
sinv	[s([inv=t|_])]
np_ap	[np([case=obj|_]),ap(_)]
intrans	[]

```
rule(main(vp03),
     vp(X),[v([subcat=pp_pp|X]),pp(_),pp(_)]).
rule(main(vp04),
     vp(X),[v([subcat=np_pp|X]),np([case=obj|_]),pp(_)]).
rule(main(vp05),
     vp(X),[v([subcat=pp|X]),pp(_)],[RC]):- not(RC = vp(_)).
rule(main(vp07),
```

```
     vp(X),[v([subcat=np_vp(VF)|X]),np([case=obj|_]),vp([vform=VF|_])]:-
              not(VF=fin).
rule(main(vp09),
     vp(X),[v([subcat=np_np|X]),np([case=obj|_]),np([case=obj|_])],[RC]):-
            not(RC=relpro(_)).

rule(main(vp10),
     vp(X),[v([subcat=np_ap|X]),np([case=obj|_]),ap(_)]).
/* A hack to catch "is that right/aff" */
rule(main(vp15),
     s([inv=t|X]),[v([subcat=np_aff|X]),np([case=subj|_]),aff(_)]).

rule(main(vp15a),
     vp(X),[v([subcat=aff|X]),aff(_)]).
rule(main(vp11),
     vp(X),[v([subcat=np|X]),np([case=obj|_])],[RC]):-
        not(RC=relpro(_)),
        not(RC=np(_)),
        not(RC=wpo(_)),
        /* not(RC=np([wh=t|_])), */
        not(RC=v([vform=to|_])),
        not(RC=vp([clitic=t|_])).
rule(main(vp12),
     vp(X),[v([subcat=pp|X]),pp(_)]).
rule(main(vp13),
     vp(X),[v([subcat=ap|X]),ap(_)],[RC]):-
        not(RC=v([vform=to|_])).

rule(main(vp34),vp(X),[vp(X),cc(_),vp(X)]).

rule(main(vp14),
     vp(X),[v([subcat=vp(VF)|X]),vp([vform=VF|_])]):-
        member(VF,[en,ing,bse,to]).

/* A hack to catch tagger errors */
rule(main(vp21),vp(X),[v([subcat=vp(fin_past)|X]),vp([vform=fin,tns=past|_])]).
rule(main(vp16),vp(X),[v([subcat=sinv|X]),s([inv=t|_])]).
rule(main(vp17),vp(X),[v([subcat=sbar(as)|X]),s([conj=as|_])]).
rule(main(vp18),vp(X),[v([subcat=sbar(that)|X]),s([conj=that|_])]).
rule(main(vp19),vp(X),[v([subcat=s|X]),s([inv=f|_])]).

rule(main(vpintrans),vp(X),[v([subcat=intrans,clitic=f|X])],[RC]):-
        when(nonvar(RC),
             ( (\+ member(RC,[np(_),wpo(_),pp(_),p(_),v(_)])),
               ( RC=advp(Attr),unify_attr_lists(Attr,[wh=f|_])
               ; RC=cs(Attr),unify_attr_lists(Attr,[conj=C|_]),not(C=that)
               ; ( \+ member(RC,[advp(_),cs(_),np(_)])))))).
```

6.11.2 Other VP rules

```
rule(main(vp31),vp(X),[advp(_),vp(X)]).

rule(main(vp33),vp(X),[vp(X),pp(_)]).

rule(main(vp36),vp(X),[cs(_),vp(X)]).

rule(main(vp37),vp(X),[np([sem=measure|_]),vp(X)]):-
        X=[vform=ing|_].
```

6.11.3 Aborted Verb phrases

```
rule(main(abort(vp01)),vp([abort=t,vform=VF|X]),[v([vform=VF|X])],[RC]):-
        when(nonvar(RC),member(RC,[ed(_),sentmark(_)])).
rule(main(abort(vp02)),vp([abort=t,vform=to|X]),[v([vform=to|X])],[RC]):-
        when(nonvar(RC),member(RC,[ed(_),s(_),sentmark(_),aff(_)])).
rule(main(abort(vp03)),vp([abort=t,clitic=t,vform=fin|V]),
        [v([clitic=t,vform=fin|V])],[s(_),cs(_)]).
```

6.12 Clauses

Features of S are ([vform=,inv=,dum=,conj=,abort=]).

Inverted Sentences (questions) e.g. "could you not?" "is it not?".

```
rule(main(s02),s([inv=t,vform=VF|_]),
        [v([aux=t,clitic=f,vform=VF|_]),np([case=subj|_]),not(_)]).

rule(main(s01),s([inv=t|SRest]),[v(V),np(NP)|Rest],RC):-
        rule(_,vp(VP),[v(V)|Rest],RC),
        match([v(V)],[v([aux=t,clitic=f|_])]),
        rule(_,s(S),[np(NP),vp(VP1)],[]),
        match([vp(VP)],[vp(VP1)]),
        match([s(S)],[s([inv=f|SRest])]).
```

Declarative sentences

```
rule(main(s03),s([vform=fin,inv=f,dum=D/*,slash=S*/|_]),
     [np([num=N,case=subj,pers=P,dum=D|_]),
      vp([vform=fin,num=N,pers=P/*,slash=S*/|_])]).
```

WH Questions

```
rule(main(s04),s([vform=fin,inv=t|_]),[advp([wh=t|_]),vp([vform=fin,aux=t|_])]).
rule(main(s05),s([vform=fin,inv=t|_]),[advp([wh=t|_]),s([vform=fin,inv=t|_])]).
rule(main(s05a),s([vform=fin,inv=t|_]),[ap([wh=t|_]),s([vform=fin,inv=t|_])]).
rule(main(s05b),s([vform=fin,inv=t|_]),[np([wh=t|_]),s([vform=fin,inv=t|_])]).
rule(main(s05c),s([vform=fin,inv=t|_]),[wpo(_),s([vform=fin,inv=t|_])]).
rule(main(s05d),s([vform=fin,inv=t|_]),[wql(_),s([vform=fin,inv=t|_])]).
```

Other Sentence rules

```
rule(main(s06),s([vform=VF,inv=I,conj=C|_]),[cs([conj=C|_]),s([vform=VF,inv=I|_])]).
rule(main(s07),s(X),[s(X),cc(_),s(X)]).
rule(main(s08),s([vform=VF,inv=I,conj=C|_]),[cc([conj=C|_]),s([vform=VF,inv=I|_])]).

rule(main(s09),s([conj=none|X]),[pp(_),s([conj=none|X])]).
```

"a distance below that there is a meadow"

```
rule(main(s10),s([dum=f|X]),[np(_),s([dum=t|X])]).

rule(main(s11),s(X),[advp(_),s(X)]).
```

Promote VP to imperative S.

```
rule(main(s12),s([vform=imp,inv=f|_]),[vp([vform=fin,pers=2|_])]).
```

6.12.1 Aborted sentences

```
rule(main(abort(s01)),s([abort=t|_]),[np([case=subj,wh=f|_])],[RC]):-
        when(nonvar(RC),
             ( RC=s(_)
             ; RC=ed(_)
             ; RC=aff(_) /* aff added ? */
             ; RC=np(Attrs),unify_attr_lists(Attrs,[wh=t|_]))).

rule(main(abort(s02)),s([abort=t,conj=C|_]),[cs([conj=C|_])],[RC]):-
        when(nonvar(RC),(RC=ed(_))).
```

6.13 Utterances

```
rule(utt(utt1),s(X),[aff(_),s(X)]).

rule(utt(1),starter(_),[start(_),aff(_)]).
rule(utt(1),starter(_),[start(_),cc(_)]).
```

```
rule(utt(1),starter(_),[start(_),cs(_)]).
rule(utt(1),starter(_),[start(_),ed(_)]).
rule(utt(1),starter(_),[starter(_),aff(_)]).
rule(utt(1),starter(_),[starter(_),cc(_)]).
rule(utt(1),starter(_),[starter(_),cs(_)]).
rule(utt(1),starter(_),[starter(_),ed(_)]).

rule(utt(1),closer(_),[aff(_)],[sentmark(_)]).
rule(utt(1),closer(_),[cc(_)],[sentmark(_)]).
rule(utt(1),closer(_),[cs(_)],[sentmark(_)]).
rule(utt(1),closer(_),[ed(_)],[sentmark(_)]).
rule(utt(1),closer(_),[aff(_),closer(_)]).
rule(utt(1),closer(_),[cc(_),closer(_)]).
rule(utt(1),closer(_),[cs(_),closer(_)]).
rule(utt(1),closer(_),[ed(_),closer(_)]).

/* We add abort=_ to s, so that some of the s can be aborted */

rule(utt(2),utt(_),[start(_),starter(_)?,s([abort=_|_])*,closer(_)?,sentmark(_)]).

rule(utt(3),utt(_),[start(_),starter(_)?,Phrase,closer(_)?,sentmark(_)]):-
        member(Phrase,[np(_),pp(_)]).
```

6.14 Defaults

Rule defaults - are applied whenever we create a new category. These take effect unless something else is specified in the rule.

```
ruledefault(a2([wh=f|_])).
ruledefault(advp([wh=f|_])).

ruledefault(np([abort=f|_])).
ruledefault(np([dum=f|_])).
ruledefault(np([type=norm|_])).
ruledefault(np([wh=f|_])).

ruledefault(pp([conj=none|_])).
ruledefault(pp([abort=f|_])).

ruledefault(vp([abort=f|_])).
ruledefault(vp([mod=f|_])).

ruledefault(v([clitic=f|_])).
ruledefault(v([tns=pres|_])).

ruledefault(s([dum=f|_])).
```

```
ruledefault(s([conj=none|_])).
ruledefault(s([abort=f|_])).
```

Final defaults - are applied after all rules have fired in an attempt to assign them values. VPs must be specified for vform, aux num pers

```
default(vp([vform=fin|_])).
default(vp([aux=f|_])).
default(vp([num=sing|_])).
default(vp([pers=2|_])).
```

Allows control over which attributes get printed.

```
dont_want_to_print(abort=f).
dont_want_to_print(aux=f).
dont_want_to_print(clitic=f).
dont_want_to_print(conj=none).
dont_want_to_print(deg=pos).
dont_want_to_print(dum=f).
dont_want_to_print(inv=f).
dont_want_to_print(sem=any).
dont_want_to_print(mod=f).
dont_want_to_print(type=norm).
dont_want_to_print(wh=f).
dont_want_to_print(tns=pres).
```

6.15 Lexicon

The detailed lexicon currently only contains subcat information for verbs. A sample of the entries in the lexicon are shown below.

lexicon(Verb,vb,SubCat).

The SubCat variable is never instantiated by this rule, only checked.

```
lexicon(G,vb,S):-
        member(G,[have,had,having]),
        when(ground(S),
             member(S,[np,vp(to),vp(en),vp(fin_past),intrans])).

lexicon(G,vb,S):-
        member(G,[do,doing,done]),
        when(ground(S),
             member(S,[pp,np_np,np,vp(bse),np_vp(ing),intrans])).
```

```
lexicon(G,vb,S):-
        member(G,[want,wants,wanting,wanted]),
        when(ground(S),member(S,[np_vp(to),vp(to),np])).

lexicon(G,vb,S):-
        member(G,[stop,stops,stopping,stopped]),
        when(ground(S),member(S,[vp(ing),np,pp,intrans])).

lexicon(let,vb,np_vp(bse)).

lexicon(G,vb,S):-
        member(G,[accord,accords,accorded,according]),
        when(ground(S),member(S,[pp])).

/* Default lexical entry for verbs */

lexicon(_,vb,S):-
        when(ground(S),member(S,[intrans,np,pp])).
```

Chapter 7

Description of the SGML markup of the parsed corpus

This chapter describes the SGML markup of the parsed MT corpus. The Document Type Description described here (`Parsing/dtd/phrasalchannel.dtd` annotates only phrasal constituents, as used when `output_parse(phrasal)` is specified. The SGML description for all the categories in the grammar can be found in the file `Parsing/dtd/syntax.dtd`.

7.1 Top level elements

The document element is `parsedchannel`, which contains a `TEXT` element, followed by an optional `SUMMARY` element. The `TEXT` element contains the parsed conversation (for giver or follower) and the `SUMMARY` element gives a summary of how many times particular grammar rules were used, and how many paragraphs of the corpus were parsed into phrases.

```
<!ELEMENT parsedchannel (TEXT,SUMMARY?)>
<!ATTLIST parsedchannel
        role    (Follower|Giver) #IMPLIED
        date    CDATA            #REQUIRED -- Date/time when this file made --
        id      ID               #REQUIRED >
```

TEXT is made up of a sequence (normally alternating) of PARA and GAP. TEXT is made up of a sequence (normally alternating) of PARA and GAP elements. PARAs consist of parsed speech, GAPs are sequences of pauses and noises.

```
<!ELEMENT TEXT ((%markupStuff)*,(PARA|GAP)*) +(MARKER)>

<!ENTITY % markupStuff "UCODE|SCODE|TIMESTAMP|TURNANCHOR|FOREIGN|
                        CITED-WORD|UNCLEAR|VOCAL|EDITORIAL" >
```

7.2 The structure of PARA

PARA is used as the initial segmentation of the speech. They are the segments used by the POS tagger (ie split before pauses). They have a faint resemblence to paragraphs or sentences.

```
<!ENTITY % paraContent "(%markupStuff;|%wordStuff;|%phraseStuff;)*">
<!ELEMENT PARA  %paraContent >
<!ATTLIST PARA
     TYPE (OK|NOTOK|UTT|TRIVIAL)    OK
     N    NUMBER                    #IMPLIED
     ID   ID                        #IMPLIED >
<!ENTITY % wordStuff   "TW|SIL|NOI|PUN" >
<!ENTITY % phraseStuff "UTT|S|NP|PP|AP|VP|ADVP|STARTER|CLOSER" >
```

N is a number (starting from 1) which counts the number of the paragraph in the file. TYPE describes how the text has been parsed:

TRIVIAL : Paragraph consists only of noises, pauses, discourse markers, filled pauses, interjections and punctuation.

UTT : Paragraph has been parsed into a single UTT element.

OK : Paragraph has been parsed into a sequence of phrasal level constituents.

NOTOK : Paragraph contains non-phrasal constituents not contained in any higher constituent.

We may have an initial timestamp outside a gap, otherwise timestamps are inside para or gap.

GAP consists of a sequence (possibly empty) of NOI followed by either sentence final punctuation or more likely a sentence tagged silence.

```
<!ENTITY % gapContent     "(SIL|NOI|%markupStuff;)*" >
<!ELEMENT GAP  %gapContent >
```

7.3 Phrase level elements

7.3.1 Phrase structure

The attributes defined for these phrase level elements are defined below.

Utterances

```
<!ENTITY % uttContent  "%paraContent" >
<!ELEMENT UTT   %uttContent >
<!ATTLIST UTT %start %dur %rule>
```

Sentences

```
<!ENTITY % SContent    "%paraContent" >
<!ELEMENT S     %SContent >
<!ATTLIST S %vform %dum %abort %conj %inv %start %dur %rule>
```

Phrasal Noun Phrases:

```
<!ENTITY % NPContent   "%paraContent" >
<!ELEMENT NP %NPContent >
<!ENTITY % Nattributes "%num %pers %case %abort %dum %type %sem %wh %start %dur %rule" >
<!ATTLIST NP %Nattributes >
```

Prepositional Phrases:

```
<!ENTITY % PPContent   "%paraContent" >
<!ELEMENT PP %PPContent >
<!ATTLIST PP %prep %conj %abort %wh %start %dur %rule>
```

Verb Phrases:

```
<!ENTITY % VPContent   "%paraContent" >
<!ELEMENT VP %VPContent >
<!ATTLIST VP %vform %num %pers %clitic %abort %aux %mod %tns %start %dur %rule>
```

Adverb Phases:

```
<!ENTITY % ADVPContent "%paraContent" >
<!ELEMENT ADVP %ADVPContent >
<!ATTLIST ADVP %abort %deg %wh %start %dur %rule>
```

Adjective Phrases:

```
<!ENTITY % APContent   "%paraContent" >
<!ELEMENT AP %APContent >
<!ATTLIST AP %abort %deg %wh %start %dur %rule>
```

Starters:

```
<!ENTITY % StarterContent "%paraContent" >
<!ELEMENT STARTER %StarterContent >
<!ATTLIST STARTER %abort %start %dur %rule>
```

Closers:

```
<!ENTITY % CloserContent  "%paraContent" >
<!ELEMENT CLOSER %CloserContent >
<!ATTLIST CLOSER %abort %start %dur %rule>
```

7.3.2 Definitions of phrasal attributes

```
<!ENTITY % case   "CASE   (subj|obj|undef_case)          obj">
```

Case of nouns: SUBJ is nominative case (subject), OBJ is objective case (object), UNDEF_CASE is when case is uncertain or not determined by parser.

```
<!ENTITY % deg    "DEG    (pos|comp|sup|undef_deg)       pos">
```

Degree of adjectives or adverbs: POS is positive (i.e. base form of adjectives), COMP is comparative form (e.g. "bigger"), SUP is the superlative form (e.g. "biggest"), UNDEF_DEG is when parser has not determined the degree.

```
<!ENTITY % dur    "DUR    NUTOKEN                        0">
```

The duration of the word or phrase in seconds.

```
<!ENTITY % num    "NUM    (sing|plur|undef_num)          sing">
```

Number of nouns: SINGular, PLURal or undetermined.

```
<!ENTITY % pers   "PERS   (1|2|3|undef_pers)             3">
```

The person of nouns:

```
<!ENTITY % vform  "VFORM  (fin|en|ing|bse|to|imp|
                           undef_vform)                  fin">
```

The form of a verb:

FIN : finite form, e.g. "goes".

EN : past participle form e.g. "gone".

ING : present participle form e.g. "going".

BSE : base infinitive form e.g. "go".

TO : "to" infinitive form e.g. "to go".

IMP : imperative form e.g. "go!".

UNDEF_VFORM : used when parser is unsure which form to assign.

```
<!ENTITY % wh     "WH     (wh|not_wh|undef_wh)           not_wh">
```

Whether a noun starts with a "wh" word, e.g. "which book".

```
<!ENTITY % abort  "ABORT  (abort|not_abort|undef_abort)  not_abort">
```

Whether a phrase is aborted(unfinished) or not.

```
<!ENTITY % aux    "AUX    (aux|not_aux|undef_aux)        not_aux">
```

Used for verbs to specify whether it is an auxiliary verb or not.

```
<!ENTITY % clitic "CLITIC (clitic|not_clitic)            not_clitic">
```

Whether a verb is a clitic or not e.g. "+'s" versus "is".

```
<!ENTITY % dum    "DUM    (dum|not_dum|undef_dum)        not_dum">
```

Whether a noun is a dummy noun such as "there" or not.

```
<!ENTITY % conj   "CONJ   CDATA                          none">
```

Used if a S or PP starts with a conjunction, if so the lexical value of the conjunction is stored as the value of this attribute.

```
<!ENTITY % inv    "INV    (inv|not_inv|undef_inv)        not_inv">
```

Is a S inverted or not? e.g. "is it time for tea?" is inverted.

```
<!ENTITY % type   "TYPE   (norm|pro|prop)                norm">
```

Type of noun phrase, PRO is a pronoun, PROP are proper nouns, NORM all the rest.

```
<!ENTITY % rule   "RULE   CDATA                          #IMPLIED">
```

The identifier of the rule which constructed this constituent.

```
<!ENTITY % sem    "SEM    (measure|any|undef_sem)        any">
```

Noun phrases are marked if they are measure phrases or not, e.g. “three inches” is a measure phrase, “two trees” is not.

```
<!ENTITY % start  "START  CDATA                          'UNDEF_START'" --undefined -->
```

Start time of element, in seconds from start of file.

```
<!ENTITY % prep   "PREP   CDATA                          #REQUIRED">
```

The head proposition in a PP.

```
<!ENTITY % mod    "MOD    (mod|not_mod|undef_mod)        not_mod">
```

Whether a VP has a modal verb as head or not.

```
<!ENTITY % tns    "TNS    (pres|past|undef_tns)          pres">
```

The tense of finite verb phrases.

7.4 The structure of the summary

```
<!ELEMENT SUMMARY (PARAS,TRIVIAL,NONTRIVIAL,UTTS,OK,NOTOK,RULE*)>
```

PARAS is the number of <PARA> elements in the file.

TRIVIAL is the number of these which are trivial.

NONTRIVIAL are the other non-trivial ones.

UTTS is the number which have been parsed as <UTT>.

OK is the number which have parsed as a sequence of reasonable pieces.

NOTOK are the rest which have not been well parsed.

RULE counts how many times each rule was used

In the last three elements i.e. UTTS, OK, and NOTOK; the PERC attribute is the percentage of all <PARA>s and NTPERC is percentage of the non-trivial ones.

```
<!ELEMENT (PARAS, TRIVIAL, NONTRIVIAL, UTTS, OK, NOTOK, RULE) - o EMPTY>
<!ATTLIST PARAS
    N       NUMBER   0 >
<!ATTLIST (TRIVIAL, NONTRIVIAL)
    N       NUMBER   0
```

```
    PERC    NUTOKEN  0 >
<!ATTLIST (UTTS, OK, NOTOK)
    N       NUMBER   0
    PERC    NUTOKEN  0
    NTPERC  NUTOKEN  0 >
<!ATTLIST RULE
    N       NUMBER   0
    R       CDATA    #REQUIRED >
```

For example, a summary element element might look like:

```
<SUMMARY>
<PARAS      N = 48>
<TRIVIAL    N = 9 PERC=18.75>
<NONTRIVIAL N = 39 PERC=81.25>
<UTTS       N = 29 PERC=60.42 NTPERC=74.36>
<OK         N = 5 PERC=10.42 NTPERC=12.82>
<NOTOK      N = 5 PERC=10.42 NTPERC=12.82>
<RULE R="main(s11)" N=1>
<RULE R="disf(rp(a))" N=2>
<RULE R="main(advp06)" N=1>
<RULE R="disf(main(s06))" N=1>
<RULE R="main(abort(np02))" N=1>
<RULE R="disf(utt(1))" N=1>
<RULE R="disf(main(hack1))" N=1>
<RULE R="main(vpintrans)" N=1>
<RULE R="main(ap16)" N=2>
<RULE R="main(np6catch)" N=2>
...
</SUMMARY>
```

7.5 Etc

Other miscellaneous SGML markup.

```
<!ELEMENT       foreign - -     (%paraContent)>
<!ATTLIST       foreign
        lang    (FR | DE | LA | ES)     #REQUIRED>

<!ELEMENT       unclear - -     (%paraContent)>
<!ELEMENT       cited-word      - -     (%paraContent)>

<!ENTITY amp "&">
<!ENTITY noise "&amp;noise">
```

```
<!ELEMENT turnanchor            EMPTY>
<!ATTLIST turnanchor    synch   NAME    #REQUIRED > <!-- really an idref-->
```

Chapter 8

Availability

8.1 Availability of software, data, documentation

All the reports and papers written during the project can be accessed at the `http://www.ltg.ed.ac.uk/~dmck/ppp.html` web page. This page also allows access to the complete parsed corpus and allows some limited search facilities on the parsed corpus.

Hard copies of the documentation can be had from the Technical Report Librarian, Human Communication Research Centre, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, UK.

8.2 WWW demonstration page

There is also an introductory web page for the Maptask Corpus and its annotations at `http://www.cogsci.ed.ac.uk/~amyi/maptask/demo_top.html`.

Bibliography

[Abney 96] Steven Abney, "*Partial Parsing via Finite-State Cascades*", In Proceedings of the ESSLLI '96 Robust Parsing Workshop. 8 pages. 1996.

[Anderson et al. 91] A.H.Anderson, M.Bader, E.G.Bard, E.Boyle, G.Doherty, S.Garrod, S.Isard, J.Kowtko, J.McAllister, J.Miller, C.Sotillo, H.Thompson & R.Weinert, "*The HCRC Map Task Corpus*", Language and Speech, 34(4), pp 351-366, 1991.

[Ballim 94] Ballim, A. & G. Russell (1994) "*LHIP: Extended DCGs for Configurable Robust Parsing*", Proceedings of the 15th international Conference on Computational Linguistics (COLING 94), Kyoto, Japan., 501-507.

[Black et.al. 92] Ezra Black, John Lafferty, and Salim Roukos, "*Development and evaluation of a broad-coverage probablistic grammar of English-language computer manuals*", In ACL 1992, pages 185–192.

[Black et.al. 93] E. Black, R. Garside, and G. Leech, editors. "*Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach*", Rodopi, Amsterdam, Atlanta, 1993.

[Brill & Resnik 94] Brill, E. and Resnik,P. "*A rule-based approach to prepositional phrase attachment*", Proceedings of the 15th Computational Linguistics conference (COLING) 1994, pp 1198-1204.

[Brill 97] Brill,E., "*Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging*", To appear in Natural Language Processing Using Very Large Corpora. Kluwer Academic Press. 1997, Available at `http://www.cs.jhu.edu/~brill/acadpubs.html`.

[Briscoe & Carroll 93] E. J. Briscoe and John Carroll, "*Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars*", Computational Linguistics, 19(1):25–59, 1993.

[Briscoe & Waegner 93] E. J. Briscoe and N. Waegner, "*Undergeneration and robust parsing*", In W. Meijs, editor, Proceedings of the ICAME Conference, Amsterdam, 1993. Rodopi.

[Briscoe 94a] Briscoe, T. "*Parsing (with) Punctuation etc*", Research Paper, Rank Xerox Research Centre, Grenoble, 1994.

[Briscoe 94b] E. J. Briscoe, "*Prospects for practical parsing: robust statistical techniques*", In P. de Haan and N. Oostdijk, editors, Corpus-based Research into Language: A Feschrift for Jan Aarts, pages 67–95. Rodopi, Amsterdam, 1994.

[Briscoe et.al 98] Ted Briscoe, John Carroll, Glenn Carroll, Stefano Federici, Greg Grefenstette, Simonetta Montemagni, Vito Pirrelli, Irina Prodanof, Mats Rooth, and Massimo Vannocchi, "*Sparkle project: PHRASAL PARSER SOFTWARE - DELIVERABLE 3.1*" `http://www.ilc.pi.cnr.it/sparkle.wp3/wp3new/wp3new.html`

[Chanod 95] Chanod, J-P., & Tapanainen P. (1995) "*Statistical and constraint-based taggers for French*", Research Paper, Rank Xerox Research Centre, Grenoble, 1995.

[Church 88] Kenneth Church. "*A stochastic parts program and noun phrase parser for unrestricted text*". In Proceedings of the Second Conference on Applied Natural Language Processing, pages 136–143, Austin, Texas, 1988. ACL.

[Cole et.al. 96] Ronald A. Cole, Editor in Chief, "*Survey of the State of the Art in Human Language Technology*", `http://www.cse.ogi.edu/CSLU/HLTsurvey/HLTsurvey.html`

[Collins & Brooks 95] Collins, M. and Brooks, J. "*Prepositional phrase attachment through a backed-off model*", Research paper University of Pennsylvania 1995.

[Cutting et al. 92] Cutting,D., Kupiec,J., Pedersen,J. & Sibun,P., "*A practical part-of-speech tagger*", In Proceedings of the 3rd Conference on Applied Language Processing, pages 133–140, Trento, Italy, 1992. Available from `ftp://parcftp.xerox.com/pub/tagger/` .

[Elworthy 93] D. Elworthy. "*Part-of-speech tagging and phrasal tagging*". Technical report, University of Cambridge Computer Laboratory, Cambridge, England, 1993.

[Feldweg 95] Feldweg,H., "*Implementation and evaluation of a German HMM for POS disambiguation*", Proceedings of the ACL SIGDAT Workshop, Dublin 1995. Available at `http://xxx.lanl.gov/abs/cmp-lg/9502038`.

[Fujisaki et.al. 89] T. Fujisaki, F. Jelinek, J. Cocke, E. Black, and T. Nishino. "*A probabilistic parsing method for sentence disambiguation*", In Proceedings of the International Workshop on Parsing Technologies, Pittsburgh, August 1989.

[Garside et.al. 87] R. Garside, G. Leech, and G. Sampson. "*Computational Analysis of English: A Corpus-based Approach*". Longman, London, 1987.

[Greenbaum 92] Greenbaum, S. (1992), "*A new corpus of English: ICE*", Directions in Corpus Linguistics: Proceedings of Nobel Symposium 82. Stockholm August 1991, ed. J.Svartvik, pp 171-179, Berlin: Mouton.

[Greene & Rubin 71] B. B. Greene and G. M. Rubin. "*Automatic grammatical tagging of English*". Technical report, Brown University, 1971.

[Heeman 94] Heeman, P. and Allen, J., "*Detecting and Correcting Speech Repairs*", Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, June 1994.

[Hindle 83a] D. Hindle. "*Deterministic parsing of syntactic nonfluencies*", In ACL 1983, pages 123–128.

[Hindle 83b] Donald Hindle. "*User manual for Fidditch, a deterministic parser*", Technical Report Technical Memorandum 7590-142, Naval Research Laboratory, 1983.

[Hindle 89] D. Hindle. "*Acquiring disambiguation rules from text*", In Proceeds of the 27th Annual Meeting of the Association for Computational Linguistics, pages 118–125, Vancouver, Canada, 1989.

[Hindle 92] Donald Hindle. "*An analogical parser for restricted domains*". In Proceedings of the Fifth DARPA Speech and Natural Language Workshop, pages 150–154. Defense Advanced Research Projects Agency, Morgan Kaufmann, February 1992.

[Hindle 93a] Donald Hindle. "*A parser for text corpora*". In B. T. S. Atkins and A. Zampolli, editors, Computational Approaches to the Lexicon. Oxford University Press, 1993.

[Hindle 93b] Hindle, D. and Rooth, M. "*Structural ambiguity and lexical relations*", Computational Linguistics 19 (1) 1993.

[Jensen 91] K. Jensen. "*A broad-coverage natural language analysis system*", In Masaru Tomita, editor, Current Issues in Parsing Technology. Kluwer Academic Press, Dordrecht, 1991.

[Jensen & Heidorn 93] K. Jensen and G. Heidorn. "*Natural Langauage Processing: The PLNLP Approach*". Kluwer Academic, Boston, Dordrecht, London, 1993.

[Jones 94] B. Jones. "*Can punctuation help parsing?*", In COLING [COL94].

[Karlsson et.al. 94] F. Karlsson, A. Voutilainen, J. Heikkil, and A. Anttila, editors. "*Constraint Grammar: A Language-Independent Formalism for Parsing Unrestricted Text*". Mouton de Gruyter, Berlin, New York, 1994.

[Koskenniemi 90] K. Koskenniemi. "*Finite-state parsing and disambiguation*". In Proceedings of the 13th International Conference on Computational Linguistics, Helsinki, 1990. ACL., pages 229–232.

[Lee 94] Kong Joo Lee et al., "*A robust parser based on syntactic information*", Proceedings of the 15th Computational Linguistics conference (COLING) 1994, pp 223-228, 1994.

[Leech & Garside 91] G. Leech and R. Garside. "*Running a grammar factory: the production of syntactically analysed corpora or 'treebanks'*", In S. Johansson and A. Stenstrom, editors, English Computer Corpora: Selected Papers and Bibliography. Mouton de Gruyter, Berlin, 1991.

[Leech et.al. 94] G. Leech, R. Garside, and M. Bryant. "*The large-scale grammatical tagging of text*". In N. Oostdijk and P. de Haan, editors, Corpus-Based Research into Language, pages 47–63. Rodopi, Atlanta, 1994.

[Levelt 89] Levelt, W.J.M. (1989) "*Speaking: From Intention to Articulation*", MIT Press 1989. *In particular, chapter 12 "Self-Monitoring and Self-Repair".*

[Lickley 94] Lickley,R., "*Detecting Disfluency in Spontaneous Speech*", Thesis, Department of Linguistics, University of Edinburgh 1994.

[Magerman & Weir 92] D. M. Magerman and C. Weir. "*Efficiency, robustness and accuracy in Picky chart parsing*", In ACL 1992.

[Marcus 80] Mitchell P. Marcus. "*A Theory of Syntactic Recognition for Natural Language*", MIT Press, Cambridge, Massachusetts, 1980.

[Marcus et.al. 83] M. Marcus, D. Hindle, and M. Fleck. "*D-theory: talking about talking about trees*", In ACL 1983, pages 129–136.

[Marcus et.al. 92] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz "*Building a large annotated corpus of English: the Penn Treebank*" `http://www.cis.upenn.edu/~ldc/treebank2/cl93.html`

[Marcus et.al. 94] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, Britta Schasberger "*The Penn TREEBANK: Annotating predicate argument structure*" `http://www.cis.upenn.edu/~ldc/treebank2/arpa94.html`

[de Marcken 90] C. de Marcken, "*Parsing the LOB corpus*", In ACL 1990, pages 243–251.

[Marshall 83] I. Marshall. "*Choice of grammatical word-class without global syntactic analysis: tagging words in the LOB corpus*". Computers in the Humanities, 17:139–150, 1983.

[McKelvie 98a] McKelvie,D., "*The Syntax of Disfluency in Spontaneous Spoken Language*", HCRC Research Paper HCRC/RP-95, Edinburgh, 1998.

[McKelvie 98b] McKelvie,D., "*Robust Parsing and Part-of-Speech Ragging of Transcribed Speech Corpora: Report of Research Activities and Results for ESRC Project R000236800*", unpublished, available as `http://www.ltg.ed.ac.uk/~dmck/PPPFinalReport/report.ps`, 1998.

[McKelvie 98c] McKelvie,D., "*SDP - Spoken Dialogue Parser*", HCRC Technical Report HCRC/TR-96, Edinburgh, 1998.

[Mikheev 97] Mikheev,A and S.Finch, "*Collocation Lattices and Maximum Entropy Models*", In Proceedings of WVLC'97 (Hong Kong). ACL 1997. pp. 15.

[Mikheev 98] Mikheev,A, "*Unsupervised Learning of Part-of-Speech Guessing Rules*", In Journal for Natural Language Engineering.. vol 2(2). Cambridge University Press. 1996.

[Miller 95] Miller, J. & R. Weinert (1995) "*Spontaneous spoken language and writing: Syntax, discourse and language acquisition*", book to be published by OUP.

[Moshier 97] M. A. Moshier, "*Is HPSG featureless or unprincipled?*", Linguistics and Philosophy 20, pp 669-695, 1997.

[Nakatani 94] Nakatani, C. (1994) "*A corpus-based Study of Repair Cues in Spontaneous Speech*", J. Acoust. Soc. Am. 95 (3) March 1994.

[Oostdijk 91] N. Oostdijk. "*Corpus Linguistics and the Automatic Analysis of English*", Rodopi, Amsterdam, Atlanta, 1991.

[Pereira & Schabes 92] Fernando C. N. Pereira and Yves Schabes. "*Inside-outside reestimation from partially bracketed corpora*", In ACL 1992, pages 128–135.

[Roach 94] Roach, P. Knowles, G., Varadi, T., Arnfield, S., "*MARSEC: A Machine-Readable Spoken English Corpus*" Journal of the International Phonetic Association volume 24.1 May 1994.

[Rose 88] S. J. De Rose. "*Grammatical category disambiguation by statistical optimization*". Computational Linguistics, 14(1):31–39, 1988.

[Schabes et.al. 93] Y. Schabes, M. Roth, and R. Osborne. "*Parsing the Wall Street Journal with the inside-outside algorithm*", In EACL 1993.

[Sharman et.al. 90] R. Sharman, F. Jelinek, and R. L. Mercer. "*Generating a grammar for statistical training*", In Proceedings of the Third DARPA Speech and Natural Language Workshop, pages 267–274, Hidden Valley, Pennsylvania, June 1990. Defense Advanced Research Projects Agency, Morgan Kaufmann.

[Shriberg 94] Shriberg,E., "*Preliminaries to a Theory of Speech Disfluencies*", Thesis, Psychology Department, University of California at Berkeley, 1994.

[Stolcke 94] Stolcke,A., "*Bayesian learning of probabilistic language models*", PhD. thesis, ICSI, University of Berkeley, 1994.

[Van Oirsouw 87] Van Oirsouw, R.R. (1987) "*The syntax of coordination*", Croom Helm Linguistics Series, 1987.

[Voutilainen 94] A. Voutilainen. "*Three Studies of Grammar-Based Surface Parsing of Unrestricted English Text*". PhD thesis, University of Helsinki, Department of General Linguistics, University of Helsinki, 1994.

[BNC 97] "*The British National Corpus*", Available at `http://info.ox.ac.uk/bnc` .

[Wheatley 92] B. Wheatley, G. Doddington, C. Hemphill, J. Godfrey, E.C. Holliman, J. McDaniel, and D. Fisher, "*Robust Automatic Time Alignment of Orthographic Transcriptions with Unconstrained Speech*", Proc. ICASSP-92, Vol. I, 533-536, 1992.