The Alvey Natural Language Tools Grammar (4th Release)[1]

Claire Grover
(grover@cogsci.ed.ac.uk)

Human Communication Research Centre, University of Edinburgh
2 Buccleuch Place, Edinburgh, EH8 9LW, UK

John Carroll, Ted Briscoe
(jac / ejb @cl.cam.ac.uk)

Computer Laboratory, University of Cambridge
Pembroke Street, Cambridge, CB2 3QG, UK

January 29, 1993

[1]This work was supported by research grant GR/F35463 from the U.K. Science and Engineering Research Council under the IED programme.

Abstract

This report provides a detailed description of the fourth and final funded release of the ANLT grammar. It includes a description of both the syntactic and semantic components of the grammar, the ANLT grammar formalism and metagrammar compilation process, and a listing of the complete grammar. It provides some background and explanation for the syntactic and semantic frameworks adopted and pointers to relevant literature. It ends with an assessment of the adequacy and coverage of the fourth grammar and of possible areas for further work.

ANLT Distribution Arrangements

A basic description of the ANLT, distribution arrangements and instructions, and a machine-readable specimen licence agreement are available in files on the FTP server ftp.cl.cam.ac.uk (128.232.0.56). To fetch this information use anonymous FTP (login with user name anonymous, and password your e-mail address), go to the directory 'nltools', and fetch the files:

basic : basic description of the ANLT and distribution arrangements
licence : a machine-readable specimen licence agreement
instruct : instructions on how to FTP the ANLT itself

The following example shows how to fetch these files:

```
$ ftp ftp.cl.cam.ac.uk
Connected to swan.cl.cam.ac.uk.
220- swan.cl.cam.ac.uk FTP server (Version 5.60+UA) ready.
...
Name (ftp.cl.cam.ac.uk:jac): anonymous
Password (ftp.cl.cam.ac.uk:anonymous): <type your e-mail address here>
...
ftp> cd nltools
250 CWD command successful.
ftp> get basic
...
ftp> quit
221 Goodbye.
```

(The $ is the Unix shell command prompt). If the FTP command does not know about the address ftp.cl.cam.ac.uk, try giving the command the internet number (128.232.0.56) instead. If you still have problems, or FTP is not available to you, then you can obtain the ANLT on magnetic tape by writing to the address below specifying the type of tape and format you require:

Lynxvale WCIU Programs
20 Trumpington St
Cambridge, CB2 1QA, UK
Fax: +223 332797

Contents

1 Overview

This report describes a general-purpose wide-coverage morphosyntactic and semantic analyser for English developed within the Alvey Natural Language Tools (ANLT) system. The ANLT system comprises a morphology and lexicon system (Pulman *et al* 1988, Ritchie *et al* 1987, 1992, Russell *et al* 1986), a chart parser for unification-based formalisms (Carroll 1993, Phillips 1986, Phillips & Thompson 1987), a substantial English lexicon (Boguraev *et al* 1987, Carroll & Grover 1989), derived semi-automatically from the machine-readable version of a conventional learners' dictionary, and associated word grammar and a wide-coverage sentence grammar for English which yields both syntactic and semantic analyses. The ANLT system has been integrated with a software environment designed to facilitate rapid development of substantial, wide-coverage grammars (hereafter the Grammar Development Environment (GDE), Carroll *et al* 1991). This report describes the fourth version of the ANLT sentence grammar produced using the GDE. A previous version of the grammar is described in Grover *et al* (1989) which is superseded by this report. The ANLT sentence grammar is written in a formalism for grammar rules which supports a notation similar to that of Generalized Phrase Structure Grammar (GPSG, Gazdar, Klein, Pullum and Sag 1985, henceforth GPSG85). Details of the formalism are discussed in Boguraev *et al* (1988), Briscoe *et al* (1987a, 1987b) and Carroll *et al* (1991). Many of the analyses adopted in the ANLT grammar are based on those proposed in GPSG85 and this document therefore presupposes some familiarity with GPSG. Users and readers who are not familiar with the theory are recommended to consult Sells (1986), Horrocks (1987) and GPSG85. A relatively recent feature of the GDE formalism (and of the grammar) is the computation of a semantic interpretation by means of rule-to-rule translations expressed in the lambda calculus (as described, for example, in Dowty, Wall and Peters 1981). The resulting logical forms provide an event-based (Davidson 1967, Parsons 1990) and unscoped (e.g. Alshawi, 1992) representation of the compositional semantics of the input sentence.

The body of this report provides a general description of the grammar—both syntax and semantics. Detailed explanations of specific rules may be found in the comments associated with each rule in the grammar listing in Appendix 1. Throughout the report reference will be made to specific rules via their rule names as used in the grammar listing. On occasion, blocks of rules will be referred to; for example, the block of ID rules N1/VPMOD1–N1/VPMOD3 and these correspond to blocks in the grammar listing.

In order to facilitate the description of the grammar, we first provide in Section 2 an introduction to the different rule types used in the GDE and illustrate their use with examples from the grammar. In Section 3 we give a general overview of the feature system used in the syntax part of the grammar and in Section 4 we give a brief overview of the semantics. Sections 5 and 6 provide a more detailed description of the syntactic/semantic analyses generated by the grammar—section 5 describes the basic phrasal categories and section 6 describes how we handle specific constructions. In Section 7 we briefly describe an illustrative scoper, which creates deterministic logical formulas, provided with the system.

Appendix 1 contains some sample lexical entries, Appendix 2 shows some sentences which we have used to test the grammar and Appendix 3 contains a complete listing of the grammar.

2 The Grammar Formalism

The grammatical formalism employed is a metagrammatical notation which induces an 'object' grammar. This notation is based on GPSG85, but has been modified and extended to be more flexible and expressive and is interpreted somewhat differently. The motivation for these changes is to provide a formalism which is a specialised programming language for specifying grammatical theories and grammars for particular languages, rather than defining a (restrictive) theory directly (see Briscoe *et al* 1987a).

An object grammar consists of a set of phrase structure (PS) rules whose categories are feature complexes and which form the input to the parser. Object grammar PS rules may have associated with them a definition of their semantic translation (although they need not). After the syntactic part of the rule has been used to produce a syntactic parse tree, the semantic part is used to produce a compositional semantic analysis via the operation of beta-reduction or some other well-defined (functional) operation on logical formulae.

Metagrammar rules are input to the GDE either as ID rules or as PS rules. ID rules ('immediate dominance' rules—see GPSG85 for details) are the same as PS rules except that the daughters are not ordered with respect to one another. Information about how to order the daughters is expressed by a set of LP rules ('linear precedence' rules). A metagrammar ID rule is input in a form such as (1) and this will be expanded out into an object grammar PS rule as in (2).

In both cases a rule consists of a rule name (VP/SFIN1), a comment (after the semicolon but note that comments are optional), a rule definition (`VP --> H[SUBCAT SFIN, SUBTYPE NONE], S[+FIN]` or its expanded form in (2)) and a semantic translation (the rest, separated by a colon). As can be seen, grammar compilation affects the syntax part of the rule definition, turning it from a concisely specified, linguistically transparent definition into something which is less readable but which contains all the information needed by the parser.[1] There is no analogous notion of metagrammar and compilation within the semantic component of the grammar.

(1)
```
IDRULE VP/SFIN1 : ; believes (that) he can do it
   VP --> H[SUBCAT SFIN, SUBTYPE NONE], S[+FIN] :
   2 = [SLASH NOSLASH], (lambda (Q) (Q (lambda (e) (lambda (x)
       (1 e x (2 (lambda (prop) (lambda (ta) (lambda (equa)
         (prop (uqe ((equa some) (e3) (ta e3))))))))))
      (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2], (lambda (Q) (Q (lambda (e) (lambda (x)
       (lambda (wh) (1 e x (2 (lambda (prop) (lambda (ta) (lambda (equa)
         (prop (uqe ((equa some) (e3) (ta e3))) wh)))))))))
      (lambda (e2) e2) (lambda (qu) qu))).
```

[1]The format shown in (2) is one which the GDE shows to the user (as a response to the command 'view normalised linearised fully-instantiated idrule VP/SFIN1'). It is not the format actually used by the parser—see Carroll *et al* (1991) for details.

(2)

```
VP/SFIN1 : ; believes (that) he can do it
   [N -, V +, BAR 2, SUBJ -, CONJ NULL, VFORM @7, BEGAP -, FIN @11,
      PAST @12, PRD @13, AUX -, NEG @17, SLASH @19, PRO -, COORD -,
      AGR [N +, V -, BAR 2, NFORM NORM, PER @21, PLU @22, COUNT @23,
      CASE @24], ELLIP -] -->
   [N -, V +, BAR 0, SUBCAT SFIN, CONJ NULL, VFORM @7, H +, FIN @11,
      PAST @12, PRD @13, AUX -, INV -, PSVE -, NEG @17, PRO -,
      COORD @, SUBTYPE NONE, AGR [N +, V -, BAR 2, NFORM NORM,
      PER @21, PLU @22, COUNT @23, CASE @24]]
   [N -, V +, BAR 2, SUBJ +, CONJ NULL, VFORM NOT, BEGAP -, FIN +,
      PAST @, PRD @, AUX @, INV -, COMP @, SLASH @19, PRO @, WH NO,
      UB NO, EVER NO, COORD @, AGR @, UDC -, ELLIP -] :
   2 = [SLASH [NOSLASH +]], (lambda (Q) (Q
            (lambda (e) (lambda (x) (1 e x (2 (lambda (prop)
                 (lambda (ta) (lambda (equa)
                       (prop (uqe ((equa some) (e3)
                            (ta e3))))))))))) (lambda (e2) e2)
             (lambda (qu) qu))) :
   2 = [SLASH [BAR 2]], (lambda (Q) (Q (lambda (e)
               (lambda (x) (lambda (wh) (1 e x
                   (2 (lambda (prop) (lambda (ta) (lambda (equa)
                            (prop (uqe ((equa some) (e3) (ta e3)))
                                   wh))))))))) (lambda (e2) e2)
            (lambda (qu) qu))).
```

Features consist of feature name/feature value pairs. Feature values may be variables (introduced by @, as in (2)). Variables are either bound (@ followed by a number) or unbound (@ standing alone). Where variables are bound, this means that they must be instantiated to the same value as a result of the parse. Where they are unbound, they can be instantiated by any legitimate value for that feature. Parsing with the object grammar involves matching categories by unifying their feature sets. Unlike GPSG, our metagrammar defines a set of partially instantiated phrase structure rules and not a set of fully instantiated local trees. Our definition of what it is for two categories to 'match' or unify differs from the GPSG definition. In GPSG categories match by 'extension': that is, a category which is less specified in terms of the number of features it contains will match a category which is more specified providing that the feature values of their common features do not conflict. In our grammar we have chosen to define matching in terms of fixed-arity term unification whereby for two categories to match we require that they each have exactly the same features with non-conflicting values for those features. To give an example, in GPSG the feature structures in (3a) and (3b) can unify with the result (4). For us, that unification would fail since the two categories do not have the same number of features. In order to get the same result we would unify (5a) and (5b).

(3) a `[N +, V -, PLU +, PER 3]`

b `[N +, V -]`

(4) `[N +, V -, PLU +, PER 3]`

(5) a `[N +, V -, PLU +, PER 3]`

b `[N +, V -, PLU @, PER @]`

Thus, for each different category type, we define the full set of features that it must have. The Category rules described in Section 2.1.4. are designed to 'flesh out', in the course of expanding the grammar, a category mentioned in a rule with the full set of features that are appropriate to it. (See Briscoe *et al* 1987b for further discussion of the use of fixed-arity unification.)

Semantic formulae associated with rules are well-formed formulae of the lambda calculus. Often different formulas are associated with distinct instantiations of features in rules. The formulae associated with instantiated local trees at parse time are combined to form the semantics of superordinate constituents using beta-reduction on these formulae. The resulting logical forms for sentences are usually unscoped first-order predicate calculus formulae (see section 4 for further details).

2.1 Rule Types

The metagrammar is designed to capture linguistic generalisations and so simplify and abbreviate the statement of 'object' rules, such as (2), and contains rules of the following eleven kinds.

2.1.1 Feature Declarations

Feature Declarations define the feature system used by the grammar. They encode the possible values of a given feature. Much of the feature system is very similar to that used in GPSG, but a feature can additionally take a variable value which ranges over the set of actual values as declared. (6a) and (6b) are atomic-valued features. (6c) is category-valued.

(6) a `FEATURE PLU{+, -}`

b `FEATURE VFORM{BSE, ING, EN, TO, NOT}`

c `FEATURE SLASH CAT`

2.1.2 Feature Set Declarations

Feature Set Declarations define sets of features which propagate in the same manner and which will appear together on particular categories. For example, features which propagate between N2s and their head daughters (PLU, PER, etc.) may be grouped together in a set called NOMINALHEAD, as in (7). The name of this set may then be used to refer to this collection of features in the rules which perform this type of propagation.

(7)
```
SET NOMINALHEAD = {PLU, POSS, CASE, PRD, PN, PRO, PROTYPE, COUNT,
                   NFORM, PER, REFL, NUM, DEMON, ADV, PART}
```

2.1.3 Alias Declarations

Aliases are a convenient abbreviatory device for naming categories and feature complexes in rules. They do not affect the expressive power of the formalism.

(8) a `ALIAS N2 = [N +, V -, BAR 2].`

b `ALIAS +NOM = [CASE NOM].`

2.1.4 Category Declarations

Category Declarations define a particular category as consisting of a given set of features. These declarations are used to expand out the partiallly specified categories which typically occur in the specification of immediate dominance or phrase structure rules. They make the formalism more explicit by obliging the grammarian to state which features will appear on a given category. Category Declarations replace part of the function of Feature Cooccurrence Restrictions in GPSG. The category declaration in (9a) causes all nominal categories in immediate dominance and phrase structure rules to be expanded with whichever features in the set NOMINALHEAD they are not already specified with. These features will be given variable values. The category declaration in (9b) adds further features to [BAR 2] nominal categories.

(9) a `CATEGORY NOUN : [N +, V -] => NOMINALHEAD.`

b `CATEGORY N2 : N2 => {SPEC, DEF, NEG, AFORM, CONJ, QFEAT, COADV, KIND}`

2.1.5 Immediate Dominance Rules

Immediate dominance (ID) rules encode permissible dominance relations in phrase structure trees. The rules in (10) define an immediate dominance relation between a mother and its daughters; however, other properties of the object rules which result from compilation (e.g. the linear order of the daughters in it) are determined by other types of rules in the grammar. (10a) defines a nominal constituent consisting of a head and an AP modifier (which will precede the head in the linearised version of the rule). The ID rule in (10b) builds an infinitival VP from the word *to* and a base VP and the rule in (10c) builds an imperative S from a single daughter VP.

(10) a
```
IDRULE N1/APMOD1 : ; busy man. The -PRD, DISTR ATT restriction on
                   ; the A2 prevents adjectives with complements
                   ; matching. The feature MOD on the N1 cuts down
                   ; on the number of parses by making premodifiers
                   ; attach lower than postmodifiers.
   N1[MOD PRE] --> A2[-PRD, DISTR ATT, -QUA], H1[MOD NONE, PRO -] :
   2 = [SLASH NOSLASH], (lambda (x) (and (2 x) (1 x))) :
   2 = [SLASH X2], (lambda (x) (lambda (wh) (and (2 x wh) (1 x)))).
```

b IDRULE VP/TO : ; to + base VP - infinitival VP of the kind

```
IDRULE VP/TO : ; to + base VP - infinitival VP of the kind
               ; which occurs in infinitival S and in numerous
               ; control constructions.
   VP[+AUX, VFORM TO, -FIN, ELLIP -, COORD -] --> H[SUBCAT TO],
   VP[BSE, ELLIP -, COORD -] : 2.
```

c

```
IDRULE S/IMPER : ; imperative S. Imperatives are distinguished by
                 ; being both VFORM BSE and +FIN. All other +FIN
                 ; forms are VFORM NOT.
   S[SLASH NOSLASH, COMP NORM] -->
   H2[-SUBJ, BSE, FIN +, AGR N2[NFORM NORM], SLASH NOSLASH] :
   (1 (lambda (prop) (lambda (ta) (lambda (equa)
        (lambda (Q) (Q (lambda (e) (prop e (pro (the (y) (hearer y)))))
                         (lambda (e2) (NOTENSE (ta e2))) equa)))))).
```

The semantic part of an ID rule may vary in complexity: it may consist of a single translation as in (10b) and (10c) or it may be made conditional on feature instantiations which may arise during a parse, as in (10a). In all cases the semantic part of a rule is an instruction as to how to build a translation for the mother node out of the translations of the daughters. The categories in a rule are numbered from 0, where 0 is the mother, 1 is the first daughter, 2 is the second daughter etc. and these numbers are used in the semantics to represent the translation associated with the daughters. Thus in (10b), the semantic part of the rule simply equates the translation of the mother with the translation of its VP daughter, 2, i.e. the presence of the infinitive marker *to* has no effect on the semantics of a VP.

In (10c) the translation is much more complex: it is not appropriate to describe it in detail here but note that, as in all the semantic translations in the grammar, functional application is at work—the daughter category, 1, is applied to an argument (which happens to be a complex lambda expression).

In (10a), the translation differs according to whether the N1 contains a gap or not. The conditions `2 = [SLASH NOSLASH]` and `2 = [SLASH X2]` refer to the way in which the feature SLASH happens to have become instantiated on the second daughter: a different translation will apply depending on whether the feature SLASH on this N1 has got the value NOSLASH (which is an alias for [NOSLASH +]) or X2 (which is an alias for a [BAR 2] category). Once again, the 1 and 2 in the lambda expressions refer to the translations of the daughter categories. In the case of the first conditional of (10a), if the A2 daughter translates as `(lambda (x1) (BUSY x1 (degree very)))` and the second daughter translates as `ABBOT` then the translation of the phrase *very busy abbot* will be (11a) which, after beta reduction, becomes (11b).

(11) a

```
(lambda (x) (and (ABBOT x)
   ((lambda (x1) (BUSY x1 (degree very))) x)))
```

b

```
(lambda (x) (and (ABBOT x) (BUSY x (degree very))))
```

For the second conditional of (10a), if the A2 daughter translates as `(lambda (x1) (CRAZY x1 (degree very)))`, and the N1 daughter translates as `(lambda (x2) (lambda (wh1) (and (DESIRE x2) (FOR x2 wh1))))` then the translation of the mother (corresponding to the phrase *very crazy desire for* as in *what does Kim have a very crazy desire for*) would be (12a) which reduces to (12b).

(12) a
```
(lambda (x) (lambda (wh) (and
  ((lambda (x2) (lambda (wh1) (and (DESIRE x2) (ABOUT x2 wh1)))) x wh)
  ((lambda (x1) (CRAZY x1 (degree very))) x))))
```
b
```
(lambda (x) (lambda (wh) (and
  (and (DESIRE x) (FOR x wh))
       (CRAZY x (degree very)))))
```

ID rules may contain optional categories (enclosed in brackets) as in the rule in (13). Compilation causes such rules to be expanded as two separate rules, one which contains the optional category and one which does not. Each compiled rule has a unique name based on the original name: the one without the optional category will be marked with '/–' and the one with it will be marked with '/+'. So the result of compiling VP/PP/PASS will be two distinct object grammar rules, one named VP/PP/PASS/– and one named VP/PP/PASS/+. ID rules with optional categories have semantics defined for both possible expansions. In the case of VP/PP/PASS, the first semantic translation is for the version where the optional P2 is present and the second is for the version where it is absent. The system uses the mention of the number associated with the optional category, in this case, 3, as its guide for pairing the expanded rules with the appropriate semantic translation.

(13)
```
VP/PP/PASS : ; This and the next few rules to do some kinds of
             ; passive not produced by the passive metarule.
             ; This one does prepositional passives - he hates
             ; being looked at - the verb has to be SUBTYPE PVERB.
   VP[PAS, SLASH NOSLASH] --> H[SUBCAT PP, PFORM @pf, SUBTYPE PVERB],
   P[SUBCAT NP, PFORM @pf], ( P2[by, PRD -, SLASH NOSLASH] ) :
   (lambda (Q) (Q (lambda (e)
         (lambda (y) ((CP 1 2) e (3 (lambda (prep) (lambda (x) x)))
                y))) (lambda (e2) e2) (lambda (qu) qu))) :
   (lambda (Q) (Q
      (lambda (e) (lambda (y) ((CP 1 2) e (uq (some (x) (entity x)))
                 y))) (lambda (e2) e2) (lambda (qu) qu))).
```

ID rules may also contain categories marked with 'kleene plus' with the usual interpretation of one or more instances of this category. For example, the rule in (14), which is one which coordinates A2s, uses the kleene plus notation. The semantic translation uses index numbers marked with '+' (e.g. 1+) and this is interpreted as meaning 'each successive category which is an instance of 1'.

(14)
```
A2/COORD1 : A2[COORD +] --> ( A2[CONJ NULL] )+, ( A2[CONJ AND] )+ :
   0 = [SLASH NOSLASH], (lambda (x) (AND (1+ x))) :
   0 = [SLASH X2], (lambda (x) (lambda (wh) (AND (1+ x wh)))).
```

The rule in (14) allows for one or more A2s not preceded by a coordinator ([CONJ NULL]) to be followed by one or more A2s each preceded by the coordinator *and* ([CONJ AND]). This will generate all of the strings *crazy and eager*, *silly crazy and eager*, *silly and crazy and eager*.

2.1.6 Linear Precedence Rules

Linear Precedence (LP) rules encode permissible precedence relations in ID rules. The rule in (15a) places an ordering on [N +] (nominal or adjectival) categories, P2s and V2s (VP or S) such that [N +] categories precede all the others, P2s precede all but [N +] categories, etc. (15a) is the equivalent of one of the three LP rules in GPSG85. The LP rule in (15b) orders V2 categories with respect to one another.

(15) a
```
LPRULE LP2 : ; as in GPSG85 - nominal categories precede PPs which
             ; precede VPs and Ss.
   [N +] < P2 < V2.
```
b
```
LP6 : ; this orders V2s with respect to one another. Non-head V2s
      ; (i.e sentential and VP subjects) precede head VPs (idrule
      ; S/V2_SUBJ1 - S/V2_SUBJ5). The exception is tag questions
      ; where the non-head follows the head.
   V2[~H, ~TAG] < V2[H +] < V2[TAG, ~H].
```

The LP rules in the grammar ensure that almost all of the rules have a unique linearisation but there are a few cases where a rule has more than one possible linearisation. For example, there is a rule named N2+/APPOS which allows a limited form of apposition in N2s. The rule has two daughters, one of which is a name and one of which is a normal N2 and there is no LP rule which orders these categories. As a result, either order is possible and the rule generates both *Sidney, the killer whale* and *the killer whale, Sidney*. The process of compilation produces as many object grammar rules as there are linearisations—in the case of N2+/APPOS, two. Naming conventions give these object grammar rules unique names numbered consecutively, i.e. N2+/APPOS/1 and N2+/APPOS/2.

2.1.7 Phrase Structure Rules

Phrase structure (PS) rules encode both dominance and precedence relations in one rule. They are included in the formalism so that the expressive power of the system is not restricted by the Exhaustive Constant Partial Ordering (ECPO) property. (A grammar has the ECPO property if the set of expansions, defined by ID rules, of any one category observes a partial ordering that is also observed by the expansions of all the other categories, see GPSG85). The grammarian may choose not to use PS rules, or may choose to abandon the ID/LP format entirely in favour of PS rules, or may choose to use a mixed system. Our grammar uses ID/LP format predominantly but it does contain a few 'marked' PS rules, for example, some rules for cardinal numbers (see PS rules NUM1–NUM9) and some rules for names (PS rules N/NAME1–N/NAME2B), one of which is reproduced in (16). Apart from the ordering of daughter categories, PSRULES are defined and behave in exactly the same way as ID rules.

(16)
```
PSRULE N/NAME2B : ; sandy jones jnr
   N[-PLU, COUNT +, PN -, -POSS, SUBCAT NULL, ADDRESS -] -->
   N[+PN, SUBCAT NULL, ADDRESS -] N[ADDRESS +, SUBCAT NULL] :
   (and 1 (title 2)).
```

2.1.8 Feature Propagation Rules

Feature Propagation Rules define how features propagate between mother and daughter categories in ID or PS rules. The effect of propagation rules is to bind variables or instantiate values of features in rules of the 'object' grammar. Propagation rules can be used to encode particular feature propagation principles, such as the various versions of the Head Feature Convention proposed for GPSG. However, such principles are not 'hard-wired' into the formalism, in order that maximum flexibility and expressiveness can be maintained (see Briscoe *et al* 1987a,b for further discussion). The propagation rule in (17) propagates nominal head features between a nominal mother and its head daughter.

(17)
```
PROPRULE HFC_NOMINAL : ; head feature propagation for nominal categories
    [N +, V -] --> [H +], U. F(0) = F(1), F in NOMINALHEAD.
```

The rule is stated in terms of an ID rule pattern which prospective input ID rules must match. The pattern includes a variable over categories (U in this case, but W is used where the input must be a lexical ID rule (i.e. one with a lexical head)) and thereby specifies that the input must be any ID rule which has a nominal mother, contains a head and optionally contains other daughters of any type. Following the pattern is a statement of feature bindings. The terms 0 and 1 refer to categories in the input rule where 0 is the mother, 1 is the first daughter named by the pattern, 2 is the second daughter named by the pattern etc. The rule, then, is stating that the values of the features defined by the set NOMINALHEAD on the mother must be bound to the values of the same features on the head daughter.

(18) a
```
IDRULE N1/N : ; an N with no complements
    N1 --> H[SUBCAT NULL] : 1.
```

b
```
N1/N : ; an N with no complements
    [N +, V -, BAR 1, CONJ NULL, PRD @13, SLASH [NOSLASH +],
     NFORM @20, PER @21, PLU @22, COUNT @23, CASE @24, PN -,
     PRO -, PROTYPE @27, PART @28, DEF @, POSS -, ADV @37,
     NUM @38, WH NO, UB NO, EVER NO, MOD NONE, COORD -,
     REFL @45, DEMON @75, COADV -] -->
    [N +, V -, BAR 0, SUBCAT NULL, CONJ NULL, H +, PRD @13,
     NFORM @20, PER @21, PLU @22, COUNT @23, CASE @24, PN -,
     PRO -, PROTYPE @27, PART @28, POSS -, ADV @37, NUM @38,
     COORD -, REFL @45, DEMON @75] : 1.
```

(18) illustrates the effect of this propagation rule. (18a) is an ID rule to which the propagation rule can apply. (18b) is the expanded object grammar rule and some parts of it are the result of the application of the propagation rule: the features from the set NOMINALHEAD have been added to both the mother and the daughter (i.e. the features PLU, POSS, CASE, PRD, PN, PRO, PROTYPE, COUNT, NFORM, PER, REFL, NUM, DEMON, ADV, PART) and for each of these features they share the same value. In most cases these values are variables which will be instantiated at parse time. For example, the variable @22 which is the value of PLU on the daughter might get instantiated to '–' and the fact that this variable is also the value of PLU on the mother means that the mother will

also be specified as [PLU –]. A few of the NOMINALHEAD features have received actual values (e.g. [POSS –]) which are the same on mother and daughter and which result from an application of a default rule (see below) subsequent to the application of the propagation rule.

Propagation rules can also be used to bind feature values on categories which are the values of category-valued features, as in the agreement rule below, where F(2[AGR]) is to be read as 'features on the category which is the value of 2's AGR feature' and where AGRFEATS is a previously defined set comprising the features NFORM, PLU, COUNT, PER and CASE.

(19)
```
PROPRULE AGR/NP_VP : ; binds feature values on subject N2 to feature values
                     ; on the category that is the value of VP[AGR].
   S --> N2, H2[-SUBJ, AGR N2], U.
   F(1) = F(2[AGR]), F in AGRFEATS.
```

2.1.9 Feature Default Rules

Feature Default Rules default specified values onto features in categories in a particular environment in an ID or PS rule. They operate only in cases where a category in a rule has not acquired an actual value for some feature—they instantiate variable values and they also operate where no specification exists at all, but they never overwrite existing concrete values. Default rules replace Feature Specification Defaults in GPSG; because the rules assign values to features in the context of an ID or PS rule, their application can be accurately controlled, and thus the need for Feature Cooccurrence Restrictions to prevent the construction of 'illegal' categories is diminished (see Briscoe *et al* 1987b). Both Feature Default rules and Feature Propagation rules have a similar syntax to Kilbury's (1986) independently motivated Category Cooccurrence Restrictions, although their function is somewhat different. The default rule in (20a) states that a N2 complement that has not yet acquired a concrete value for CASE will be accusative. (20b) states that a non-adverbial A2 complement will be [PRD +]. Default rules use the variables U and W in the same way that propagation rules (and metarules) do—both range over any other material in the rule but W requires that the rule must be a lexical rule (i.e. have a lexical head).

(20) a
```
DEFRULE N2/CASE : ; defaults N2 complements (sisters
                  ; of lexical heads) to CASE ACC.
   [] --> N2, W. CASE(1) = ACC.
```
b
```
DEFRULE A2/PRD : ; A2 complements default to PRD +.
   [] --> A2[ADV -], W. PRD(1) = +.
```

2.1.10 Metarules

Metarules encode systematic relationships between sets of ID or PS rules by automatically adding further rules to the basic set produced by the grammar writer and the derived set produced by the application of other metarules. Metarules can be written to apply to ID rules or PS rules (including PS rules which result from the linearisation of ID rules). They can also be restricted to apply only to ID rules containing a lexical head through the use of the W category variable (as opposed to the U 'unrestricted' category variable) in the input pattern. The metarule in (21a) (comment omitted) adds new passive VP rules which are like their active counterparts except that the object N2 has disappeared and an optional P2[by] occurs. In fact, the output the user sees is an expansion of the output rule which has factored into two rules, one where the P2[by] is present and one where it is absent. The new rules are defined on the basis of the subset of existing VP rules which match the input pattern occurring before the `==>`. The ouput pattern (after the `==>`) defines the form of the new rules.

(21)
```
PASSIVE :
   VP[AGR N2[NFORM NORM], SLASH @] -->
   H[SUBCAT (NP, NP_NP, NP_PP, NP_PP_PP, NP_LOC, NP_ADVP, NP_SFIN,
            NP_NP_SFIN, NP_SBSE, NP_WHS, NP_WHVP, OC_NP, OC_AP,
            OC_INF, OC_ING, OC_PP_ING), SUBTYPE (NONE, DMOVT, EQUI),
      AGR N2[NFORM NORM], PSVE -], N2[-PRD, PRO @, SLASH @], W.
   ==>
   VP[EN, +PRD] --> H[PSVE +, EN, +PRD], ( P2[PFORM BY, PRD -] ), W :
   2 = [SLASH NOSLASH], 6 = [SLASH NOSLASH],
      (lambda (s) (s (lambda (prop) (lambda (ta)
             (lambda (equa) (lambda (Q) (Q
                      (lambda (e) (lambda (y) ((lambda (dsubj)
                                  ((lambda (2) (prop e dsubj)) y))
                               (2 (lambda (by) (lambda (np) np))))))
                      ta equa))))))) :
   2 = [SLASH NOSLASH], 6 = [SLASH X2], (lambda (s)
         (s (lambda (prop) (lambda (ta)
                (lambda (equa) (lambda (Q) (Q (lambda (e)
                         (lambda (y) (lambda (wh)
                               ((lambda (dsubj) ((lambda (2)
                                          (prop e dsubj)) y))
                                   (2 (lambda (by) (lambda (np) np))
                                     wh))))) ta equa))))))) :
   2 = [SLASH NOSLASH],
      (lambda (s) (s (lambda (prop) (lambda (ta) (lambda (equa)
                (lambda (Q) (Q
                      (lambda (e) (lambda (y) ((lambda (2) (prop e
                                 (uq (some (x1) (entity x1))))) y)))
                       ta equa))))))).
```

The semantics attached to the metarule defines functions which take the input semantics and transform them into semantics suitable for the output rule: specifically, they change the expectations about the order of arguments—the first argument of the verb is now to be found as the complement of the P2[by] and the second argument is to be found outside the VP altogether. (This may not be obvious given the complexity of the lambda expressions but it should become clearer when the translations associated with VPs are described in section 5.4.5.) The conditions on the semantics pair input conditions with output conditions. The categories in the metarule are numbered from 0 starting with the mother of the input pattern through, in this case, to 7, which is the W variable in the output pattern. The condition pair `2 = [SLASH NOSLASH], 6 = [SLASH NOSLASH]` means that the semantics associated with the condition `[SLASH NOSLASH]` on the N2 daughter of the input rule (2 in the metarule) is to be transformed by the function given into a semantics to be associated with a condition [SLASH NOSLASH] on the P2[by] in the output rule (6 in the metarule). The intuition behind the conditions is that just as the semantics of a VP differs according to whether it contains a gap or not, so will the semantics of passive versions of that rule. The first two pairs of conditions are ones which provide the semantics for the version of the output which contains a P2[by]: the first pair, `2 = [SLASH NOSLASH], 6 = [SLASH NOSLASH]`, deals with the case where there is no gap in the P2[by] in the output rule and the second pair, `2 = [SLASH NOSLASH], 6 = [SLASH X2]`, with the case where there is a gap in the P2[by]. (The metarule, together with other aspects of the grammar, ensures that the location of the gap will always be the object N2 in the input and the P2[by] in the output.) The third clause, `2 = [SLASH NOSLASH]`, provides the semantics for the version of the output which doesn't contain a P2[by] (and where other aspects of the grammar will ensure there is no gap.[2]). In this case, since there is no possibility of a gap, no condition is necessary on the output rule. In all three cases, the semantic form to be transformed is the one which is associated with no gap in the input rule.

The ID rule in (22) is one of the inputs to the PASSIVE metarule and the two resulting output rules are given in (23). Notice that each output from a metarule receives a unique name based on the input rule name, the metarule name and presence or absence of optional categories ('/+' and '/–').

(22)
```
VP/NP : ; abandons his friends
   VP --> H[SUBCAT NP], N2[-PRD] :
   2 = [SLASH NOSLASH], (lambda (Q) (Q (lambda (e) (lambda (x)
       (1 e x 2))) (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2], (lambda (Q) (Q (lambda (e) (lambda (x) (lambda (wh)
       (1 e x (2 wh))))) (lambda (e2) e2) (lambda (qu) qu))).
```

(23) a
```
VP/NP(PASSIVE/-) :
      VP[PAS, -AUX, -PRO, CONJ NULL, BEGAP -, FIN @11, PAST @12,
          NEG @17, SLASH NOSLASH, COORD -, AGR N2[NFORM NORM,
          PER @21, PLU @22, COUNT @23, CASE @24], ELLIP -] -->
      V[PAS, -INV, -AUX, -PRO, SUBCAT NP, CONJ NULL, H +, FIN @11,
          PAST @12, PSVE +, NEG @17, COORD -, SUBTYPE NONE,
         AGR N2[NFORM NORM, PER @21, PLU @22, COUNT @23, CASE @24]] :
    (lambda (Q) (Q (lambda (e) (lambda (y) (1 e (uq (some (x1)
      (entity x1))) y))) (lambda (e2) e2) (lambda (qu) qu))).
```

[2]This doesn't mean that gaps are disallowed in passives without a P2[by] but it does mean that the rules output from this metarule disallow them. Other metarules will produce 'gappy' versions—see, for example, metarules MSLASH4a–MSLASH4d.

```
b  VP/NP(PASSIVE/+) :
     VP[PAS, -AUX, -PRO, CONJ NULL, BEGAP -, FIN @11, PAST @12,
        NEG @17, SLASH @19, COORD -, AGR N2[NFORM NORM, PER @21,
        PLU @22, COUNT @23, CASE @24], ELLIP -] -->
     V[PAS, -INV, -AUX, -PRO, SUBCAT NP, CONJ NULL, H +, FIN @11,
        PAST @12, PSVE +, NEG @17, COORD @, SUBTYPE NONE,
        AGR N2[NFORM NORM, PER @21, PLU @22, COUNT @23, CASE @24]],
     P2[-PRD, -GER, -POSS, by, CONJ NULL, BEGAP @, NEG @, SLASH @19,
        PRO @, LOC @, WH NO, UB NO, EVER NO, COORD @, MODTYPE @] :
     2 = [SLASH NOSLASH],
        (lambda (Q) (Q (lambda (e) (lambda (y) (1 e (2 (lambda (by)
                         (lambda (np) np))) y))) (lambda (e2) e2)
              (lambda (qu) qu))) :
     2 = [SLASH X2], (lambda (Q) (Q (lambda (e) (lambda (y)
              (lambda (wh) (1 e (2 (lambda (by) (lambda (np) np)) wh)
                     y)))) (lambda (e2) e2) (lambda (qu) qu))).
```

2.1.11 Top Category Declaration

An optional declaration is a TOP definition. If one is defined, then the parser will only return those parses whose category is consistent with the TOP declaration. If it is left undefined then all possible parses will be returned. The TOP declaration in the grammar is given in (24).

(24) `TOP [T S], [T NP].`

2.2 Metagrammar Interpretation

In GPSG, the set of legal local trees is defined declaratively by simultaneous application of the various rule types during the 'projection' from ID rules to fully instantiated local trees. In the ANLT system, although the metagrammatical notation shares many similarities with GPSG, there are crucial differences in the interpretation of the notation. Metagrammatical rules jointly specify a set of partially instantiated phrase structure rules, rather than fully instantiated local trees. That is, rules are allowed to contain variable values for features and these variables are instantiated at parse time by unification. Propagation rules specify restrictions on the instantiation of these variables; for example, the expanded version of ID rule N2+/DET1a which builds NPs from determiners and nominal phrases will force agreement between the determiner and the nominal phrase by binding the PLU variables on all the categories as illustrated in (25) (irrelevant features, semantics and comment excluded).

(25)
```
N2+/DET1a : N2[+SPEC, PLU @22] -->
   DetN[AGR N2[PLU @22]], N2[-SPEC, H +, PLU @22].
```

If we assume that *this* is specified as [AGR N2[PLU –]], *these* is specified as [AGR N2[PLU +]] and *the* is specified as [AGR N2[PLU @4]], then matching these to the DetN in (25) will cause all three instances of @22 to be instantiated to – in the case of *this* and + in the case of *these* while it will remain uninstantiated in the case of *the*. Since *sandwiches* is specified as [PLU +] it can match the head category in (25) so long as the determiner is either *these* or *the* (and in the latter case *sandwiches* will cause @22 to be instantiated to +). *sandwiches* cannot match, however, if the determiner is *this* since @22 would need to be instantiated to – for the determiner but to + for *sandwiches*.

The object grammar is produced by applying propagation, default and category rules, in that order, to ID (and PS) rules and then applying the non-linear metarules one-by-one in a predefined order to the set of fleshed out ID rules. After each metarule has applied, propagation and default rules are applied to any new ID rules and these are added to the original set before the application of the next metarule. Next, the resulting expanded set of rules is linearised according to the LP rules and any linear metarules are applied to the complete pool of PS rules. This procedure is summarised in Figure 1.

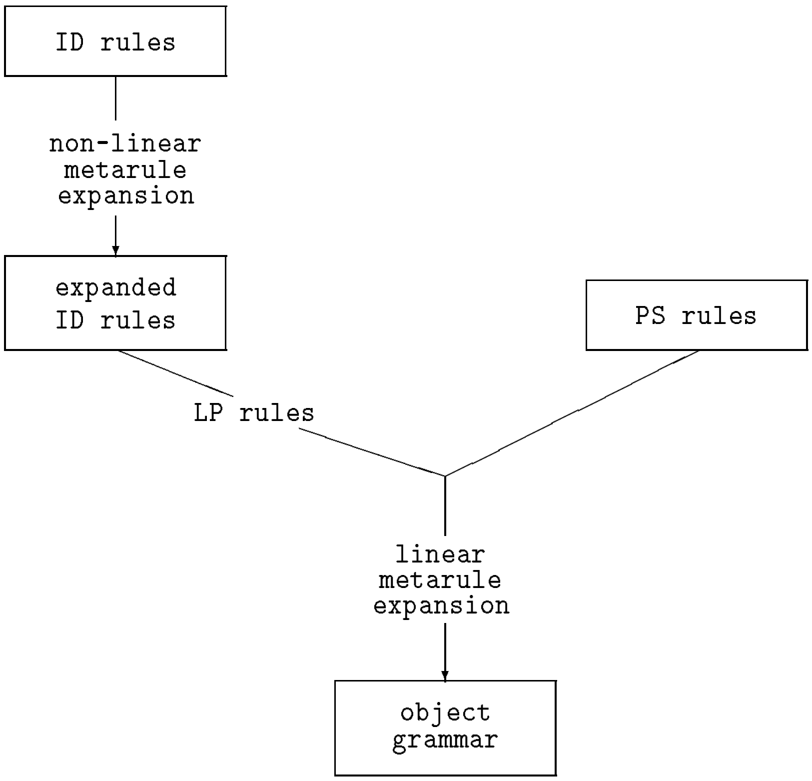


Figure 1. Metagrammar Compilation

3 Syntactic Features

In Section 3.1 we describe the feature system, explaining which features appear on which categories and what purpose they serve. In Section 3.2 we outline the way in which we achieve the kinds of feature propagation which, in GPSG, are dealt with by means of feature propagation principles such as the Head Feature Convention.

3.1 The Feature Make-up of Categories

3.1.1 Major Categories

Major categories are ones which are defined in terms of the features N, V and BAR. If a category has the features N and V it must also have BAR (although the opposite is not the case). N and V combine to define the four major category types: verbal categories are [V +, N –], nominal categories are [N +, V –], adjectival categories are [V +, N +] and prepositional categories are [V –, N –]. The feature BAR distinguishes lexical categories ([BAR 0]) from phrasal categories ([BAR 2]). [BAR 1] describes a level which is intermediate between lexical and phrasal. For each major category type there is a set of head features which must appear on all instances of that category type, regardless of their BAR feature value. In addition there are other features which must appear on only some instances of a category type depending on their BAR feature value or some other feature value. The tables in sections 3.1.1.1—3.1.1.4 indicate the distribution of features for the four major category types and following each table is a brief description of the purpose of most of the features mentioned. There are a few features which occur with all four major category types and it seems more appropriate to describe these separately. Section 3.1.1.7, therefore, describes features common to all major category types. In two cases the grammar uses categories which are specified for BAR but which do not have N and V specifications. These categories generalise across major categories which share some features in common but not all. The two cases are [PRD +] categories and [ADV +] categories and the features relevant to them are described in sections 3.1.1.5 and 3.1.1.6 repectively.

3.1.1.1 Verbal Categories

The feature specification [V +, N –] differentiates verbal categories from the other major categories. Lexical verbal categories (i.e. verbs) are [BAR 0]. Following the analysis in GPSG85 (which originates from Borsley (1983)), there are no [BAR 1] verbal categories and VP and S are both [BAR 2] (i.e. full phrasal projections). VP and S are differentiated by means of the feature SUBJ (to indicate the presence or absence of a subject): VP is [SUBJ –] and S is [SUBJ +]. In the grammar, and in this text, the symbols S, VP and V are aliases for the feature specifications at the beginning of Table 1.

Table 1 displays all the features which occur on verbal categories. The first four are shown with values since these are the feature value pairs which are used in the definition of the aliases S, VP and V. The next seven features (PRD, FIN, AUX, VFORM, PAST, AGR, PRO—marked in bold font) are head features for verbal categories (as defined by the set definition VERBALHEAD) and are propagated between mother and head in verbal rules by the propagation rule HFC_VERBAL. Features in parentheses are ones which occur on some instances of the category they are listed under, but not on all (for example, a few VPs are specified for the feature WH, but most are not). Features marked with an asterix are ones which play a similar role on several categories and which are described in Section 3.1.1.7.[3]

S	**VP**	**V**
[V +] [N –] [BAR 2] [SUBJ +]	[V +] [N –] [BAR 2] [SUBJ –]	[V +] [N –] [BAR 0]
VFORM	**VFORM**	**VFORM**
PRD	**PRD**	**PRD**
AUX	**AUX**	**AUX**
PAST	**PAST**	**PAST**
AGR	**AGR**	**AGR**
PRO	**PRO**	**PRO**
INV		INV
COMP		
UDC		
(TAG)		
		PSVE
	NEG	NEG
		(CONEG)
		*(PRT)
		*(PFORM)
		*(PREP)
		*SUBCAT
		*SUBTYPE
*CONJ	*CONJ	*CONJ
*COORD	*COORD	*COORD
*SLASH	*SLASH	
*BEGAP	*BEGAP	
*ELLIP	*ELLIP	
*WH	*(WH)	
*UB	*(UB)	
*EVER	*(EVER)	

Table 1: Verbal Features

VFORM **values: BSE, ING, EN, TO, NOT**

[3] PRD and PRO are features which occur on all or nearly all major categories but since they are also head features and since they reflect different linguistic phenomena according to which category they occur with, we describe them in each major category section instead of in Section 3.1.1.7.

The feature VFORM (verb form) distinguishes between a number of possible untensed forms. [VFORM BSE] marks the base or uninflected form of the verb as in:

(26) a Kim wants to *help*
b Kim may *help*
c Kim wants to *be* an astronaut
d They require that Kim *attend*

[VFORM ING] appears on all verbs with the *ing* suffix, whether present participle (27a) or gerund (27b).

(27) a Kim is *reading* the book
b Kim *reading* the book surprised me

[VFORM EN] appears on both the past participle and the passive form of the verb, for example:

(28) a Lee has *finished* his tea
b The mouse was *eaten* by the cat

[VFORM TO] is used to identify infinitive VPs. Following GPSG85, we treat the word *to* occuring in infinitives (*John wants* to *go*) as a main verb albeit of a rather marked type since it doesn't inflect in any way: *to* is the only verb in the dictionary specified as [VFORM TO] and this feature-value pair propagates up to the VP dominating it, thus ensuring that the VP immediately containing it is also marked as [VFORM TO].

[VFORM NOT] appears on all finite, non-imperative verb forms and is to be interpreted as meaning that the verb has no meaningful value for VFORM—the purpose of the VFORM feature is to distinguish between different types of non-finite forms and as such it has no real contribution to make to the specification of finite forms.

FIN **values: +, –**

The feature FIN (finite) distinguishes tensed forms of a verb ([FIN +]) from untensed forms ([FIN –]). FIN interacts with VFORM in the following way: in general [FIN +] verbs will be VFORM NOT whilst [FIN –] verbs will have one of BSE, ING, EN, TO as a VFORM value. The only exception to this are imperatives which are [+FIN, VFORM BSE], following the analysis of Warner (1984).

PRD **values: +, –**

The feature PRD (predicative) appears on major categories. When it appears on verbs, it interacts with the VFORM feature to distinguish between passives and past participles, and present participles and gerunds: passive verbs are [VFORM EN, PRD +]; past participles are [VFORM EN, PRD –]; present participles are [VFORM ING, PRD +]; gerunds are [VFORM ING, PRD –]. The linguistic motivation for this analysis stems from the observation that the verb *be* subcategorises for predicative complements. Passive verbs and present participles are both complements of *be*, therefore they are [PRD +]. Past participles and gerunds are not complements of *be*, therefore they are [PRD –].

AUX **values: +, –**

The feature AUX distinguishes auxiliaries from non-auxiliaries: the auxiliaries *be*, *have*, *do*, *will*, the modals *can*, *may*, *ought* etc, and the infinitive verb *to* are all [AUX +]. All other verbs are [AUX –].

PAST **values: +, –, FUT**

The feature PAST encodes past tense ([PAST +]), present tense ([PAST –]) and future tense ([PAST FUT]). All finite verbs are either [PAST +] or [PAST –] with the exception of the future marking auxiliaries *will* and *shall* which are [PAST FUT]. Since tense is not relevant to non-finite forms these are all marked as [PAST NOT] where this is interpreted as meaning that the verb has no meaningful value for PAST. Earlier versions of the grammar (before semantics were added) used PAST to distinguish between past and present tense morphology. (With the addition of semantics, there was a need to distinguish between past, present and future temporal reference and it was at this point that the value FUT was added. It would probably be more logical to replace the PAST feature by one called TENSE with values PAST, PRES, FUT but since there are many irregular entries in the large lexicon with specifications for the feature PAST which would need to be changed it was decided to keep the feature name PAST.)

AGR **category-valued**

AGR is a category-valued feature which encodes information about the category that verbs, adjectives or determiners agree with. The value of AGR is restricted such that only a limited set of features appear as part of the category value and this means that AGR in our grammar is really only 'pseudo' category-valued, unlike GPSG85 where the value of AGR is an exact copy of the category agreed with.

Verbs will be either [AGR N2] (agrees with an N2 subject) or [AGR V2] (agrees with an S or VP subject). These AGR values will be further instantiated:

The N2 in [AGR N2] has additional features defined by the set AGRFEATS (NFORM, PER, PLU, CASE, COUNT) since these are the ones relevant to agreement. The values for these features will be bound to the values for the same features on the subject N2. The NFORM feature is the one which defines whether a verb can agree with a dummy subject (*it*, *there*) or not.

The V2 in [AGR V2] has the additional features FIN, SUBJ, VFORM and UB. The values for these features will be bound to the values for the same features on the V2 subject. These features define the type of V2 subject (finite S, infinitival S, infinitival VP, base S and question S) that a verb can cooccur with (see ID rules S/V2_SUBJ1–S/V2_SUBJ5 and propagation rule AGR/V2_VP).

PRO **values: +, –**

Under certain conditions, auxiliary verbs can occur without their usual VP complements as in the examples in (29). In these cases, the form of the missing complement is understood to be the same as some previously occurring VP complement which acts as an antecedent for the missing (or ellided) VP. Our analysis of cases like this involves allowing the auxiliary to stand in place of itself and its missing complement: i.e. to analyse it as a pro-form. The feature PRO marks the relevant auxiliary entries in the lexicon and is picked up by rules for building [PRO +] VP constituents (see, for example, ID rules VP/PRO and VP/PRO2). All other verbs and VP rules are specified as [PRO –].

(29) a Kim wants to go to the zoo and Lee wants *to* too.
b Kim will have finished the book by Monday. Lee won't *have*.

INV **values: +, –**

The feature INV distinguishes inverted sentences from uninverted ones. The S rules which are output from the subject-auxiliary inversion (SAI) metarule are [INV +]. S and V are specified for INV, whilst VP is not. Non-auxiliary verbs are specified as [INV –] since they cannot invert. Most auxiliaries have a variable value for INV to allow them to appear in both inverted and non-inverted structures. Some, however, are always [INV +], e.g. the 1st person singular entry for *aren't* ([INV +, AGR N2[PER 1, PLU –]]) as in *aren't I clever*.

COMP **values: THAT, FOR, IF, WHETHER, NORM, ER, AS**

The COMP feature occurs on S and encodes the type of complementiser an S contains. For example, an S[COMP THAT] contains a *that* complementiser, an S[COMP WHETHER] contains a *whether* complementiser etc. An S with no complementiser is [COMP NORM].

UDC **values: +, –**

The feature UDC occurs only on S categories and indicates whether that S is an unbounded dependency construction or not (i.e. whether it contains an initial preposed constituent or not). This is used to constrain the distribution of such S categories: all complement S's are required to be [UDC –] in order to block examples such as those in (30). This may be a little too restrictive but it helps to cut down on spurious ambiguities.

(30) a ? Kim acknowledges the fact that S[Kim, Lee helped _].
b ? Sandy wonders whether [Kim, Lee helped _].

TAG **values: +, –**

The feature TAG is used to mark the tag in a tag-question (i.e. *isn't he* in *he is clever, isn't he*). We analyse the tag as a rather marked type of S (see ID rule S/TAG) and distinguish it as [TAG +]. No other S has the feature TAG.

PSVE **values: +, –**

All verbs have an extra feature PSVE (passive). This feature is necessary since there is nothing in the grammar to prevent passive verbs from matching active VP rules. (Passive verbs are distinguished by the feature specification [VFORM EN, PRD +] but since active VP rules are left unspecified for VFORM and PRD, these features cannot be used to prevent passive verbs from matching.) All passive verbs are therefore specified as [PSVE +] and a default rule (V/PASS) ensures that the lexical head of all passive VP rules is similarly specified. All active verbs are marked as [PSVE −] and this feature specification is also defaulted onto the heads of all active VP rules (default rule V/UNPASS). Without this feature **fido is attacked the dog* would get a successful parse.

NEG **values: +, −**

The feature NEG (negative) occurs on V and VP but not on S. Most lexical entries for verbs are [NEG −] but auxiliaries with the *n't* suffix are [NEG +]. VPs are [NEG +] either by virtue of containing a [NEG +] verb or because they are modified by *not* (see ID rule VP/NEG).

CONEG **values: +, −**

CONEG is a feature which only occurs on auxiliary verbs and which is used to constrain the possibility of them occurring with negative complements. There is a metarule, NEGATION, which creates new versions of the auxiliary rules which are just like the basic ones except that the word *not* is inserted between the auxiliary and its complement (*did/can not leave*). There is also an ID rule VP/NEG which allows *not* to precede a non-finite VP and this makes it possible for examples like *did/can not leave* to receive two parses. In the case of modals, this seems appropriate: *Kim can not leave* is ambiguous between a reading where the *not* scopes just over the complement (Kim is capable of not leaving) and one where it scopes over the modal and its complement (Kim is not able to leave). In the case of tense and aspect carrying auxiliaries, this ambiguity is not so readily available—*Kim did not leave* cannot really be said to be ambiguous. Lexical entries for non-negative auxiliaries receive a [CONEG −] specification by default and a propagation rule, AUX/CONEG, binds the value of CONEG on the auxiliary to the value of NEG on the VP complement in the ID rules which are responsible for the tense and aspect auxiliaries. This prevents a parse of *Kim does not leave* where the *not* forms a constituent with the VP complement. The rules for modal verbs are not affected by the propagation rule so the (somewhat redundant) presence of CONEG on modal verbs does not affect their ability to cooccur with a negative VP complement. There are also lexical entries for auxiliaries which are inherently negative—ones with *not* cliticised to them and these entries have a variable value for CONEG since they can cooccur with negative complements: *Kim didn't not help*, *Kim can't not help*. The semantics of negatives is decribed in more detail in section 6.6.

3.1.1.2 Nominal Categories

The feature specification [N +, V −] differentiates nominal categories from the other major categories. Lexical nominal categories (i.e. nouns) are [BAR 0]. There is a [BAR 1] nominal category and this is the level at which modifiers are attached. The [BAR 2] level is the phrasal projection although there are two distinct types of [BAR 2] nominal categories: ones which are [SPEC +] and ones which are [SPEC −]. It is N2[SPEC +] which corresponds to the standard category of NP. In the grammar, and in this text, the symbols N2, N1 and N are aliases for the feature specifications at the beginning of Table 2.

N2	**N1**	**N**
[N +] [V –] [BAR 2]	[N +] [V –] [BAR 1]	[N +] [V –] [BAR 0]
NFORM	**NFORM**	**NFORM**
PER	**PER**	**PER**
PLU	**PLU**	**PLU**
COUNT	**COUNT**	**COUNT**
CASE	**CASE**	**CASE**
POSS	**POSS**	**POSS**
PN	**PN**	**PN**
PRO	**PRO**	**PRO**
PROTYPE	**PROTYPE**	**PROTYPE**
REFL	**REFL**	**REFL**
DEMON	**DEMON**	**DEMON**
NUM	**NUM**	**NUM**
PART	**PART**	**PART**
ADV	**ADV**	**ADV**
PRD	**PRD**	**PRD**
SPEC		
QFEAT		
DEF	DEF	(DEF)
KIND		
AFORM		(AFORM)
COADV	COADV	
	MOD	
NEG		
		(ADDRESS)
		*(PRT)
		*(PFORM)
		*SUBCAT
*CONJ	*CONJ	*CONJ
*COORD	*COORD	*COORD
*SLASH	*SLASH	
*BEGAP		
*WH	*WH	*(WH)
*UB	*UB	*(UB)
*EVER	*EVER	*(EVER)

Table 2: Nominal Features

Table 2 displays all the features which occur on nominal categories. The first three are shown with values since these are the feature value pairs which are used in the definition of the aliases N2, N1 and N. The next fifteen features (NFORM, PER, PLU, COUNT, CASE, POSS, PN, PRO, PROTYPE, REFL, DEMON, NUM, PART, ADV, PRD—marked in bold font) are head features for nominal categories (as defined by the set definition NOMINALHEAD) and are propagated between mother and head in nominal rules by the propagation rule HFC_NOMINAL. Features in parentheses are ones which occur on some instances of the category they are listed under, but not on all. Features marked with an asterix are ones which play a similar role on several categories and which are described in Section 3.1.1.7.

NFORM **values: NORM, THERE, IT**

This feature distinguishes 'dummy' forms (*it*, *there*) from normal nouns; pleonastic *it* is specified as [NFORM IT], existential *there* is specified as [NFORM THERE], whilst all other nouns, pronouns and proper names are specified as [NFORM NORM]. NFORM is relevant to agreement since not all verbs and adjectives can have N2 subjects of all types.

PER **values: 1, 2, 3**

This feature indicates whether a noun, pronoun or proper name is 1st, 2nd or 3rd person.

PLU **values: +, −**

This feature indicates whether a noun is singular ([PLU −]) or plural ([PLU +]).

COUNT **values: +, −**

COUNT distinguishes mass nouns ([COUNT −]) from countable nouns ([COUNT +]).

CASE **values: NOM, ACC**

CASE indicates case marking and has two values, NOM (nominative) and ACC (accusative). Almost all lexical entries for nouns have variable values for CASE since common nouns and names do not inflect for case. Pronouns, however, do exhibit case-marking and so these do have a specification for CASE.

POSS **values: +, −**

The feature POSS encodes possessiveness. All noun entries are [POSS −] except for the entry for the possessive morphemes *'s*, *s'* and *'* which are [POSS +]. The grammar contains special rules for building a possessive N2 out of a non-possessive N2 and a possessive morpheme (ID rules N2/POSSa–N2/POSSc). The morphological analyser is not able to separate the possessive morpheme from the word it attaches to, so a possessive N2 has to be entered to the parser with a space between the N2 and the possessive morpheme (i.e. instead of *the cat's* one must type *the cat 's*.

PN **values: +, −**

The feature PN distinguishes proper names ([PN +]) from other nouns ([PN −]).

PRO **values: +, −**

The feature PRO distinguishes pronouns ([PRO +]) from other nouns ([PRO −]). Although pronouns are often thought of as pro-NPs rather than pro-nouns, we give them [BAR 0] entries in the lexicon and then use ID rules to build pro-N2s. This allows postmodifiable pronouns to be simply treated and helps to keep [BAR 2] entries out of the lexicon (something that is desirable for the LR parser).

PROTYPE **values: NONE, COMPOUND, PMOD, PMOD+, THAT**

PROTYPE distinguishes between different types of pronoun. [PROTYPE NONE] pronouns are ones such as the personal pronouns which do not generally permit postmodification. The other values indicate that modification is possible. [PROTYPE COMPOUND] pronouns are a compound of a quantifier plus a place/time/thing indication (*something*, *anywhere*). They can be postmodified (*somewhere in France*) and they are also the only nominals which can be postmodified by a simple adjective (*somewhere quiet*: see ID rule N1/POST_APMOD2). [PROTYPE PMOD] pronouns are bare quantifiers or deictics (*all*, *this*) which can be postmodified (*all that you do*). [PROTYPE PMOD+] pronouns are similar but can additionally occur with a determiner: *the few that were left*. [PROTYPE THAT] singles out the pronoun *that* which because it is a homograph of the complementizer *that*, the relative pronoun *that* and the determiner *that* gives rise to a great deal of spurious ambiguity. Treating it in a class of its own enables us to reduce this ambiguity to an extent.

REFL **values: +, –**

REFL indicates whether a noun has reflexive morphology ([REFL +] – *myself*, *himself*) or not ([REFL –]). The default is [REFL –].

DEMON **values: +, –**

This feature indicates whether a pronoun is demonstrative or not. It is used to constrain the possibilities of postmodification of demonstrative pronouns (see ID rules N1/RELMOD2 and N1/RELMOD3).

NUM **values: CARD, ORD, +, –**

The feature NUM is used to encode whether a noun is a cardinal or ordinal number or neither. All noun entries are [NUM –] since there are no nominal entries in the lexicon for cardinals or ordinals. Instead, cardinal and ordinal nominal constituents are built from cardinal and ordinal A2s (see ID rules N/NUMBER1, N/NUMBER2 and N2+/ADJ4).

PART **values: OF, OF2, NO_OF, –**

The grammar contains rules for several different kinds of partitive construction (ID rules N2+/PART1–N2+/PART6B). Each of these rules builds an N2[+SPEC] from an initial N2 or A2 (for example, *some*, *several*, *three*, *few*, *a pound*) followed by *of* followed by an N2[+SPEC] (e.g. *the biscuits*). Occasionally the *of* need not occur (*all/both the biscuits*). The initial N2 or A2 is required to be specified for the feature PART with values OF, OF2 or NO_OF and this constrains the type of constituent that can occur in this position. Most nouns and noun positions are marked by default as [PART –].

ADV **values: +, –**

Nouns such as *time*, *year*, *second* can head N2s which can be used adverbially: *I saw him last year/ten seconds ago.* Such nouns are marked as [ADV +] and, by default, all others are [ADV –].

PRD **values: +, –**

The feature PRD encodes whether an N2 is being used predicatively or not and the semantic translation differs considerably according to this distinction. Noun entries have a variable value for PRD which will become instantiated according to the N2 they head (indefinites can be [PRD +] but definites cannot). Furthermore N2s will be required to be either [PRD +] or [PRD −] by rules that licence them (for example ID rules VP/BE_PRD and VP/BE_NP). Most N2 complements of verbs will be required to be [PRD −]—i.e. this is the default.

SPEC **values: +, −**

The feature SPEC (specifier) appears on N2 but not on N1 or N, thereby creating two distinct types of N2 (N2[+SPEC] and N2[−SPEC]). The distinction between the two types of N2 was introduced because of the need to recognise more levels of structure while keeping N2 as the phrasal projection. In earlier versions of the grammar, most N2 positions permitted either N2[+SPEC] or N2[−SPEC] to appear but in this version only N2[+SPEC] is allowed to occur as the complement of other categories and N2[−SPEC] now only occurs as the daughter of N2[+SPEC].

QFEAT **values: +, −, NO**

One of the motivations for the +/− SPEC distinction was the fact that there are two distinct levels at which quantifiers can attach: in determiner position (*all the books*) and below determiner position (*the many books*) i.e. under N2[SPEC +] and N2 [SPEC −] respectively. The feature QFEAT appears on an N2[−SPEC] and indicates whether it contains one of these low attaching quantifiers or not. The feature is needed because the semantic translation will differ considerably depending on the presence or absence of such a quantifier.

DEF **values: +, −**

This feature encodes definiteness. DEF appears on all N2s and N1s. An N2 acquires its DEF value from the DEF value of its determiner (see proprule PDEF3). DEF also occurs on N[PRO +] but not on any other N. Pronouns can be either definite or indefinite (e.g. *some* is indefinite whilst *all* is definite). DEF is used to restrict the distribution of some N2s: for example, the N2 complement occurring with existential *there* (*there is a mouse in the bathtub*) must be indefinite.

KIND **values: +, −**

The feature KIND occurs only on N2 and is used to allow for a 'natural kind' interpretation of non-predicative indefinites, e.g. *lions* in *lions are dangerous*. There are not many positions where there is an absolute requirement that an indefinite N2 should or should not have a 'kind' interpretation but this feature is used to that end in one or two rules (see ID rules N2+/COMPAR*, N2+/N2-b, N2/POSSb etc.).

AFORM **values: ER, EST, AS, NONE**

The feature AFORM (adjective form) reflects whether an adjective is comparative, equative, superlative or none of these. The feature has to appear on N2 as well as on adjectival categories since the rules for comparatives need to know whether an N2 contains a comparative quantifier or not. For example, in *Lee ate more chocolates than Kim ate*, the N2 *more chocolates* has to be marked for AFORM in order to licence the *than*-clause. Pronouns are also marked for AFORM since these can be comparative/superlative (*more, most*).

COADV **values: +, –**

This feature is used to constrain the distribution of words such as *ago* and *later* which can only post-modify [ADV +] nouns. Adjectival categories and non-lexical nominal categories have the feature COADV and propagation rules such as COADV1–COADV4 ensure a pattern of bindings which achieve an appropriate distribution.

MOD **values: PRE, POST, NONE**

The feature MOD (modified) appears only on N1. It is used to control the order in which modifiers attach to N1 (pre-modifiers attach lower than post-modifiers). It is simply a device to prevent multiple parses reflecting syntactically spurious attachment ambiguities in N2s which contain both pre- and post-modifiers. With the addition of semantics, there is less justification for this feature since low vs high attachment could reflect scope. However, since modification of nominals is currently represented as simple conjunction with no particular scoping intended, the feature has been retained.

NEG **values: +, –**

NEG appears on N2 but not on N1 or N. A class of negative N2s (ones modified by *not*) is recognised in the grammar but their distribution is restricted such that they only appear in coordinations of N2 (e.g. *a pencil but not a pen*). The *not* occurring between copula 'be' and an N2 (e.g. *fido is not a cat*) is not treated as being in constituency with the N2.

ADDRESS **values: +, –**

The specification [ADDRESS +] occurs on nouns which are titles such as *Ms*, *Sir*, *jnr* etc. This feature does not occur on any other noun and so the distribution of such nouns is highly constrained—they can only occur in conjunction with proper names (see PS rules N/NAME2A and N/NAME2B).

3.1.1.3 Adjectival Categories

The feature specification [N +, V +] differentiates adjectival categories from the other major categories. Lexical adjectival categories (i.e. adjectives and adverbs) are [BAR 0]. There is a [BAR 1] adjectival category and this is the level at which degree specifiers are attached. The [BAR 2] level is the phrasal projection. Adverbs are treated as a special kind of adjective so the feature specifications [ADV +] and [ADV –] have a very distinctive role to play. In the grammar, and in this text, the symbols A2, A1 and A are aliases for the feature specifications at the beginning of Table 3.

Table 3 displays all the features which occur on adjectival categories. The first three are shown with values since these are the feature value pairs which are used in the definition of the aliases A2, A1, and A. The next ten features (ADV, QUA, AGR, PRD, GRADE, NEG, DEF, NUM, PART, COADV—marked in bold font) are head features for adjectival categories (as defined by the set definition ADJHEAD) and are propagated between mother and head in adjectival rules by the propagation rule HFC_ADJ. Features in parentheses are ones which occur on some instances of the category they are listed under, but not on all. Features marked with an asterix are ones which play a similar role on several categories and which are described in Section 3.1.1.7.

A2	A1	A
[N +] [V +] [BAR 2]	[N +] [V +] [BAR 1]	[N +] [V +] [BAR 0]
ADV	**ADV**	**ADV**
QUA	**QUA**	**QUA**
AGR	**AGR**	**AGR**
PRD	**PRD**	**PRD**
GRADE	**GRADE**	**GRADE**
NEG	**NEG**	**NEG**
DEF	**DEF**	**DEF**
NUM	**NUM**	**NUM**
PART	**PART**	**PART**
COADV	**COADV**	**COADV**
AFORM	AFORM	AFORM
DISTR	DISTR	DISTR
		*(PRT)
(PFORM)		*(PFORM)
		*SUBCAT
		*SUBTYPE
*CONJ	*CONJ	*CONJ
*COORD	*COORD	*COORD
*SLASH	*SLASH	
*BEGAP		
*WH	*WH	
*UB	*UB	
*EVER	*EVER	

Table 3: Adjectival Features

ADV **values: +, –**

The feature ADV distinguishes adverbs from adjectives. For example, *clever* is an A[ADV –] while *cleverly* is an A[ADV +].

QUA **values: +, –**

The feature QUA distinguishes quantifying adjectives and adverbs ([QUA +]: *all*, *many*, *some*, *both*, *often*) from non-quantifying ones ([QUA –]: *big*, *stupid*, i.e. most adjectives and adverbs). Quantifying adjectives are seen as a special case since they occur in positions not available to other adjectives and since they function semantically as quantifiers. Quantifying adverbs (e.g. *often*, *frequently*) also have a special distribution and semantics.

AGR **category-valued**

All adjectives have an AGR specification and most behave exactly like verbs with respect to the AGR feature—they are either [AGR N2] or [AGR V2]. The exception is predicative quantifying adjectives, such as *many* or *few*, which can occur either in predicative position or prenominally as quantifiers. When they occur in predicative position they must agree with an N2 but when they occur prenominally they must agree with an N1. For this reason the BAR feature on the [N +, V −] category value of AGR on the lexical entries for such adjectives has been left unspecified. Non-predicative quantifying adjectives, such as *all*, attach under N2[+SPEC] (*all books*) and some of these are able to cooccur with with an N2[−SPEC] sister which already contains a quantifier (*all three books*) while others cannot (**both two books*). For these adjectives, QFEAT is added to the set of features on the category value of AGR in order to achieve a correct distribution. QFEAT occurs as part of AGR on determiners as well. Since adverbs are a kind of adjective, they also have the feature AGR. This specification is somewhat redundant since adverbs do not enter into agreement relations with other categories.

PRD **values: +, −**

The feature PRD, in combination with the feature DISTR, controls the distribution of adjectives and adjective phrases. In the lexicon, some adjectives are marked as [PRD −] because they can only head APs which occur in attributive position, for example, *main* (*the main reason*, **the reason is main*), and others must be [PRD +], for example, *asleep* (*the man is asleep*, **an asleep man*). Quantifying adjectives fall into two classes depending on whether they are [PRD +] or [PRD −]: *all* and *some* are A[QUA +, PRD −] while *many* and *few* are A[QUA +, PRD +]. Each type attaches at a different level inside an N2 and the [PRD −] type cannot occur in predicative position.

GRADE **values: +, −**

This feature distinguishes adjectives and adverbs which are 'gradable' from those which are not. Gradable adjectives are ones which can participate in comparative constructions (*more stupid than Lee*) and which can occur with degree specifiers (*so stupid*). Two other classes of adjectives are the truly intersective ones such as *four-legged* and non-restricting ones such as *bogus* and *alleged*. These classes are not distinguished in the large lexicon and we therefore only distinguish between gradable and non-gradable adjectives.

NEG **values: +, −**

NEG appears as a feature on all major categories but it is only in the case of adjectives that it is a head feature. This is because some quantifying adjectives are inherently negative, for example, *no* and *neither* etc. A2s modified by *not* (*not clever*, *not many*, *not often*—see ID rule A2/NEG and A2/NOT) are also [NEG +]

DEF **values: +, −**

When quantifying adjectives occur as N2 specifiers the DEF value of the N2 is derived from their DEF value. For example, *many* in *many books* is indefinite and so is the containing N2, and *both* in *both books* is definite and so is the containing N2.

NUM **values: CARD, ORD, −, +**

Cardinal numbers are described in the lexicon with features which are outside of the usual feature system (CN1, CN2, CN3, CN4, AND) and PS rules NUM1–NUM9 combine them to make more complex cardinals. ID rules A/NUMBER1 and A/NUMBER2 recognize cardinals as quantifying adjectives (*three books*) where the A category is marked as [NUM CARD]. Ordinals are treated as ordinary adjectives (*the fourth book*) except that they are marked as [NUM ORD]. Other adjectives and adverbs are [NUM –].

PART **values: OF, OF2, NO_OF, –**

PART plays the same role on adjectival constituents as it does on nominal ones: adjectives which can head the initial A2 in a partitive (e.g. *all* in *nearly all (of) the books*) are specified as [PART OF] or [PART NO_OF]. All other adjectives and adverbs and A2 positions are marked by default as [PART –].

COADV **values: +, –**

This feature is marks adjectives such as *ago* and *later* which can post-modify only [ADV +] nouns. The propagation rules COADV1–COADV4 ensure a pattern of bindings which achieve an appropriate distribution.

AFORM **values: ER, EST, AS, NONE**

The feature AFORM (adjective form) encodes information about whether an adjectival constituent is comparative ([AFORM ER]), equative ([AFORM AS]), superlative ([AFORM EST]), or none of these ([AFORM NONE]). It appears on A2, A1 and A but has not been treated as a head feature, since sometimes the AFORM value of an A2 derives from the lexical head (*bigger*, *biggest*) and sometimes it derives from the DetA (*more stupid*, *as stupid*, *most stupid*). Propagation rules A2/AFORM, A1/AFORM1 and A1/AFORM2 ensure an appropriate pattern of propagation.

DISTR **values: PRD, ATT, PST, PREPP**

DISTR (distribution), in combination with PRD, controls the distribution of adjective phrases. There are three positions in which non-quantifying, non-adverbial A2s can occur: prenominal modifer position, postnominal modifier position and predicative positions (as a complement to a verb). It is not the case that all A2s can occur in all three positions. A2s containing complements or a 'than/as' clause of comparison cannot occur in prenominal position (**the happy to see you man*), most A2s which do not contain complements or 'than/as' clauses cannot occur in postnominal position (**the man happy*), but the presence or absence of complements does not affect the ability of APs to occur in predicative position (*the man is happy to see you, the man is happy*). The grammar reflects these facts by marking the three positions as follows: prenominal = [DISTR ATT, PRD –], postnominal = [DISTR PRD, PRD +] and predicative = [DISTR @, PRD +]. The rules that build A2s, with or without complements, are then suitably marked for the features PRD and DISTR so that the resulting A2s will only occur in appropriate places. The presence of a degree specifier also affects the specification for DISTR: *so* prevents the A2 from occurring prenominally (**a so happy woman*) but *rather* does not (*a rather happy woman*).

3.1.1.4 Prepositional Categories

The feature specification [N –, V –] differentiates prepositional categories from the other major categories. Lexical prepositional categories (i.e. prepositions) are [BAR 0]. There is a [BAR 1] prepositional category and the [BAR 2] level is the phrasal projection. In the grammar, and in this text, the symbols P2, P1 and P are aliases for the feature specifications at the beginning of Table 4.

Table 4 displays all the features which occur on prepositional categories. The first three are shown with values since these are the feature value pairs which are used in the definition of the aliases P2, P1, and P. The next five features (PFORM, PRD, PRO, LOC, MODTYPE—marked in bold font) are head features for prepositional categories (as defined by the set definition PREPHEAD) and are propagated between mother and head in adjectival rules by the propagation rule HFC_PREP. Features in parentheses are ones which occur on some instances of the category they are listed under, but not on all. Features marked with an asterix are ones which play a similar role on several categories and which are described in Section 3.1.1.7.

P2	**P1**	**P**
[N –] [V –] [BAR 2]	[N –] [V –] [BAR 1]	[N –] [V –] [BAR 0]
PFORM	**PFORM**	**PFORM**
PRD	**PRD**	**PRD**
PRO	**PRO**	**PRO**
LOC	**LOC**	**LOC**
MODTYPE	**MODTYPE**	**MODTYPE**
POSS	POSS	
GERUND	GERUND	GERUND
NEG		NEG
		*(PREP)
		*SUBCAT
*CONJ		*CONJ
*COORD	*COORD	*COORD
*SLASH	*SLASH	
*BEGAP		
*WH	*WH	*(WH)
*UB	*UB	*(UB)
*EVER	*EVER	*(EVER)

Table 4: Prepositional Features

PFORM **values: numerous—see grammar listing**

The feature PFORM (prepositional form) encodes the type of preposition appearing in a P2. For example, the preposition *with* is marked as [PFORM WITH] and this information passes from the preposition to the P2 mother. This enables categories to subcategorize for particular prepositions: for example, the verb *give* subcategorises for a P2[PFORM TO].

One value for PFORM which is not the name of a preposition is NORM. This value is usually used for P2s which occur in modifying positions where there is no requirement for a particular preposition. Since most prepositions can head both subcategorized P2s and modifier P2s, there are at least two lexical entries for most prepositions, one which reflects which preposition it is (e.g. [PFORM WITH]) and one which characterises it as [PFORM NORM]. In this way we capture the distinction between subcategorized prepositions and modifier ones. Some verbs subcategorize for more than one P2, e.g. *Kim talked about the proposal with her colleagues*, and, in the absence of list-valued features, our solution involves special double values for PFORM, in this case, [PFORM ABOUT_WITH]. The prepositions *about* and *with* have entries with this specification in order for the VP rule to pick them up.

PRD **values: +, –**

As with all major categories, PRD appears on prepositional categories. Here it is used to distinguish between 'case-marking' prepositions ([PRD –]) and 'contentful' ones ([PRD +]). Prepositions such as *to* in *Kim gives a book to Lee* are generally considered not to contribute any information to the semantic translation. The semantic translation rules therefore pick up on the [PRD –] specification and ignore the *to*, returning just the meaning of its N2 complement. [PRD +] P2s receive a translation where the preposition is a relation over two arguments, the second one being the N2 complement of the preposition.

PRO **values: +, –**

The feature PRO identifies prepositional pro-forms such as *where*, *there*, *now*, *then*, *when* etc. These are marked as [PRO +] and dealt with by special rules for building pro-P2s, while all other prepositional categories are marked as [PRO –].

LOC **values: +, –**

The feature LOC distinguishes between locative and non-locative prepositional categories. It is necessary because some lexical heads subcategorise not for P2s headed by a particular preposition but for P2s which are locative. For example *put* takes a P2 complement which may be headed by any of a range of distinct prepositions (*on/under/in*, etc.) so long as they are locative.

MODTYPE **values: NML, VBL**

MODTYPE helps to avoid some spurious attachment ambiguities by encoding whether a P2 headed by a particular preposition can be a modifier of verbal categories ([MODTYPE VBL]) or a modifier of nominal categories ([MODTYPE NML]). Most prepositions head P2s which can modify either V2 or N2 and so these entries have a variable value for MODTYPE. A few types of P2 can only modify one of V2 or N2, for example, *because* taking a finite S complement cannot modify an N2 (**the biscuit because you like them*) so it is marked as [MODTYPE VBL].

POSS **values: +, –**

The feature POSS appears on P2 and P1, but not on P. It is used to identify possessive P2s such as *of Lee's* in *a book of Lee's*—a P2 or P1 will be [POSS +] if it contains a possessive N2 and [POSS –] otherwise.

GERUND **values: +, –**

The feature GERUND appears on P2 and P1 but not on P and is used to identify a P2 containing a gerundive complement; for example, *despite (Lee) arriving on time*. A P2 or P1 will be [GERUND +] if it contains a gerund and [GERUND –] otherwise.

NEG **values: +, –**

NEG appears on P2 and on P but not on P1. A class of negative P2s and Ps (ones modified by *not*) is recognised in the grammar but their distribution is restricted such that they only appear in coordinations of P2 (*on the table but not under it*) and P (*on but not under the table*). The *not* occurring between copula *be* and a P2 (e.g. *fido is not in his kennel*) is not treated as being in constituency with the P2.

3.1.1.5 Predicative Categories

There are places in the grammar where it is convenient to group phrases together which differ in major category features but which have some other feature in common. One such case concerns the position after *be* which can be filled by any kind of predicative category: *Kim is helping/helped* (VP); *Kim is crazy* (A2); *Kim is in the kitchen* (P2); *Kim is a genius* (N2). Moreover, these can be coordinated irrespective of whether they have the same values for the features N and V: *Kim is in the kitchen and helping* (P2 and VP); *Kim is a genius and proud of it* (N2 and A2). In order to describe this more succinctly a category with the alias X2 ([BAR 2]) is used. This category does not have the features N and V but it does count as a major category because it has the feature BAR and because X2[+PRD] dominates [PRD +] major categories (see ID rules PRD1–PRD4). There are no [BAR 1] and [BAR 0] versions of X2—it is always a category which 'sits on top of' other more specified categories.

X2[+PRD] occurs with a few other features: AGR, NEG, CONJ, COORD, SLASH, BEGAP, ELLIP, WH, UB, EVER. NEG is inherited from the daughter, and so is AGR if the daughter is VP or A2. Since N2 and P2 are not specified for AGR there is no propagation between X2[+PRD] and N2 and P2 daughters. The other features are all ones which occur on most major categories and which are described in section 3.1.1.7 below.

3.1.1.6 Adverbial Categories

Another place where it is convenient to group differing major categories is some adverbial positions. Each of the four major category types can be used adverbially: *having waited an hour, Kim left* (VP); *on Sunday Kim left* (P2); *the next day Kim left* (N2); *Angrily Kim left* (A2). To an extent these can also be coordinated without regard for the features N and V: *Kim left quickly but taking care to shut the door* (A2 and VP); *Kim left either last week or at the weekend* (N2 and P2). Following the same strategy as with predicatives, the alias X2 is used. Again X2 does not have the features N and V but X2[+ADV] dominates [ADV +] major categories (see ID rules X2/MOD1–X2/MOD4).

X2[+ADV] occurs with the following other features: QUA, LOC, PRD, NEG, CONJ, COORD, SLASH, BEGAP, ELLIP, WH, UB, EVER. It inherits QUA from an A2 daughter, but with all other daughters an X2[+ADV] is [QUA –]. LOC and PRD are inherited from a

P2 daughter and with all other daughters they are left unspecified. NEG is inherited from all daughters. The other features are all ones which occur on most major categories and which are described in section 3.1.1.7 below.

3.1.1.7 Features Common across Major Category Type

[BAR 2]	[BAR 1]	[BAR 0]
		(PRT)
(PFORM)		(PFORM)
		(PREP)
		SUBCAT
		(SUBTYPE)
CONJ	(CONJ)	CONJ
COORD	COORD	COORD
SLASH	SLASH	
BEGAP		
(ELLIP)		
(WH)	WH	(WH)
(UB)	UB	(UB)
(EVER)	EVER	(EVER)

Table 5: Features Common across Major Categories

PRT **values: numerous—see grammar listing**

The feature PRT is used to describe particles but it also occurs on those verbs, adjectives and nouns which need to cooccur with a particle (*fall over*, *full up*, *rummage around*). (Verbs occurring with particles are generally referred to as 'phrasal verbs'.) When PRT occurs on a lexical head, it describes which particle that head is to occur with. For example, the verb *fall over* has an entry under the head word *fall* and this entry contains the feature specification [PRT OVER]. This ensures that it is recognised as a phrasal verb and that it only occurs with the particle *over*. In rules which introduce phrasal verbs, adjectives and nouns (see any rule with 'PHR' as part of its name) the value for PRT on the head is bound to the value for PRT on the particle. Non-phrasal verbs, adjectives and nouns do not have the feature PRT.

PFORM **values: numerous—see grammar listing**

PFORM is a feature which describes prepositions but it also occurs on verbs, adjectives and nouns which subcategorise for a P2 complement of a particular type. Its function on these lexical heads is to ensure that they occur with the right preposition. For example, one subcategorisation for the verb *agree* is for a P2[PFORM WITH] (*I agree with you*), so the entry for this use of *agree* has the feature specification [PFORM WITH]. The rules which introduce P2 complements bind the value of PFORM on the head to the value of PFORM on the P2 to ensure correct pairings. Only verbs, adjectives and nouns which subcategorise for a P2 have a specification for PFORM.

There is one place where PFORM occurs on a non-lexical, non-prepositional category: on the A2 mother in ID rule A2/THAN/AS. This rule generates strings such as *than annoyed*, *as annoyed* which occur in comparatives like *more angry than annoyed* and *as much angry as annoyed*. The presence of PFORM is necessary in order to achieve the correct *er....than* or *as....as* pairings.

PREP **values: numerous—see grammar listing**

Some verbs occur with a free-floating preposition (one that is not part of a P2) such as *on* in *I was banking on seeing him*. The feature PREP encodes the nature of this preposition and functions in exactly the same way as the feature PFORM on verbs. Only verbs which subcategorise for this kind of preposition have a specification for PREP.

A few prepositions subcategorise for particular P2 complements, e.g. *because* subcategorises for a P2[PFORM OF] (*because of his bad leg*). Because these prepositions already have the feature PFORM to describe themselves, this feature can't be used to constrain the type of their P2 complements. Instead the feature PREP is used: *because* has an entry as P[SUBCAT PP, PFORM NORM, PREP OF].

SUBCAT **values: numerous—see grammar listing**

All lexical major categories are marked for the feature SUBCAT. This feature describes which complements a lexical head subcategorises for. The heads in all lexical ID rules have a non-variable value for SUBCAT and by this means we ensure that major category lexical entries can only match rules which introduce appropriate complements. For example, *accept* has three entries corresponding to three different subcategorisations: [SUBCAT NULL] (*I accept*—simple intransitive), [SUBCAT NP] (*Lee accepted Kim's apology*—simple transitive) and [SUBCAT SFIN] (*Lee accepted that it wasn't Kim's fault*—sentential complement). The ID rules VP/INTR, VP/NP and VP/SFIN1 are responsible for building intransitive, transitive and sentential-complement-taking VPs respectively and the heads in these rules are specified as [SUBCAT NULL], [SUBCAT NP] and [SUBCAT SFIN] repectively. This means that each instance of *accept* can only occur with just the complements it requires.

SUBTYPE **values: numerous—see grammar listing**

All major categories have the feature SUBCAT but verbs and adjectives are additionally specified for the feature SUBTYPE which provides further information about a particular subcategorisation. For example, the verbs *tell* and *bother* can both occur with an N2 object and a sentential complement (*he tells me that it will rain* and *it bothers me that it will rain*) and both have the SUBCAT value NP_SFIN. They are, however, very different from one another—*tell* has normal agreement properties whilst *bother* agrees with the dummy NP *it* and, furthermore, it is an 'extraposition' verb which means it can also occur with a sentential subject as in *that it rains bothers me*. To distinguish these two verbs, we use the feature SUBTYPE with values NONE and EXTRAP respectively and this feature is picked up by the appropriate ID rules (see VP/NP_SFIN1, VP/NP_SFIN2A and VP/NP_SFIN2B). Often SUBTYPE doesn't really have a role to play since not all SUBCAT types have variants—in this case the default SUBTYPE NONE is used.

SUBTYPE operates on adjectives in exactly the same way as it operates on verbs in that it provides extra distinctions between adjectives that have the same SUBCAT value. For example, many adjectives take an infinitival VP complement and are marked as [SUBCAT VPINF], but there are differences between them—*easy* is marked as [SUBTYPE TOUGH] because it is a 'tough-movement' adjective (its VP complement contains an object gap: *he is easy to beat _*), while *normal* is marked as [SUBTYPE EXTRAP] because it allows extraposition of its VP complement (*to stare wildly at people is not normal, it is not normal to stare wildly at people*).

CONJ values: NULL, BOTH, NEITHER, EITHER, AND, OR, NOR, BUT

Within a coordinate structure, conjuncts preceded by a coordinating word such as *and*, *but*, *neither* etc. are given an appropriate value for the feature CONJ ([CONJ AND], [CONJ BUT], [CONJ NEITHER] etc.). Conjuncts which are not preceded by a coordinating word (e.g. the initial conjunct in *Kim walks and Lee runs*) are marked as [CONJ NULL], as are all major category constituents which do not occur as part of a coordinate structure. The only exception here is [BAR 1] prepositional categories—since there are no rules for coordinating P1, there is no need to mark P1 with CONJ.

COORD values: +, −

The feature COORD (coordinated) occurs on all major categories of all BAR levels. The top node of a coordination of categories is marked as [COORD +] and any occurrence of a non-coordinated major category is [COORD −]. The only exception to this rule is with some S and VP rules (see ID rules ELL/*) which have a complex pattern of binding between COORD and the feature ELLIP which indicates the presence of a form of ellipsis.

SLASH values: numerous—see grammar listing

The category-valued feature SLASH encodes information about the 'missing' category in an unbounded dependency construction and this information is propagated through the tree between the point where the dependency is introduced (usually by the preposing rules S/NP_UDC1–S/AP_UDC2) and the point where it terminates (usually a gap generated by the rules TRACE1–TRACE8). By this means we ensure that the category at the top of the dependency and the category at the bottom agree with repect to relevant features.

The category which is the value of SLASH is not a category like any basic category in the grammar. Instead it is a [BAR 2] category which also has the following features: N, V, PLU, COUNT, PER, NFORM, CASE, PFORM, LOC, GERUND, ADV, PRD, QUA, SUBJ. Not all of these features are relevent to all missing categories. N and V establish the basic identity of the category (N2, P2, A2, V2) and the other features either play a role or not depending on this. If the category is an N2 then PLU, COUNT, NFORM, CASE, PRD and ADV are bound at the top and bottom of the dependency, if it is a P2 then PFORM, LOC, GERUND, PRD are bound, and if it is an A2 then PRD, QUA and ADV are bound. The value of SLASH is only instantiated to V2 in a very constrained way: it only occurs in comparatives such as *more apples than you thought*, *more apples than Kim can* and here the only relevant feature is SUBJ (is the missing material an S or a VP). A detailed description of our treatment of unbounded dependency constructions can be found in section 6.4.

BEGAP values: +, −

BEGAP occurs on all [BAR 2] categories and indicates whether that constituent can be a gap or not. Such a feature is needed because the SLASH feature by itself does not distinguish between a category which contains a gap and one which *is* a gap. For example, an N2 containing an N2 gap (as in *who does Kim have* N2[*a picture of* __]) is specified as N2[SLASH N2] and since an N2 gap (as in *what does Kim have* NP[__]) is also specified as N2[SLASH N2] there would be no way of distinguishing them without the feature BEGAP. The former is N2[SLASH N2, BEGAP −] and the latter is N2[SLASH N2, BEGAP +].

The feature BEGAP is useful both for increasing parser efficiency and for capturing linguistic constraints on the positions in which gaps can appear. For example, coordinate constructions can contain gaps so long as each conjunct contains the same kind of gap (*who does Kim have* [[*a picture of* _] *and* [*a book about* _]), but individual conjuncts cannot be gaps (**what does Kim have* [[*a picture of* _] *and* [_]).

ELLIP **values: +, −**

The feature ELLIP is used to guide the semantic translations in some VP and S rules in order to reconstruct a limited amount of ellipsis in coordinations. In examples such as *Kim will have finished her supper and gone to bed* we want to achieve a semantic translation which involves two complete propositions each seen as a separate event, i.e. we want a translation like the one which we get from the coordination of two sentences: *Kim will have finished her supper and Kim will have gone to bed.* The problem here is that the coordination is of two VPs low in the sentence and tense and aspect information and distinct event variables have to be added into the translation of the VPs higher up in the parse tree than the coordination node. We are able to achieve the correct translation by marking coordinations of non-finite VPs such as *finished her supper and gone to bed* as being elliptical ([ELLIP +]) and by marking the VP and S nodes higher up as [ELLIP +] as well. The semantic translation rules are then made sensitive to the instantiation of the feature ELLIP (see ID rules ELL/*).

WH **values: +, −, NO**

The feature WH encodes whether a constituent is a 'wh' constituent or not. Lexical items such as *who*, *what* and *where* are marked as [WH +] and this specification is inherited by the larger constituent in which such words occur. N2s, P2s and A2s may be [WH +] (*which books*, *on which table*, *how happy*) and so may Ss (*who helped Kim*). When there is no wh-word present then the constituent is [WH −]. WH does not generally occur as a feature on VPs except in the case of the infinitival interrogative VP complements of verbs such as *wonder* and *ask* as in *Kim wondered who to talk to* and *Kim wondered whether to talk to Lee* (see ID rules VP/WH1 and VP/WH2). WH also occurs on [BAR 1] categories since these are part of the path of upwards propagation of 'wh-ness'. It only occurs on [BAR 0] categories which are also pro-forms (*what*, *where*). WH always cooccurs with the features UB and EVER.

UB **values: R, Q, NO**

The feature UB distinguishes between different types of unbounded dependency: [UB Q] marks 'wh' questions, [UB R] marks relative clauses and [UB NO] either indicates topicalisation or that the construction is not unbounded. The value for UB originates with the wh-word which also determines the value for WH. For example, *what* can be a question pronoun (*what did Kim do*) and is therefore marked as [UB Q]—this specification is inherited by the S in which it occurs.

EVER **values: +, –, NO**

Some wh-words such as *whichever*, *whoever* etc. are marked as [EVER +]. This reflects not just their morphology but also the fact that they can participate in the 'free relative' construction (*whatever Kim does annoys Lee*, *Kim reads whichever books Lee reads*—see ID rules N2+/FREEREL and N2+/FREEREL2).

3.1.2 Minor Categories

DetN

DetNs (noun determiners) are characterised by the feature set [QUA +, SUBCAT DETN]. In addition they have the features DEF, POSS, DEMON, AGR, WH, UB, EVER. The feature DEF indicates definiteness and propagates from the DetN to its N2 mother (e.g. *the* is [DEF +] and *a* is [DEF –]). POSS identifies possessive determiners such as *my* and *her*—other DetNs are [POSS –]. DEMON distinguishes demonstrative determiners ([DEMON +]) from non-demonstrative ones ([DEMON –]). The feature AGR ensures that determiners agree with their nominal sisters. In addition to the usual features (PER, PLU, COUNT, NFORM, CASE), which appear on AGR N2, the AGR N2 on DetNs is also specified for QFEAT since not all determiners can occur when there is a lower quantifier (i.e. when the N2[–SPEC] is [QFEAT +])—*those few books* vs. **a one book*.

The features WH, UB and EVER also occur on DetNs. For example, *whose book* ([WH +, UB Q, EVER –]) and ([WH +, UB R, EVER –]); *whichever book* ([WH +, UB Q, EVER +]); *this book* ([WH NO, UB NO, EVER NO]).

DetA

DetAs (adjective determiners or 'degree modifiers') are characterised by the feature set [SUBCAT DETA] and have the additional features AFORM, DISTR, WH, UB and EVER. The value for AFORM on a DetA determines the AFORM value of the entire A2. For example, *more* is [AFORM ER] and the A2 *more happy* acquires this specification. DISTR also propagates from a DetA to the entire A2 and overrides an existing value for DISTR that an adjective may have. For example, the [SUBCAT NULL] entry for *crazy* is [DISTR ATT] which means it can occur in prenominal position but *too* is specified as [DISTR PRD] and so an A2 which contains *too* cannot occur prenominally (**a too crazy man*). The DetAs *how* and *however* are [WH +, UB Q] (and [EVER –] and [EVER +] respectively). All other DetAs have NO as values for these features. The only wh-A2s are ones which contain *how* or *however*.

Complementisers

Complementisers are characterised simply by a SUBCAT feature:

that = [SUBCAT THAT]
for = [SUBCAT FOR]
whether = [SUBCAT WHETHER]
if = [SUBCAT IF]
as = [SUBCAT AS]
than = [SUBCAT THAN]

Coordinators

Coordinators are characterised by the feature specification [CONJN +] and they additionally have a SUBCAT value which reflects their form, i.e. one of AND, OR, NOR, BOTH, BUT, EITHER, NEITHER. For example, *and* is [CONJN +, SUBCAT AND] and *nor* is [CONJN +, SUBCAT NOR].

Particles

Particles which appear with phrasal verbs, adjectives and nouns (*turn off*, *pick up*, etc.) are simply characterised by the feature PRT, i.e. [PRT UP] defines the particle in *pick up* and [PRT DOWN] defines the particle in *put down*.

Partitive 'of'

The *of* which appears in partitives has the specification [PFORM OF].

Cardinal Numbers

PS rules NUM1–NUM9 generate complex cardinals such as *fifty seven*, *three thousand two hundred and twenty one* etc. These rules and the lexical entries for cardinals use the following features:

```
FEATURE CN1{ZERO, TEN, TEEN, TY, HUN, THOU}
FEATURE CN2{BIG, SMALL, -}
FEATURE CN3{A, ONE}
FEATURE CN4{+, -}
FEATURE AND{+, -}
```

These features are independent of the main feature system and are sufficient to generate most complex cardinals. Cardinals only start to participate in the main grammar and feature system by virtue of ID rules A/NUMBER and A/NUMBER2.

3.1.3 Other Features

There are three features used in the grammar which have not been described in previous sections:

```
FEATURE T{S, NP}
FEATURE H{+, -}
FEATURE NULL{+, -}
```

T is used to pick out 'top' categories for the parser to prevent it from returning all possible analyses of a string. The ID rules T1–T3 and T/NP1 and T/NP2 define a variety Ss and N2s which are considered to be parses which it would be appropriate to return. In order for only these parses to be returned, the TOP construct must also be defined:

```
TOP [T S], [T NP].
```

The feature specification [H +] is used to pick out the head daughter in a rule (in order for head feature propagation to work properly). It behaves unusually in that unification ignores it. For example, in its expanded form, the head daughter in the intransitive VP rule looks like this:

(31)
```
[N -, V +, BAR 0, SUBCAT NULL, CONJ NULL, VFORM @7, H +, FIN @11,
   PAST @12, PRD @13, AUX -, INV -, PSVE -, NEG @17, PRO -,
   COORD -, SUBTYPE NONE, AGR [N +, V -, BAR 2, NFORM @20,
   PER @21, PLU @22, COUNT @23, CASE @24]]
```

An expanded entry for an intransitive verb such as *laugh* looks like this:

(32)
```
[N -, V +, BAR 0, SUBCAT NULL, CONJ NULL, VFORM NOT, FIN +,
   PAST -, PRD -, AUX -, INV -, PSVE -, NEG -, PRO -,
   COORD @44, SUBTYPE NONE, AGR [N +, V -, BAR 2, NFORM NORM,
   PER 3, PLU -, COUNT @23, CASE NOM]]
```

The two categories will unify even though the one in the rule is specified as [H +] and the one which is the verb entry is not.

The feature NULL is used by the parser to create empty categories ('gaps' or 'traces'). The daughter category in the rules TRACE1–TRACE8 is specified simply as [NULL +] and this is a signal to the parser to insert an empty category (which it indicates as 'E'). For example, the result of parsing *who does Kim help* is.

(33) `(((who) (does (kim) (help (E)))))`

3.2 Feature Propagation

The GDE provides two methods of encoding information about the way in which features propagate in trees—propagation rules and explicit statements in ID rules. These methods are used to express various feature propagation principles similar to those of GPSG, such as the Head Feature Convention (HFC), the Foot Feature Principle (FFP) and the Control Agreement Principle (CAP), and to encode more idiosyncratic propagation regimes.

3.2.1 Capturing the HFC

The HFC is designed to capture the intuition that mothers and heads share values for many features. In the ANLT formalism, we state this relationship by means of propagation rules which bind the values of head features on mother categories to the values of those features on their head daughters. Firstly a propagation rule binds values for the features N and V between mothers and heads:

(34)
```
PROPRULE PROP_NV : ; copy value of V and N from mother
                   ; to head daughter.
   [V (+, -), N (+, -)] --> [H +], U.
      F(1) = F(0), F in {V, N}.
```

Thereafter, we define a set of head features for each major category type (the sets NOMINALHEAD, VERBALHEAD, ADJHEAD and PREPHEAD) and propagation rules which bind values for these features between mother and head (propagation rules HFC_VERBAL, HFC_NOMINAL, HFC_ADJ and HFC_PREP). The result of such propagation rules is to block analyses in which mother and head have conflicting values for some head feature; for example, the tree in (35a) is licensed by the grammar, but the tree in (35b) is not, since the N2 subject is required by the S rule to be nominative but the lexical item *her* is accusative.

(35)
```
a.              S                         b.             S
         ______/ \______                          ______/ \______
    N2[CASE NOM]        VP                   N2[CASE NOM]        VP
         |              |                         |              |
 N[+PRO, CASE NOM]      V                  N[+PRO, CASE ACC]     V
         |              |                         |              |
        she            left                      her            left
```

3.2.2 Capturing the FFP

The FFP controls the propagation of a set of features (known as 'foot' features) which propagate in a less predictable way than head features do. The features in our grammar which are equivalent to GPSG's foot features are WH, UB, EVER and SLASH. We group WH, UB and EVER together but have a separate treatment of SLASH. The GPSG definition of the FFP requires that the foot features on a mother be the unification of the FOOT features on its daughters. This basically means that a mother might acquire its foot feature specifications from any of its daughters, be they heads or non-heads. In unary rules there is only one daughter which can possibly pass its features up and in some other rules there is still only one possible propagation path—for example, in VP rules which have a lexical head and just one complement, SLASH can only pass between the complement and the VP since SLASH does not occur on lexical categories. In other rules there may be more than one propagation path, for example, in a rule in which an N2 dominates a determiner and a head, wh-ness may propagate from the determiner (*which book*) or from the head (*a picture of whom*). Propagation rules (see PFOOT1–PFOOT8, PSLASH1–PSLASH10) establish one propagation path per rule, whether that it is the only possible path or whether there is another possibility as well. In the latter case we have to produce other versions of the rule where the other path is established. In the case of WH, UB and EVER, the places where we have to produce a second rule are so few that they have been done by hand, as for example with the rules in (36) (semantics suppressed).

(36) a
```
IDRULE N2+/DET1a : ; the, a, this dog. A number of different specifiers
                   ; attach under N2[+SPEC] ie Det, POSS NP, A2[+QUA]
                   ;  - in complementary distribution. This rule does
                   ; Det and only [PRD -]. [PRD +] must be indefinite
                   ;  and the semantics is different - see N2+/DET1b.
   N2[+SPEC, PRD -] -->
         DetN[AGR N2, WH NO, UB NO, EVER NO], H2[-SPEC] :
```

b
```
IDRULE N2+/DET2 : ; FOOT features propagate from determiner - which
                  ; book
   N2[+SPEC, +WH, EVER @ev, UB @ub, PRD -] -->
      DetN[AGR N2, +WH, EVER @ev, UB @ub],
      H2[-SPEC, WH NO, EVER NO, UB NO] :
```

In N2+/DET1a a propagation rule establishes a propagation path between the mother and the head such that both will have the same variable value for WH, UB and EVER – this will handle both cases where there is a wh-word inside the head (*a picture of whom*) and cases where there is no wh-word at all (*the book*). The rule N2+/DET2 explicitly binds WH, UB and EVER between the determiner and the mother (and requires the head to be [WH NO, EVER NO, UB NO]). The fact that determiner and mother must be [WH +] prevents this rule from being used to parse N2s which do not contain a wh-determiner.

We treat SLASH similarly except that the number of rules in which there is an alternative propagation path is large enough to warrant a treatment where we generate alternate versions by means of metarule (see metarules MSLASH1a–MSLASH5b) rather than by hand. For example, the rule in (37) generates VPs containing an N2 object and a finite S complement. A propagation rule binds SLASH between the mother and the N2 and a default rule defaults the S to [SLASH NOSLASH]. This means the rule can generate examples where there is no gap (*Kim tells Lee that it is true*) and examples where the gap is at/in the N2 object (*who does Kim tell E that it is true*). It cannot, however, generate examples where there is a gap in the S complement (*who did Kim tell Lee that he saw E*)—to do that we have a second rule generated by metarule, as in (38).

(37)
```
IDRULE VP/NP_SFIN1 : ; tells her (that) he can do it
   VP -->
     H[SUBCAT NP_SFIN, SUBTYPE NONE], N2[-PRD, NFORM NORM], S[+FIN] :
   2 = [SLASH NOSLASH],
      (lambda (Q) (Q (lambda (e) (lambda (x) (1 e x 2
         (3 (lambda (prop) (lambda (ta) (lambda (equa)
              (prop (uqe ((equa some) (e3) (ta e3))))))))))
                (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2], (lambda (Q) (Q (lambda (e) (lambda (x)
         (lambda (wh) (1 e x (2 wh)
            (3 (lambda (prop) (lambda (ta) (lambda (equa)
                 (prop (uqe ((equa some) (e3) (ta e3)))))))))))
                    (lambda (e2) e2) (lambda (qu) qu))).
```

(38)
```
VP/NP_SFIN1(MSLASH1b) :
   VP[-AUX, -PRO, CONJ NULL, VFORM @7, BEGAP -, FIN @11, PAST @12,
      PRD @13, NEG @17, SLASH X2[N @1, V @2, SUBJ @4, PRD @85,
      NFORM @20, PER @111, PLU @112, COUNT @113, CASE @114,
      PFORM @30, LOC @31, GERUND @32, QUA @34, ADV @37], COORD -,
      AGR N2[NFORM NORM, PER @21, PLU @22, COUNT @23, CASE @24],
      ELLIP -]] -->
   V[-INV, -AUX, -PRO, SUBCAT NP_SFIN, CONJ NULL, VFORM @7, H +,
     FIN @11, PAST @12, PRD @13, PSVE -, NEG @17, COORD @,
     SUBTYPE NONE, AGR N2[NFORM NORM, PER @21, PLU @22,
     COUNT @23, CASE @24]],
   N2[-PRD, -POSS, -ADV, +ACC, -NEG, +SPEC, CONJ NULL, BEGAP -,
      SLASH NOSLASH, NFORM NORM, PER @, PLU @, COUNT @, PN @,
      PRO @, PROTYPE @, PART -, AFORM @, DEF @, NUM @, WH NO,
      UB NO, EVER NO, COORD @, REFL @, QFEAT NO, DEMON -,
      COADV @, KIND @],
   S[-INV, +FIN, CONJ NULL, VFORM NOT, BEGAP -, PAST @, PRD @,
     AUX @, COMP @, SLASH X2[N @1, V @2, SUBJ @4, PRD @85,
     NFORM @20, PER @111, PLU @112, COUNT @113, CASE @114,
     PFORM @30, LOC @31, GERUND @32, QUA @34, ADV @37], PRO @,
     WH NO, UB NO, EVER NO, COORD @, AGR @, UDC -, ELLIP -] :
   (lambda (Q) (Q (lambda (e) (lambda (x) (lambda (wh)
      (1 e x 2
         (3 (lambda (prop) (lambda (ta) (lambda (equa)
            (prop (uqe ((equa some) (e3) (ta e3))) wh)))))))))
         (lambda (e2) e2) (lambda (qu) qu))).
```

Notice that the metarule not only binds SLASH between the mother and the S, it also instantiates its value to be at least [SLASH X2]—this ensures that the rule will only be used for cases where the S does contain a gap. It also causes the N2 object to become [SLASH NOSLASH]. More details about SLASH can be found in Section 6.4.

3.2.3 Capturing the CAP

The CAP is responsible for ensuring that categories which need to agree with other categories do, in fact, do so. In our grammar we state explicit bindings of agreement features by means of propagation rules. For example, (39) is an example of a propagation rule which forces agreement between an NP subject and a VP:

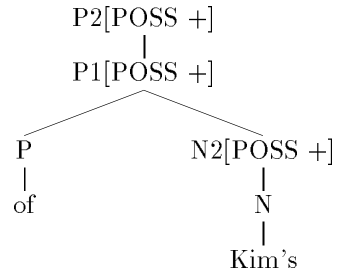
(39)
```
PROPRULE AGR/NP_VP : ; binds feature values on subject N2 to feature
                     ; values on the category that is the value of
                     ; VP[AGR]
    S --> N2, H2[-SUBJ, AGR N2], U. F(1) = F(2[AGR]), F in AGRFEATS.
```

The effect of this is to make the values of the features defined by the set AGRFEATS on the subject N2 (i.e. F(1)) be equal to the values of the same features on the N2 which is the category value of the VP (i.e. F(2[AGR])). More details about agreement and the places where agreement takes place are provided in section 6.5.

3.2.4 Other Propagations

Many other feature propagations fall outside of the scope of the GPSG feature principles. For these, a miscellaneous set of propagation rules is used. For example, some features are very like head features in that they propagate between mother and head, but are not included in the set of head features because they do not appear on all instances of a category type. For example, the feature POSS appears on P2 and P1 but not on P, as in (40).

(40)



In such cases, these features are bound by rather more specific propagation rules. (41) is the propagation rule responsible for POSS propagation between P1 and P2 in (40). (It is also responsible for propagating the feature GERUND, which also appears on P2 and P1 but not P.)

(41)
```
PROPRULE PP : ; POSS and GERUND not head features (only appear on P2
              ; and P1) since the values come from the NP not from
              ; P0. This passes them from a P2 mother to either a P1
              ; or a P2 head daughter.
    P2 --> [H +, BAR (1, 2)], U. F(0) = F(1), F in {POSS, GERUND}.
```

Other features may propagate from non-heads to mothers. For example the definiteness of an N2 is generally determined by the definiteness of the specifier, that is, DEF propagates from a specifier to its N2 mother as expressed by the following propagation rule:

(42)
```
PROPRULE PDEF3 : ; binds DEF on [+QUA] (DetN or AP) to N2 mother
    N2 --> [+QUA], U. DEF(0) = DEF(1).
```

As mentioned above, bindings between feature values can be stated explicitly in ID or PS rules. For example, the rules which introduce particles for phrasal verbs, adjectives and nouns explicitly bind PRT on the head to PRT on the particle as in (43) to ensure the grammaticality patterns in (44).

(43)
```
IDRULE VP/NP_PHRA : ; turn the light off / turn it off. Linearises
                    ; so NP preceds Prt. Idrule VP/NP_PHRB deals
                    ; with the other linearisation where NP must
                    ; be -PRO.
   VP --> H[SUBCAT NP, PRT @p], N2[-PRD], [PRT @p] :
   2 = [SLASH NOSLASH], (lambda (Q)
         (Q (lambda (e) (lambda (x) ((CP 1 3) e x 2)))
               (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2], (lambda (Q) (Q (lambda (e) (lambda (x)
                  (lambda (wh) ((CP 1 3) e x (2 wh)))))
               (lambda (e2) e2) (lambda (qu) qu))).
```

(44) a He rang up his sister

b * He rang away his sister

c * He rang over sister

3.2.5 Feature Defaults

Feature default rules state default values for features on particular categories in particular environments. As such, they prevent the need for restating regular feature value specifications on each individual rule. An explanation for each particular default rule can be found in the grammar. Default rules enrich the featural specification of categories in ID or PS rules to ensure, for example, that all N2 complements will be [CASE ACC] unless they are already marked as [CASE NOM], or that most N2 complements will be [NFORM NORM], or that all VP mothers will be marked as [AUX –] unless they are already marked as [AUX +], and so forth. (45) is the rule that achieves the AUX default:

(45)
```
DEFRULE AUX : ; lhs VP defaults to [AUX -]
   VP --> W. AUX(0) = -.
```

4 Semantics

The GDE provides facilities for defining semantic representations associated with all relevant syntactic rule types in the form of expressions of the (typed or untyped) lambda calculus as described, for example, in Dowty, Wall and Peters (1981). Formulae of the lambda calculus are constructed for each (partial) syntactic analysis which the parser yields, by taking the semantics of each local tree and building larger expressions according to the formulae associated with the rules which define the local trees. The GDE performs beta-reduction on these formulae (by default).

In the search for suitable semantic representations (SRs) for the grammar, it was decided that two distinct levels of representation would be optimal: a single unscoped SR should be assigned by the grammar and this would be handed to a scoping and post-processing module designed to yield a single, 'neutral' (usually first-order) formula capable of being input to an automated theorem prover. This allows the semantics built into the grammar to be relatively application-neutral. Specific applications could modify the scoping and post-processing module to incorporate appropriate hooks into domain-specific components. In the parser, a 'view semantics' command displays the unscoped SR and an 'interpret' command shows the output of the scoping module (see section 7 for details). This distinction between unscoped SRs and 'final' logical forms is similar to the distinction made by Alshawi *et al* (1988) between 'quasi logical forms' and 'logical forms'. Indeed, many aspects of our semantic analysis owe much to the work of Alshawi *et al* (1988) and Alshawi (1992).

In this section we give only a brief overview of the semantic representations defined by the grammar. More detail about the semantics of major categories and of constructions can be found in sections 5 and 6 respectively. The illustrative scoping module provided with the grammar is described in section 7.

4.1 The Semantics of Sentences

The semantic representations (SRs)that are defined in the grammar look as much as possible like expressions in first-order predicate logic since in almost all cases lambda abstractions are fully reduced. We employ 'event' (or more accurately what Bach (1986) terms eventuality) individual variables (indicated by 'e' in their names, but otherwise formally indistinct from other individual variables) in order to produce the kind of 'event-based' semantics described in Davidson (1967) and Parsons (1990). We also follow Barwise and Cooper (1981) in using generalized quantifiers. These are unscoped in the SRs produced by the grammar but can be scoped using the algorithms described in Hobbs and Shieber (1987) or Vestre (1991), for example. (The GDE's illustrative scoping module produces just one scoping out of all possible scopings.) (46) shows the SR associated with the the declarative sentence *those abbots helped him.*

(46)
```
"those abbots helped him"
(DECL
 (HELP (uqe (some (e1) (PAST e1)))
  (dd (the (x1) (and (pl x1) (distal x1) (ABBOT x1))))
  (pro (the (x2) (and (sg x2) (male x2))))))
```

'DECL' indicates that the sentence is declarative. The predicate involved is HELP and it takes three arguments: the first is the event argument, the second corresponds to the subject of *help* and the third to the object. Each of these arguments is labelled as being of a certain type: the event argument is labelled 'uqe' (unscoped quantifier representing an event), the subject argument is labelled 'dd' (definite description) and object argument is labelled 'pro' (pronoun). These different labels are intended to be useful to a scoper/resolution module which may follow different strategies with different types of verbal argument. Associated with the event argument is tense information (PAST) and an existential quantifier ('some') which is the default quantifier associated with events in the absence of an overt event quantifier. We do not distinguish different types of eventuality (events, states, processes etc.) since an accurate computation of such distinctions depends on the interaction of a number of factors such as the basic type of a verb, tense and aspect, and context. Because our lexicon is so large and is not hand-coded, we are unable to record the basic type of a verb and so we have no basis on which to compute the necessary distinctions. There are sentences which we know should be interpreted as states rather than events (e.g. ones where the main verb is copular *be–Kim is crazy*) but since in general we cannot make the distinction there is little point in making it just occasionally.

Apart from declaratives, the grammar recognises yes-no questions, wh-questions and imperatives. The SRs for these are very similar to their declarative equivalents:

(47)
```
"did those abbots help him"
(YNQU
 (HELP (uqe (some (e1) (PAST e1)))
  (dd (the (x1) (and (pl x1) (distal x1) (ABBOT x1))))
  (pro (the (x2) (and (sg x2) (male x2))))))
```

(48)
```
"who did those abbots help"
(WHQU
 (HELP (uqe (some (e1) (PAST e1)))
  (dd (the (x1) (and (pl x1) (distal x1) (ABBOT x1))))
  (pro (the (_wh1) (human _wh1)))))
```

(49)
```
"help him"
(IMP
 (HELP (uqe (some (e1) (NOTENSE e1)))
  (pro (the (y1) (hearer y1)))
  (pro (the (x1) (and (sg x1) (male x1))))))
```

Yes-no question SRs are identical to declaratives except for the initial 'YNQU'. Wh-question SRs have an initial 'WHQU' and also contain a '_wh' variable indicating the questioned element. Imperative SRs contain an indication that it is the hearer to whom the command is directed. Subordinate sentences are not prefixed by DECL, YNQU, WHQU or IMP.

4.2 The Semantics of Noun Phrases

Barwise and Cooper (1981) suggest that all N2s can be translated as generalized quantifiers, but a distinction is often made between N2 meanings which are anaphoric (i.e. which are cospecified with an antecedent / prior discourse referent made available by the linguistic context) and those which are true quantifiers. In a system which seeks to provide interpretations for entire discourses, it is necessary to link anaphoric N2s with their antecedents and it therefore seems necessary to distinguish a variety of different types of N2. Although neither the SRs produced by the grammar nor the scoping or post-processing module that we provide attempt to resolve anaphors, we do distinguish different types of N2 for the benefit of those who want to build their own resolution component. Some examples of SRs associated with N2s are as follows:

(50)
```
"those abbots"
(dd (the (x1) (and (pl x1) (distal x1) (ABBOT x1))))

"all abbots"
(uq (all (x1) (and (pl x1) (ABBOT x1))))

"kim"
(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))

"abbots"
(uq (some (x1) (and (pl x1) (ABBOT x1))))

(kind (all (x1) (and (pl x1) (ABBOT x1))))

"he"
(pro (the (x1) (and (sg x1) (male x1))))

"ourselves"
(rpro (the (x1) (and (pl x1) (speaker x1))))
```

Apart from the initial label, all the N2 SRs above have the same tri-partite structure—a quantifier followed by a variable that it binds followed by a term in which the bound variable occurs. When scoped, this final term becomes the restriction set of the generalized quantifier (see section 7 for details). The interpretation of the quantifiers is not part of the SR and would have to be specified by the user in the context of a particular application. The common structure is consistent with the assumption that all N2s can be treated as generalized quantifiers but the initial labels given to different kinds of N2 leave open the possibility that different strategies may be pursued depending on type. ‘dd’ marks a definite description, indicating that an antecedent may be available; ‘pro’ marks pronouns and ‘rpro’ reflexive pronouns, indicating again that antecedents may be available; ‘uq’ marks an unscoped quantifier and indicates that there is no need to search for an antecedent, but that quantifier scoping is required; ‘name’ marks names, a kind of definite description which applications may need to treat differently; and ‘kind’ marks natural kind interpretations of bare plurals. Although we mark pronouns and reflexives, we do not attempt to resolve them and do not represent in the SR syntactic structural information (e.g. c-command relations) that might have been used to constrain interpretations.

The SRs above are all associated with non-predicative N2s ([PRD –]). Predicative N2s also occur and these are given SRs quite different from the examples above. The SR associated with *Kim is an abbot* is:

(51)
```
"kim is an abbot"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (ABBOT (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))))
```

where the translation of the predicative N2 *an abbot* is the predicate SR `(lambda (x) (ABBOT x))` which is applied to the SR for *Kim*.

4.3 Events and N2s, A2s and P2s

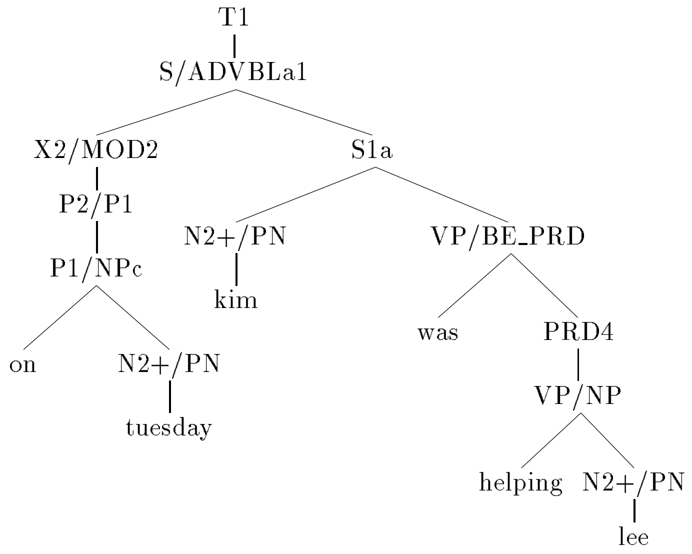
A standard event-based approach would associate event variables not only with predicates deriving from verbs but with predicates deriving from nouns, prepositions and adjectives. We have suppressed event variables other than those associated with verbs because of the large number of existential quantifiers over events which would otherwise occur and which would make our SRs somewhat unreadable. Instead, wherever N2s, A2s or P2s occur as the main predicate of a clause (usually after copula *be*) we insert the predicate BE and associate an event(uality) variable with it:

(52)
```
"kim is an abbot"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (ABBOT (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))))

"kim is crazy"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree unknown))))

"kim is in the abbey"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (IN (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (dd (the (x2) (and (sg x2) (ABBEY x2)))))))
```

4.4 Building Semantic Representations

The semantic representions associated with the complete Ss and N2s returned by the parser (with the TOP category declaration on) are quite simple and readable but the forms associated with subconstituents of these are often very complicated and hard to read, and the formulae in ID and PS rules can be very complex indeed. The reason for this division between simplicity at a high-level and complexity at a low-level is that the pieces of information needed for a complete translation of a sentence (or an N2) come from various sub-parts and these pieces of information cannot be put together to form a whole until the topmost level. If they are put together too soon then information coming from higher up in a tree cannot be incorporated. We can illustrate with a concrete example: consider the sentence *on tuesday Kim was helping Lee* which has the parse tree in (53) (annotated with rule names rather than categories) and which has the final semantic form in (54).

(53)



(54)
```
(DECL
 (and
  (HELP #1=(uqe (some (e1) (PAST (PROG e1))))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))
  (AT-TIME #1# TUESDAY)))
```

The basic predicate-argument structure originates in the lowest VP where the lexical item *helping* contributes the name of the predicate, HELP, and where the rule that builds the VP provides the information that HELP is a three-place predicate whose third argument is the object N2 and whose other arguments are to be found outside the VP. At the higher VP the combination of *be* with an [VFORM ING] complement tells us that we have progressive aspect (the PROG operator) and the fact that *was* is finite and past tense can be used at the point where the lower S is built to provide the PAST operator. It is also here that translation of the subject argument can be incorporated. The structure corresponding to the event argument cannot be put together at the lower S because there may be a quantifier higher up (as, for example, in *often Kim was helping Lee*). In this case there is no quantifer so the default `some` is assumed but only at the point where the S is deemed complete—i.e. at the point where the T1 rule picks it up. At the point of the lower S there is also still the possibility of a higher non-quantificational modifier which does indeed occur in this example—*on tuesday*. The translation of this modifier needs to be conjoined with the basic predicate-argument structure and both need the event structure as their first arguments.

At each point in the translation, provision needs to be made for incorporating information which may occur higher up in the tree and so it is inevitable that each node, apart from the top node, has a complex lambda expression associated with it which may be hard to decipher. In the following sections we try wherever possible to explain these expressions since it is necessary to understand them in order to be able to alter or add to the grammar.

5 Category Structure and Semantics

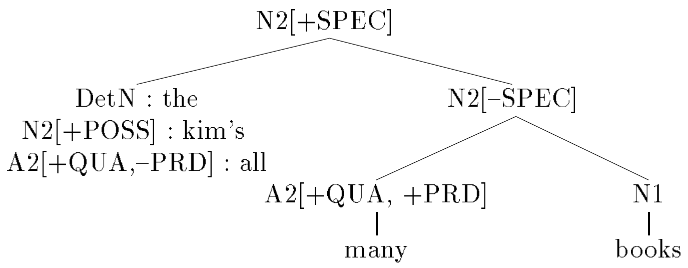
In this section we describe the basic structure of major category phrases and indicate the range of structures that are defined by the grammar. At the same time we describe the semantic translations given to the categories.

5.1 N2

5.1.1 Top Level Structure

The top levels of N2 structure can be schematised as in (55).

(55)



In (55), there are two N2 nodes distinguished by the feature SPEC. This allows us to recognise an extra level of structure inside N2 without the need to recognise an extra BAR level. In specifier position (under N2[+SPEC]), determiners, possessive N2s or [PRD –] quantifying adjectives (*all*, *both*, etc.) can appear. N2[–SPEC] may contain a [PRD +] quantifying adjective (*many*, *few*, cardinals, ordinals etc.) and N1. By identifying two types of quantifying adjective which attach at different levels, we are able to generate N2s containing more than one specifier/quantifier (*the many books*, *all three books*, *John's many virtues*) without having to build complex determiner phrases.

5.1.2 Modifiers

Modifiers are introduced by means of recursive rules as daughter to N1 and sister to N1. The set of N2 modifiers defined by the grammar is as follows:

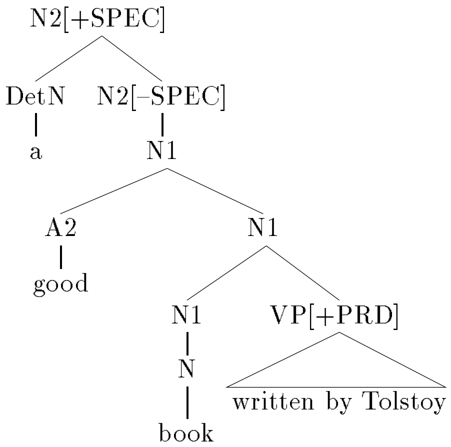
- prenominal A2 (*good book*)
- prenominal possessive N2 (*the women's javelin competition*)
- postnominal A2 (*a man taller than Lee*)
- reduced relatives (described as VP[+PRD])—*a book written by Tolstoy*, *a man writing a book*
- P2 (*a man with an umbrella*)

- possessive P2 (*a book of Lee's*)
- infinitival VP (*the man to help you*),
- 'gappy' infinitival VP (*the man to buy books from _*),
- relative clauses (*the man who sold me a book*, *the man who I bought a book from*)
- 'that'-less relatives (*the man I bought a book from*).

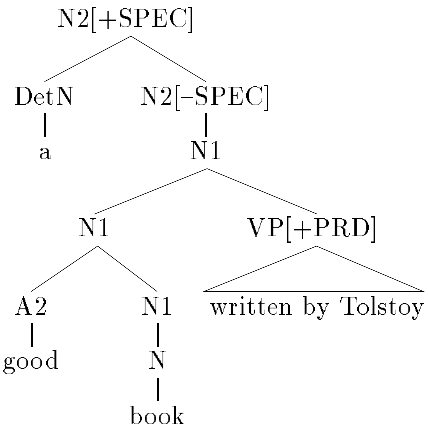
Free relatives are also catered for—see the section on relative clauses below. A problem arises when both a pre-modifier and a post-modifier occur inside an N2, in that a spurious attachment ambiguity is created. For example, *a good book written by Tolstoy* would be analysed as ambiguous between (56a) and (56b)

(56)

a.

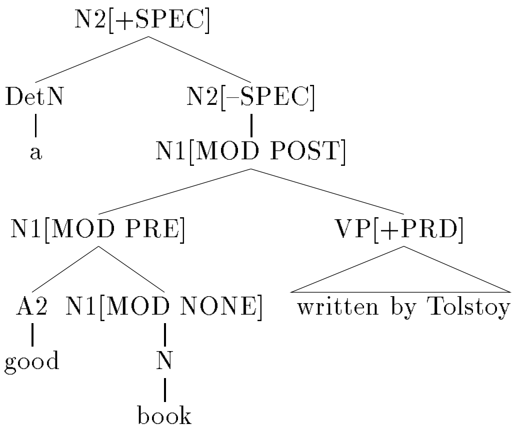


b.



To solve this problem we have introduced a feature MOD with values PRE, POST and NONE which is used in such a way as to ensure that pre-modifiers attach lower in an N1 than post-modifiers. Thus (56b) is the only tree which will be produced. (57) demonstrates the distribution of the feature MOD.

(57)



5.1.3 Complements

Complements to nouns are introduced as daughters of N1 and sisters of N (the lexical head). The lexical head carries a SUBCAT feature to ensure that nouns cooccur with appropriate complements. The full set of N complements recognised by the grammar can be found in the grammar—the rules N1/N–N1/OFN1 are ones which introduce nouns with their complements.

5.1.4 N2 Semantics

In the previous section we described how non-predicative and predicative N2s receive different translations. At the N1 level and below they have the same translation but at the N2 levels they are treated differently. Predicative N2s have a translation as a predicate of the form `(lambda (x) (...))` while at the topmost level, non-predicatives receive a translation which is a complete beta-reduced expression which cannot be altered in any way (see the examples in (49)). In order to produce such forms, information from varying sources is required. The lexical entry for the head noun provides the basic predicate and the lexical ID rule that it matches provides the predicate-argument structure for that predicate. In (58) we show the SRs for some N1s which dominate lexical heads and their complements—each of these is of the form `(lambda (x) (....))`.

(58)
```
"desire to help"
(lambda (x1)
   (DESIRE x1
           (HELP (uqe (some (e1) (NOTENSE e1)))
                 (pro (the (y1) (animate y1))))))

"doubts about the abbot"
(lambda (x1)
      (and (DOUBT x1)
           (ABOUT x1 (dd (the (x2) (and (sg x2) (ABBOT x2)))))))

"discovery that it was raining"
(lambda (x1)
   (DISCOVERY x1
              (RAIN (uqe (some (e1) (PAST (PROG e1)))))))
```

At the level of N1, the translations of modifiers are conjoined and merged with the translations of their N1 sisters:

(59)
```
"crazy abbot"
(lambda (x1)
   (and (ABBOT x1)
        (CRAZY x1 (degree unknown))))
```

```
"crazy desire to help"
(lambda (x1)
   (and (DESIRE x1
                (HELP (uqe (some (e1) (NOTENSE e1)))
                      (pro (the (y1) (animate y1)))))
        (CRAZY x1 (degree unknown))))

"abbot anxious about abbeys"
(lambda (x1)
   (and (ABBOT x1)
        ((CP ANXIOUS ABOUT) x1 (kind (all (x2) (and (pl x2) (ABBEY x2))))
            (degree unknown))))

"abbot abandoning the abbey"
(lambda (x1)
   (and (ABBOT x1)
        (ABANDON (uqe (some (e1) (NOTENSE (PROG e1))))
                 x1 (dd (the (x2) (and (sg x2) (ABBEY x2)))))))

"the abbot who lee helped"
(lambda (x1)
   (and (ABBOT x1)
        (HELP (uqe (some (e1) (PAST e1)))
              (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
               x1)))
```

It is at the higher levels that the semantics of N2s begins to become complex. At these levels, information about quantifiers, about the singular/plural/mass distinction, about definiteness etc. needs to be brought together. The semantics of N2[−SPEC] varies according to whether it contains a low attaching A2[+QUA, +PRD] or not. If it does it is [QFEAT +] and receives a translation as in (60). If it does not then it is [QFEAT −] and has a translation like an N1.

(60)
```
"many abbots"
(lambda (Q1) (lambda (T1)
    (T1 many (lambda (x1) (and (Q1 x1) (ABBOT x1))))))

"three abbots"
(lambda (Q1) (lambda (T1)
    (T1 (NN \3) (lambda (x1) (and (Q1 x1) (ABBOT x1))))))
```

Here the `(lambda (T1) (T1 ..))` is serving simply to carry up information about the quantifier separately from the rest of the translation since once the quantifier has bound its variable that variable is inaccessible. We can only use the quantifier to bind the variable associated with the noun meaning at the point where there is definitely no other expression that needs to be predicated of it and that point is higher up when the N2[+SPEC] is translated. In the rules for N2[+SPEC], the `T1` is instantiated to a function which expects two arguments (a quantifier and the lambda expression corresponding to the N1 translation) and which puts the two together so that the quantifier binds the variable. The *(lambda (Q1) (..))* is also waiting to be instantiated at the N2[+SPEC] level to a predicate expressing information about whether the N2[+SPEC] is plural, singular or mass, and possibly about possessive determiners or N2s in specifier position.

If a quantifier occurs in an N2[−SPEC] then that will be the quantifier for the whole N2 irrespective of which determiners/quantifiers attach under N2[+SPEC]. Some examples are given in (61).

(61)
```
"the many abbots"
(dd (many (x1) (and (pl x1) (ABBOT x1))))

"the three abbots"
(dd ((NN \3) (x1) (and (pl x1) (ABBOT x1))))
```

Here the lower quantifier (*many*, *three*) is the quantifier for the N2[+SPEC]. An initial *the* before the quantifier has the effect of labelling the translation as 'dd' (definite decription) which implies that an antecedent for the expression should be found.

Information about the singular/plural/mass distinction is incorporated at the N2[+SPEC] level and is a translation of the values of the syntactic features PLU and COUNT. The predicates 'pl', 'sg' and 'ms' represent the feature value-pairs [PLU +, COUNT +], [PLU −, COUNT +] and [PLU −, COUNT −] respectively. By this means we preserve syntactic information which will be needed for the interpretation of N2s. In some cases, 'pl' will indicate that the expression denotes a plural entity, 'sg' that it denotes an individual and 'ms' that it denotes a mass—(62). There is still much uncertainty about the semantics of plurals and mass terms. We assume, following Link (1983, 1987) that the domain of individuals should contain plural individuals and masses but we have not been able to map from syntactic structure to a semantic representation that accurately reflects the semantic domain. Since 'pl', 'sg' and 'ms' reflect syntactic information rather than semantic, there is not necessarily a direct correlation between these and the semantic domain. In the case of classical quantifiers such as *all* and *most*—(63)—we have examples of what Link calls "spurious plural quantification" where the N2 is syntactically plural but where the quantifier actually ranges over singular individuals in the same way as *every* does—(64). In cases such at these it is important to remember that 'pl', 'sg' and 'ms' reflect syntax and that further processing would be need to achieve an appropriate semantic characterisation.

(62)
```
"the abbot"
(dd (the (x1) (and (sg x1) (ABBOT x1))))

"the abbots"
(dd (the (x1) (and (pl x1) (ABBOT x1))))

"the butter"
(dd (the (x1) (and (ms x1) (BUTTER x1))))
```

(63)
```
"all abbots"
(uq (all (x1) (and (pl x1) (ABBOT x1))))

"most abbots"
(uq (most (x1) (and (pl x1) (ABBOT x1))))
```

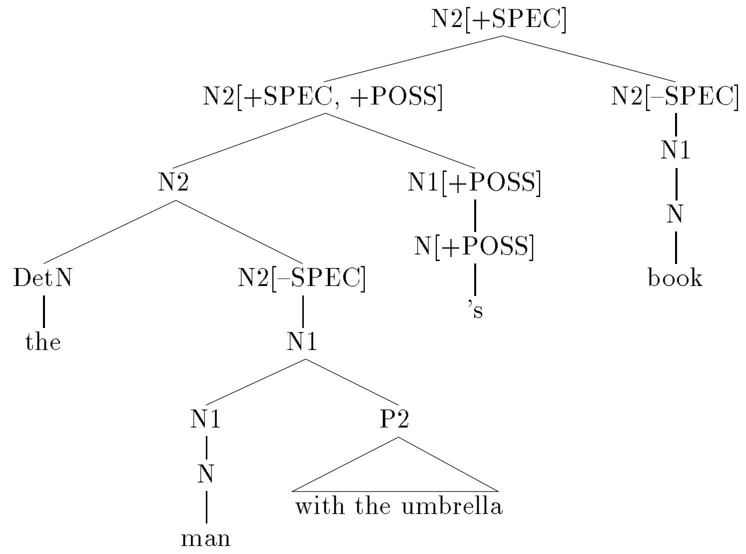
(64)
```
"every abbot"
(uq (every (x1) (and (sg x1) (ABBOT x1))))
```

There are related problems concerning collective versus distributive readings of plurals and coordinated N2s, and the question of group denoting nouns such as *committee*. We have been unable to make such distinctions in this release of the grammar.[4]

5.1.5 Possessive N2

Special rules define possessive N2s and the positions in which they can occur (see ID rules N2+/POSSNP1–N2+/POSSNP3, N2/POSSa–N2/POSSc, N1/POSSMOD1–N1/POSSMOD2 and N1/POSS). The structure assigned to an N2 such as *the man with the umbrella 's book* is shown in (65).

(65)



Possessives denote a relationship between the 'possessor' and the 'possessee' but the predicate instantiating this relationship is not recovered and is represented by a predicate variable in the SR:

(66)
```
"the abbot 's butter"
(dd (the (x1) (and (ms x1)
     (lambda (R1) (R1 x1 (dd (the (x2) (and (sg x2) (ABBOT x2))))))
     (BUTTER x1))))
```

Possessive determiners receive a similar interpretation:

(67)
```
"our abbey"
(uq (some (x1) (and (sg x1)
     (lambda (R1) (R1 x1 (pro (the (y1) (and (pl y1) (speaker y1))))))
     (ABBEY x1))))
```

[4] Some nouns, such as *committee* are marked as [GROUP +] in the lexicon but we have not provided an analysis of these in this release.

5.1.6 Partitives

An additional type of N2[+SPEC] is the partitive. Partitives are assigned one of the following two structures (see ID rules N2+/PART1–N2+/PART6B):

(68)

```
                         N2[+SPEC]
          ___________________|___________________
         |                   |                   |
N2/A2[PART OF/OF2]         P[of]             N2[+SPEC]
         |                   |                   |
    each, which              of              the book
   all, some, half                           the books
   any, both, one                            the bread
  many, few, much
   either, neither
 three, twenty one
    three pounds
```

(69)

```
                      N2[+SPEC]
            ______________|______________
           |                             |
 N2/A2[PART NO_OF]                   N2[+SPEC]
           |                             |
          all                        the book
          both                       the books
          half                       the bread
          such                        a book
```

In these structures the leftmost constituent may be an N2 (*either*, *three pounds*) or a quantifying A2 (*many*, *nearly all*). If it is an N2 then it is the head of the partitive and features such as PLU propagate from it. If the leftmost constituent is an A2 then the final N2[+SPEC] is the head and features propagate from there (although the A2 has to agree—see proprule AGR/NOM). Whether head or not, the leftmost constituent is marked with a non-negative value for PART and therefore is the constituent which triggers the partitive structure and determines whether it will contain an *of* or not.

Partitives are given translations such as the following:

(70)

```
"all of the abbots"
(uq
 (some (x1)
  (and (pl x1) (part x1 #1=(dd (the (x2) (and (pl x2) (ABBOT x2)))))
   (proportion x1 #1# ALL))))

"half the butter"
(uq
 (some (x1)
  (and (ms x1) (part x1 #1=(dd (the (x2) (and (ms x2) (BUTTER x2)))))
   (proportion x1 #1# HALF))))
```

```
"eighty nine of those abbeys"
(uq
 (some (x1)
  (and (pl x1)
   (part x1 #1=(dd (the (x2) (and (pl x2) (distal x2) (ABBEY x2)))))
   (quantity x1 #1# (NN (+ \80 \9))))))

"three grams of butter"
(uq
 (some (x1)
  (and (pl x1)
   (QUANTITY x1 (uq ((NN \3) (x2) (and (pl x2) (GRAM x2))))
    (uq (some (x3) (and (ms x3) (BUTTER x3))))))))
```

The first three examples denote a set of x1s (or a mass of x1 in the case of *half the butter*) such that the x1s form part of the set/mass denoted by the translation of the final N2[+SPEC]. The relationship between the two sets/masses is indicated by the proportion/quantity terms: in the first example, the set of x1s is equal to ALL of the set denoted by the N2, in the second the mass x1 is HALF the mass denoted by the N2 *the butter* and in the third, the set of x1s is a precise quantity of members of the set denoted by the N2 *those abbeys*: 89. Numbers are marked as NN and given arithmetic translations which are solved by the post-processing module. In the fourth example there is a QUANTITY relation between the two N2s but no 'part' relation.

5.1.7 Pronominals

There are a number of different kinds of [PRO +] nominals treated in the grammar. The feature PROTYPE distinguishes ones which can be modified from ones which cannot. All pronouns occurring on their own without modifiers have a structure where an N2[+SPEC] directly dominates the lexical head (see ID rule N2+/PRO). For the ones which take a modifier (e.g. *those who helped Kim*, *someone with a big nose*) an intermediate N1 node occurs but no N2[–SPEC] (see ID rules N2+/N1PROa–N2+/N1PROb and N1/PRO1–N1/PRO4).

Not all syntactic pro-forms translate as pronominals in the semantics. The simple unmodifiable ones, such as the personal pronouns do—(71), but the more complex ones translate as quantifiers or definite descriptions as appropriate—(72).

(71)
```
"us"
(pro (the (x1) (and (pl x1) (speaker x1))))

"ourselves"
(rpro (the (x1) (and (pl x1) (speaker x1))))
```

(72)
```
"someone"
(uq (some (x1) (and (sg x1) (person x1))))

"three"
(uq ((NN \3) (x1) (and (pl x1) (entity x1))))
```

```
"someone who helps"
(uq
 (some (x1)
  (and (and (sg x1) (person x1)) (HELP (uqe (some (e1) (PRES e1))) x1))))

"those in the abbey"
(dd
 (the (x1)
  (and (and (pl x1) (distal x1) (entity x1))
   (IN x1 (dd (the (x2) (and (sg x2) (ABBEY x2))))))))
```

Some pronominals can head partitives as described in the previous section and these have a semantic translation which can be incorporated into the translation of the partitive:

```
(73)    "neither" (of the abbots)
        (lambda (x1) (lambda (y1) (proportion x1 y1 NEITHER)))
```

5.1.8 Free Relatives

Free relatives are generated by the ID rules N2+/FREEREL and N2+/FREEREL2 and have SRs such as the following:

```
(74)    "whichever abbot helped lee"
        (dd
         (the (_wh1)
          (and (sg _wh1) (ABBOT _wh1)
           (HELP (uqe (some (e1) (PAST e1))) _wh1
            (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))))))

        "what/whatever kim helps"
        (dd
         (the (_wh1)
          (and (entity _wh1)
           (HELP (uqe (some (e1) (PRES e1)))
            (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))) _wh1))))
```

5.1.9 WH N2s

Wh nominal constituents have a straightforward syntax but their interpretation varies according to whether they are [UB Q] (wh-question constituents) or [UB R] (relative N2s). The former have a semantic interpretation which has the same form as an equivalent non-wh N2 except that the variable is a '_wh' variable (so that this can be identified as the questioned element):

(75)
```
"the abbot"
(dd (the (x1) (and (sg x1) (ABBOT x1))))

"which abbot"
(dd (the (_wh1) (and (sg _wh1) (ABBOT _wh1))))
```

N2s which are [UB R] are lambda abstracts since they have to be combined with a relative clause and an N1 interpretation:

(76)
```
(the book) "the cover of which" (is torn)
(lambda (z1) (dd (the (x1)
      (and (sg x1) (and (COVER x1) (OF x1 z1))))))

(the man) "whose book" (I read)
(lambda (z1) (dd (the (x1)
      (and (and (sg x1) (BOOK x1))
         (lambda (R1) (R1 x1 z1))))))
```

5.1.10 Non-restrictive Modification

There is no provision in the grammar for non-restrictive nominal modifiers except for one ID rule, N2+/APPOS, which permits an N2[+SPEC] to occur in apposition to a name, or a name to occur in apposition to an N2, as in (77).

(77)
```
"kim the abbot"
(dd
 (the (x1)
  (and (sg x1)
   (equal x1 (name (the (x2) (and (sg x2) (named x2 KIM) (animate x2)))))
   (equal x1 (dd (the (x3) (and (sg x3) (ABBOT x3))))))))

"the abbot kim"
(dd
 (the (x1)
  (and (sg x1)
   (equal x1 (name (the (x2) (and
```

5.1.11 Deixis

Deictic pronouns and N2s containing deictic determiners have a standard syntax but have translations as definite descriptions which encode whether they require their antecedents to be distal or proximal:

(78)
```
"this"
(dd (the (x1) (and (or (sg x1) (ms x1)) (proximal x1) (entity x1))))

"those abbeys"
(dd (the (x1) (and (pl x1) (distal x1) (ABBEY x1))))
```

5.1.12 Bare Plurals

Bare plurals receive different interpretations in different contexts. Although they are syntactically unambiguous, they tend to receive more than one semantic interpretation. When they are non-predicative (for example, subjects or objects) they are ambiguous between an existential quantifier reading and a 'natural kind' reading as in (79). The information needed to disambiguate these examples is not available in the syntax and so the ambiguity is not resolved.

(79)
```
"abbots seldom abdicate"

(DECL
 (ABDICATE (uqe (seldom (e1) (PRES e1)))
  (uq (some (x1) (and (pl x1) (ABBOT x1))))))

(DECL
 (ABDICATE (uqe (seldom (e1) (PRES e1)))
  (kind (all (x1) (and (pl x1) (ABBOT x1))))))
```

Predicative bare plurals (which usually occur after *be*) have only one interpretation, as predicates:[5]

(80)
```
"they are abbots"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (ABBOT (pro (the (x1) (and (pl x1) (entity x1)))))))
```

5.1.13 Names

Proper names are treated like ordinary nouns, except they are marked as [PN +]. In the usual case where they occur as complete N2s, they are generated by the rule N2+/PN which builds a N2[+SPEC] from a [PN +] lexical head. They receive a semantics as a type of expression with the label 'name'. Otherwise they have the structure of a definite description with a 'the' quantifier which predicates a naming relation between the entity and the name as in (81). The PS rules N/NAME1–N/NAME2B build complex names out of simple names and titles:

(81)
```
"kim"
(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))

"kim jones jnr"
(and
 (and (name (the (x1) (and (sg x1) (named x1 JONES) (animate x1))))
  (first-name (name (the (x2) (and (sg x2) (named x2 KIM) (animate x2))))))
 (title JNR))
```

[5]Non-predicative N2s can also occur after *be* where *be* is interpretated as a predicate asserting equality between two entities—in this case the existential quantifier vs. natural kind ambiguity surfaces. This means that an example such as *they are abbots* actually receives two parses and three interpretations.

Proper names can also occur with determiners and modifiers and in this case we use the ID rule N1/PN to allow names to behave as if they were ordinary common nouns. The semantic representations they receive are illustrated in (82).

(82)
```
"this kim"
(dd
 (the (x1)
  (and (sg x1) (proximal x1)
   (equal x1
          (name (the (x2) (and (sg x2) (named x2 KIM) (animate x2))))))))

"a mr jones from paris"
(uq
 (some (x1)
  (and (sg x1)
   (and
    (equal x1
     (and (name (the (x2) (and (sg x2) (named x2 JONES) (animate x2))))
      (title MR)))
    (FROM x1
          (name (the (x3) (and (sg x3) (named x3 PARIS) (place x3)))))))))
```

5.1.14 Headless N2s

Some N2s can contain an adjective but no nominal head (e.g. *the happy*, *the happiest*, *the happier*, *the fourth*). These are generated by rules where an N2[+SPEC] immediately dominates a determiner and an A2—see ID rules N2+/ADJ1–N2+/ADJ4. We choose this treatment rather than using a lexical rule to convert adjectives to nouns because the adjectives can be modified in the same way as other adjectives (*the extremely poor*). Another possibility would have been to generate empty nominal heads but there seemed no need to do this. In the semantics, the 'missing' lexical head is reconstructed and the adjective is predicated of it:

(83)
```
"the crazy"
(dd (the (x1) (and (pl x1) (and (human x1) (CRAZY x1 (degree unknown))))))

"the fourth"
(dd (the (x1) (and (sg x1) (and (entity x1) (FOURTH x1)))))
```

5.1.15 Compounds

The system makes a distinction between morphological compounding and syntactic compounding where compounds which have an orthography with no blank spaces (*tablecloth*, *paper-work*) are recognised by the word grammar while compounds with blank spaces (*shoe lace, paper clip*) must be recognised by the sentence grammar. Accordingly, the grammar contains some ID rules (N/COMPOUND1–N/COMPOUND3) for noun-noun compounds (*shoe lace*), adjective-noun compounds (*cold drink*) and preposition-noun compounds (*after care*), respectively. Our treatment of compounding does not cater for all possibilities and leads to a great deal of ambiguity in parses. There is no means to decide on the correct structure for several-word compounds (such as *warm air flow regulator*). Semantically, compounds are not analysed but are given structures paralleling the syntactic structures:

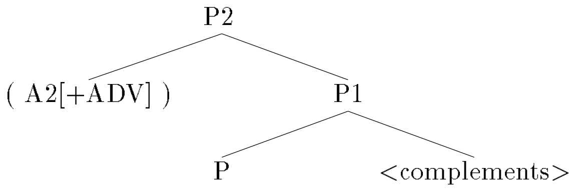
(84)
```
"air flow regulator"

(dd (the (x1) (and (sg x1)
   ((compound (compound AIR FLOW) REGULATOR) x1))))

(dd (the (x1) (and (sg x1) ((compound AIR (compound FLOW REGULATOR)) x1))))
```

5.2 P2

5.2.1 The Structure of P2

The overall structure of P2 is sketched in (85).

(85)



Under P2, the optional A2[+ADV] node allows for adverbial modifiers, as in *almost under the table*—see ID rule P2/ADVMOD. The set of complements of P recognised by the grammar is illustrated in (86).

(86)	in the cupboard	ID rules P1/NPa–P1/NPb
	on tuesday	ID rule P1/NPc
	because of the weather	ID rule P1/PPa
	down in the cellar	ID rule P1/PPb
	of Kim 's	ID rule P1/POSS
	because he ate too much	ID rule P1/SFIN
	after visiting Lee	ID rule P1/VPING
	despite Kim having left	ID rule P1/SING

Pro-P2s such as *when*, *where* and *why* are generated by the rule P2/PRO1 which makes the lexical head an immediate daughter of P2. Other pro-P2s can be modified by relative clauses and P2s (*here in the garden*, *then when he finished*) and so we treat these as nominal (N2[+PRO, +PRD]) in order for them to be modified but then the ID rule P2/PRO2 picks them up so that they can function as pro-P2s.

5.2.2 The Semantics of P2

Like N2s, P2s can be either predicative ([PRD +]) or non-predicative ([PRD –]). Adjunct P2s are always predicative and translate as two-place relations whose first argument is either an event (in the case of VP adjuncts) or an entity (in the case of N2 adjuncts). The second argument is always the translation of the complement of the P2:

(87)
```
"kim abdicated in the abbey"
(DECL
 (and
  (ABDICATE #1=(uqe (some (e1) (PAST e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
  (IN #1# (dd (the (x2) (and (sg x2) (ABBEY x2)))))))

"the abbot with kim abdicated"
(DECL
 (ABDICATE (uqe (some (e1) (PAST e1)))
  (dd (the (x1) (and (sg x1)
   (and (ABBOT x1)
    (WITH x1
     (name (the (x2) (and (sg x2) (named x2 KIM) (animate x2))))))))))

"kim helped because lee helped"
(DECL
 (and
   (HELP #1=(uqe (some (e1) (PAST e1)))
        (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
   (and
      (HELP #2=(uqe (some (e2) (PAST e2)))
            (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))
      (BECAUSE #1# #2#))))
```

Argument P2s are sometimes predicative (in which case they are relations just like adjunct P2s) and sometimes non-predicative. With the non-predicatives, the N2 which is the complement of the preposition is seen directly as an argument of the higher predicate. In some cases, i.e. cases of true 'case-marking' prepositions, the preposition makes no real contribution to the semantic translation and information about it is lost (for example *to* in *Kim gives some butter to the abbot*—(88)). In other cases, the preposition is incorporated into the translation of the higher predicate to make a new complex predicate (e.g. *at* in *the abbot is looking at Kim*—(89)). Complex predicates are bracketed with an initial `CP` which is used to identify them to the post-processing module. This changes `(CP LOOK AT)` into the predicate `LOOK-AT` (see section 7).

(88)
```
"kim gives some butter to the abbot"
(DECL
 (GIVE (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq (some (x2) (and (ms x2) (BUTTER x2))))
  (dd (the (x3) (and (sg x3) (ABBOT x3))))))
```

```
(89)    "the abbot is looking at kim"
        (DECL
         ((CP LOOK AT) (uqe (some (e1) (PRES (PROG e1))))
          (dd (the (x1) (and (sg x1) (ABBOT x1))))
          (name (the (x2) (and (sg x2) (named x2 KIM) (animate x2))))))
```

The mechanism by which we make the semantics of an N2 complement of a non-predicative P2 available to be integrated higher up is to assign a semantics to the P2 which is a function over the preposition and the N2 meaning:

```
(90)    "to the abbot"
        (lambda (Q1) (Q1 TO
            (dd (the (x1) (and (sg x1) (ABBOT x1))))))


        "at kim"
        (lambda (Q1) (Q1 AT
            (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))))
```

This makes the two pieces of information available separately. In the examples in (88) and (89), the N2 meaning is directly integrated as an argument of the verb which subcategorises the P2. In (88) the information about the preposition is not used at all but in (89) it is used to create a complex predicate.

Pro-P2s receive an interpretation where an appropriate prepositional meaning and second argument has been supplied:

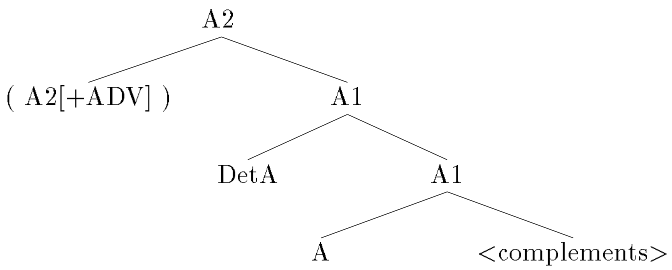
```
(91)    "there"
        (lambda (y1) (AT y1
          (dd (the (x1) (and (sg x1) (distal x1) (place x1))))))

        "here where he abdicated"
        (lambda (y1) (AT y1
          (dd (the (x1) (and (and (sg x1) (proximal x1) (place x1))
                             (and (ABDICATE #1=(uqe (some (e1) (PAST e1)))
                                  (pro (the (x2) (and (sg x2) (male x2)))))
                                  (AT #1# x1)))))))
```

5.3 A2

5.3.1 The Structure of A2

A2 structure is outlined in (92).

(92)



The adverbial attaching at the top level allows for the examples in (93).

(93) a extremely stupid
b nearly all
c incredibly quickly

DetAs attach at the A1 level to permit the examples in (94). The recursive nature of the rule means that more than one DetA can occur and this seems to be correct in some cases (*too too stupid*, *so very happy*) but not so correct in other cases (**too so stupid*).

(94) a so stupid
b very many
c too quickly

The grammar provides for a number of different complements of A—see ID rules A1/A—A1/PP_WHS. The types of adjective recognized include ones taking a variety of P2 and S complements, control adjectives (*eager to see the film*, *likely to see the film*), tough-movement adjectives (*the book is easy to read*), adjectives which take V2 subjects (*that you haven't read the book is obvious*, *for you to read the book would be easy, to stay at home would be possible*) and adjectives that take pleonastic *it* as subject (*it would be possible (for us) to stay at home*). For more details see the sections on control and agreement, unbounded dependencies, extraposition and comparatives in section 6.

5.3.2 The Semantics of A2s

A2s translate as predicates where the number and type of arguments depends on the sub-categorisation type. Some examples:

(95)
```
"lee is crazy"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))
   (degree unknown))))

"lee is eager to help"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EAGER #1=(name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))
   (HELP (uqe (some (e2) (NOTENSE e2))) #1#) (degree unknown))))

"lee is certain that the abbot abdicated"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (CERTAIN (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))
   (ABDICATE (uqe (some (e2) (PAST e2)))
    (dd (the (x2) (and (sg x2) (ABBOT x2)))))
   (degree unknown))))

"lee is easy to help"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EASY
   (HELP (uqe (some (e2) (NOTENSE e2))) (pro (the (z1) (entity z1)))
    (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1)))))
   (degree unknown))))
```

All gradable adjectives are treated as having a final argument which represents the 'degree' associated with them. In the case of the examples above which do not contain degree modifiers, this argument is simply `(degree unknown)`. In examples which do contain degree modifiers (*so*, *too*, *more*, *as* etc.) the degree argument is more informative:

(96)
```
"lee is so crazy"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))
   (degree so))))
```

(97)
```
"how crazy is lee"
(WHQU
 (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))
   (degree _wh))))
```

```
(98)   "kim is more crazy than lee is"
       (DECL
        (BE (uqe (some (e1) (PRES e1)))
         (CRAZY (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
          (degree
           ((more than)
            (lambda (d1)
             (BE (uqe (some (e2) (PRES e2)))
              (CRAZY (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
               (degree d1)))))))))
```

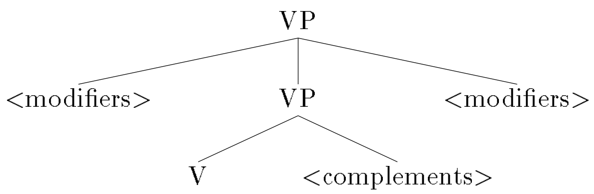
As illustrated in (98), in adjectival comparatives, the comparison occurs inside the degree argument. See section 6.8 for more details.

5.4 VP

5.4.1 The Structure of VP

The general structure of VP is:

(99)



5.4.2 Complements

A large variety of verb complements are included in the grammar. The set of VP rules includes ones for intransitives, transitives, ditransitives, verbs taking P2s, verbs taking N2s and P2s, verbs taking A2s, verbs taking S complements of various types, auxiliary verbs, control verbs (subject raising, subject equi, object raising, object equi), verbs taking embedded question complements, copula *be*, verbs taking sentential or pleonastic *it* subjects etc. For details see the grammar and the sections below.

5.4.3 Modifiers

VP final modifiers may be any of A2[+ADV], P2 or N2[+ADV] or a coordination of these (see ID rule VP/MOD1) but VP initial modifiers may only be A2[+ADV] (see ID rule VP/MOD2):

(100) a Kim played tennis enthusiastically
b Kim played tennis in the garden
c Kim played tennis this week

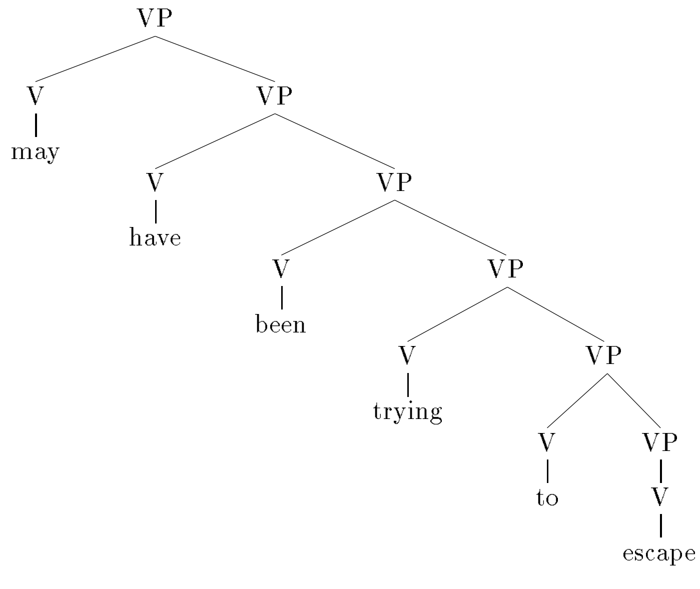
(101) a Kim was enthusiastically playing tennis
b *Kim has in the garden played tennis
c ?Kim was this week playing tennis

Because the structure assigned to auxiliary sequences is a deep right-branching structure, care has been taken to restrict the positions in which modifiers can attach. VP final modifiers can only attach to [AUX –] VPs thus ensuring that in an example like *Lee may have been reading the book without understanding it*, the modifier *without understanding it* can only attach to the VP containing the main verb (*reading the book*). VP initial adverbs can only attach to non-finite VPs since adverbs which attach between a subject and a finite VP are dealt with by the S rules.

5.4.4 Auxiliaries

Sequences of auxiliaries receive a recursive structure as in (102) where the auxiliary is treated as the head of its VP and where its complement is another VP. The required VFORM value of the VP complement is encoded in the auxiliary rules and this ensures that only grammatical sequences of auxiliaries are recognised.

(102)



5.4.5 The Semantics of VPs

VPs all receive a SR which contains three pieces of information: predicate argument structure, tense and aspect information and quantifier information. These pieces of information are kept separate in order that they may individually be added to or modified higher in the tree. The means of keeping them separated is to make them arguments of a lambda-abstracted predicate `Q`: each VP has an SR of the form `(lambda (Q) (Q` <pred-arg structure> <tense-aspect info> <quantifier info>`))`. The predicate argument information is of the form `(lambda (e) (lambda (x) (..)))` where `e` is the event variable associated with the verb and `x` is a variable which will be instantiated to the entity denoted by the subject N2. There is a very large range of verb subcategorisation types dealt with by the grammar and each of the rules receives a fairly straightforward predicate argument structure. The most complicated rules from the semantic point of view are the auxiliary and modal verb rules since these are the ones that provide tense and aspectual information. We treat tense information by means of the four operators PRES, PAST, FUT and NOTENSE over the event variable associated with the formula.[6] Aspectual information is encoded by the operators PERF and PROG. In the following examples, we show first the translation of the S and then the translation of the topmost VP:

(103)
```
"kim has helped the abbot"
(DECL
 (HELP (uqe (some (e1) (PRES (PERF e1))))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (dd (the (x2) (and (sg x2) (ABBOT x2))))))

"has helped the abbot"
(lambda (Q1) (Q1
  (lambda (e1) (lambda (x1)
     (HELP e1 x1 (dd (the (x2) (and (sg x2) (ABBOT x2)))))))
  (lambda (e2) (PERF e2))
  (lambda (qu1) qu1)))
```

(104)
```
"kim was often helping the abbot"
(DECL
 (HELP (uqe (often (e1) (PAST (PROG e1))))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (dd (the (x2) (and (sg x2) (ABBOT x2))))))

"was often helping the abbot"
(lambda (Q1) (Q1
  (lambda (e1) (lambda (x1)
     (HELP e1 x1 (dd (the (x2) (and (sg x2) (ABBOT x2)))))))
  (lambda (e2) (PROG e2))
  (lambda (qu1) often)))
```

[6] NOTENSE indicates that the tense cannot be ascertained.

(105)
```
"kim has been helping the abbot"
(DECL
 (HELP (uqe (some (e1) (PRES (PERF (PROG e1)))))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (dd (the (x2) (and (sg x2) (ABBOT x2))))))

"has been helping the abbot"
(lambda (Q1) (Q1
  (lambda (e1) (lambda (x1)
     (HELP e1 x1 (dd (the (x2) (and (sg x2) (ABBOT x2)))))))
  (lambda (e2) (PERF (PROG e2)))
  (lambda (qu1) qu1)))
```

Most modal verbs are interpreted as ambiguous between epistemic and deontic readings. The epistemic readings are described by means of the two propositional operators NEC and POS:

(106)
```
"kim must be in the abbey"
(DECL
 (NEC
  (BE (uqe (some (e1) (PRES e1)))
   (IN (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
    (dd (the (x2) (and (sg x2) (ABBEY x2))))))))

"must be in the abbey"
(lambda (Q1) (Q1
  (lambda (e1) (lambda (x1)
    (NEC (BE e1 (IN x1 (dd (the (x2) (and (sg x2) (ABBEY x2)))))))))
  (lambda (e2) e2)
  (lambda (qu1) qu1)))
```

Deontic readings are treated just like the SRs of subject equi verbs; the subject of the modal verb is the same as the subject of the verb of the embedded proposition argument of the modal:

(107)
```
"kim must be in the abbey"
(DECL
 (MUST (uqe (some (e1) (PRES e1)))
  #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (BE (uqe (some (e2) (NOTENSE e2)))
   (IN #1# (dd (the (x2) (and (sg x2) (ABBEY x2))))))))

"must be in the abbey"
(lambda (Q1) (Q1
  (lambda (e1) (lambda (x1)
    (MUST e1 x1
     (BE (uqe (some (e2) (NOTENSE e2)))
      (IN x1 (dd (the (x2) (and (sg x2) (ABBEY x2)))))))))
  (lambda (e3) e3)
  (lambda (qu1) qu1)))
```

VP modifiers are treated as predicates which are applied to the event variable associated with the verb and which are conjoined with the main predicate:

(108)
```
"lee abdicated in paris"
(DECL
 (and
  (ABDICATE #1=(uqe (some (e1) (PAST e1)))
   (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1)))))
  (IN #1# (name (the (x2) (and (sg x2) (named x2 PARIS) (place x2)))))))
```

5.4.6 Phrasal Verbs

Phrasal verbs occur with a number of different subcategorisation possibilities, for example, there are intransitive phrasal verbs (*fall over*), transitive ones (*blow it up*), ditransitive ones (*give him back the book*), ones taking sentential complements (*point out that he is wrong*) etc. For this reason phrasal verbs are treated in exactly the same way as their non-phrasal counterparts, except that they have an additional feature PRT to encode the form of the particle that they require. ID rules with 'PHR' in their names are rules specifically for phrasal verbs and look much like the equivalent non-phrasal rules except for the feature PRT both on the head and the particle. The presence of particles in so many rules leads to a certain degree of complication in the LP rules.

In the semantics, phrasal verbs are treated just like their non-phrasal counterparts except that the predicate is a complex one made up of the verb plus the particle:

(109)
```
"the abbot falls over"
(DECL
 ((CP FALL OVER) (uqe (some (e1) (PRES e1)))
  (dd (the (x1) (and (sg x1) (ABBOT x1))))))
```

5.4.7 Pro VPs

Auxiliary verbs are able to occur without their complements under certain conditions. The term 'VP ellipsis' is usually used to describe this phenomenon. An auxiliary's complement can only be missing if there is some other VP in the sentence or discourse which serves as an antecedent for that complement and constrains its interpretation. The examples in (110) illustrate VP ellipsis.

(110) a Kim doesn't like onions and Lee doesn't either.

b Kim doesn't want to see the film but Lee might.

c You shouldn't have!

d Lee can't go to the meeting but somebody ought to.

Syntactically we treat the auxiliaries which can occur without complements as pro-forms (i.e. [PRO +]). These head VPs which are also [PRO +]—see ID rules VP/PRO and

VP/PRO2 and PS rules V/PRO/SO1–V/PRO/SO3. Since we do not try to relate pro VPs to their antecedents, we do not attempt to restrict their distribution in any way.

Semantically, we represent the propositions associated with pro-VPs as involving the predicate 'PROVP' which takes an event argument and a subject argument:

(111)
```
"lee will have"
(DECL
 (PROVP (uqe (some (e1) (FUT (PERF e1))))
  (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))))

"kim is eager to"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EAGER #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (PROVP (uqe (some (e2) (NOTENSE e2))) #1#) (degree unknown))))

"he isn't"
(DECL
 (NOT
  (PROVP (uqe (some (e1) (PRES e1)))
  (pro (the (x1) (and (sg x1) (male x1)))))))
```

The presence of PROVP in the SR is designed to signal to a resolution component that this is a case of VP ellipsis and an antecedent is required. In examples where the pro-VP is supported by *either* or *neither*, as in *Kim didn't either*, *neither did Lee*, the predicate is 'NPROVP' rather than 'PROVP' to indicate that a negative antecedent is required:

(112)
```
"kim didn't either", "neither did kim"
(DECL
 (NOT
  (NPROVP (uqe (some (e1) (PAST e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))))
```

With modal pro-VPs, the epistemic/deontic ambiguity occurs:

(113)
```
"kim might"

(DECL
 (POS
  (PROVP (uqe (some (e1) (PRES e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))))

(DECL
 (MIGHT (uqe (some (e1) (PRES e1)))
  #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (PROVP (uqe (some (e2) (NOTENSE e2))) #1#)))
```

5.5 S

5.5.1 The Structure of S

ID rules S1a and S2–S/V2_SUBJ5 expand S as a subject and a VP. The following possibilities are recognised:

(114)	Kim reads the book	(nominative N2 subject, finite VP)	S1a
	fido be a good dog	(nominative N2 subject, BSE VP)	S2
	him to read the book	(accusative subject, infinitive VP)	S3
	him reading the book	(accusative subject, ING VP)	S4
	that Kim reads bothers Lee	(finite S subject, finite VP)	S/V2_SUBJ1
	for us to go is possible	(non-finite S subject, finite VP)	S/V2_SUBJ2
	to go is possible	(non-finite VP subject, finite VP)	S/V2_SUBJ3
	that he leave is necessary	(BSE S subject, finite VP)	S/V2_SUBJ4
	whether/what he won't clear	(question S subject, finite VP)	S/V2_SUBJ5

ID rules S/THAT1–S/AS expand S as a complementiser followed by an appropriate type of S. The following possibilities are recognised:

(115)	that Kim reads	(*that* + finite S)	S/THAT1
	that fido be good	(*that* + BSE S)	S/THAT2
	for him to read	(*for* + non-finite S)	S/FOR
	whether he reads	(*whether* + finite S)	S/Q1
	if he reads	(*if* + finite S)	S/Q2
	than Kim reads	(*than* + finite S)	S/THAN
	as Kim does	(*as* + finite S)	S/AS

ID rules S/NP_UDC1–S/AP_UDC2 expand S as a preposed constituent (N2, P2 or A2) followed by a slashed S whose value for SLASH is of the same category as the preposed constituent (see section 6.4 for details of unbounded dependencies).

ID rule S/IMPER expands S just as VP[FIN +, VFORM BSE] (i.e. an imperative VP).

ID rules S/TAGQUESTION and S/TAG define complete tag questions and the tag part of tag questions, respectively.

Inverted sentences are produced by means of the SAI metarule (see section 6.2).

5.5.2 Modifiers of S

ID rules S/ADVBLa1 and S/ADVBLa2 permit A2[+ADV], N2[+ADV], P2 or a coordination of any of these to precede an S to generate examples such as:

(116) a enthusiastically Kim played tennis

b in the garden Kim played tennis

c this week Kim played tennis

Adverbials which are sentence final always attach to VP rather than S.

ID rules S1b and S1c respectively allow an A2[+ADV, −QUA] and an A2[+ADV, +QUA] to attach between an NP subject and the VP:

(117) a Kim enthusiastically played tennis

b Kim often played tennis

5.5.3 Semantics of S

In section 4 and elsewhere, we have given examples of the SRs which we assign to different kinds of main clause S—declaratives, yes-no questions, wh-questions and imperatives. In fact these representations are not the translations which are associated with the S nodes themselves but are the translations of the node which dominates them, i.e. the top category nodes in ID rules T1–T3. The translation of an S node is, in fact, an incomplete expression much like the translation of a VP and it falls to the rules which introduce S to take this expression and complete it. The translation of an S node, then, is the same as VP except that the subject is included: i.e. it is an expression of the form `(lambda (Q) (Q` <pred-arg structure> <tense-aspect info> <quantifier info>`))` where the predicate argument information is of the form `(lambda (e) (..))` and where `e` is the event variable.

Rules which introduce S have to provide a three-place function for the `(lambda (Q) (Q ..))` translation of S to apply to where this function will specifiy how to put the three pieces of information together. To give a concrete example, the SR for the S node in the sentence *Kim has often helped* is this:

(118)
```
"kim has often helped" (S node)
(lambda (Q1) (Q1
    (lambda (e1) (HELP e1 (pro (the (x1) (and (sg x1) (male x1))))))
    (lambda (e2) (PRES (PERF e2)))
    (lambda (qu1) often)))
```

Here the first argument of Q1 is the predicate argument structure, the second argument is the tense and aspect information and the third argument is information about the quantifier. The top category rule T1 is specified as follows:

```
(119)   T1 : ; root symbol for the parser. It is here that the information
             ; that root sentences are always finite is encoded. This means
             ; that the parser gives two parses for finite sentences (one as
             ; T, one as S) whilst only one parse for non-finite
             ; sentences (as S)
           [T S] --> S[H +, COMP NORM, +FIN] :
           1 = [VFORM NOT, INV -], (DECL (1 (lambda (prop)
                       (lambda (ta) (lambda (equa)
                             (prop (uqe ((equa some) (e) (ta e)))))))) :
           1 = [VFORM NOT, INV +],
              (YNQU (1 (lambda (prop) (lambda (ta) (lambda (equa)
                             (prop (uqe ((equa some) (e) (ta e)))))))) :
           1 = [VFORM BSE],
              (IMP (1 (lambda (prop) (lambda (ta) (lambda (equa)
                            (prop (uqe ((equa some) (e) (ta e)))))))).
```

The first semantic condition (`1 = [VFORM NOT, INV --]`) is the one that is relevant to our example. The translation specifies that 1, the translation of S, should be applied to the function
`(lambda (prop) (lambda (ta) (lambda (equa)`
`(prop (uqe ((equa some) (e) (ta e))))))).`
This in turn is a three-place function looking for a predicate-argument structure (`(lambda (prop)..`), tense and aspect information (`(lambda (ta) ..`) and quantifier information (`(lambda (equa) ..`)—the arguments to Q1. The result of applying this function to those arguments will be an expression formed by applying the predicate-argument expression to a quantifier structure built using the tense and aspect information and the quantifier information. The final beta-reduced translation of the T node is given in (120).

```
(120)   "he has often helped" (T node)
        (DECL
         (HELP (uqe (often (e1) (PRES (PERF e1))))
          (pro (the (x1) (and (sg x1) (male x1))))))
```

Rules which introduce subordinate S categories also have to specify the completion of the S. For example, the subordinate S in *Kim believes he has often helped* has the same translation as it would have in a main clause—(118). The rule for generating VPs with a verb with a finite sentential complement specifies exactly the same procedure for putting the three parts of the S translation together:

```
(121)   VP/SFIN1 : ; believes (that) he can do it
           VP --> H[SUBCAT SFIN, SUBTYPE NONE], S[+FIN] :
           2 = [SLASH NOSLASH],
           (lambda (Q) (Q
               (lambda (e) (lambda (x) (1 e x (2 (lambda (prop)
                  (lambda (ta) (lambda (equa)
                    (prop (uqe ((equa some) (e3)
                     (ta e3)))))))))))
               (lambda (e2) e2) (lambda (qu) qu))) :
                  .
```

The function `(lambda (prop) (lambda (ta) (lambda (equa) ...` is the same as before except this time the S translation is embedded inside the translation of the higher VP and no `DECL` label is introduced. The SR associated with the top category of the whole example is given in (122).[7]

(122)
```
"kim believes he has often helped"
(DECL
 (BELIEVE (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (HELP (uqe (often (e2) (PRES (PERF e2))))
   (pro (the (x2) (and (sg x2) (male x2)))))))
```

A very similar function is needed for VP rules which introduce VP complements (i.e. rules for control verbs)—the three pieces of information in the translation of the VP must be put together to form a proposition. The only real difference is that a VP complement must first be supplied with a subject. For more details about control constructions see section 6.5.

5.6 X2

As explained in section 3, the grammar utilises phrasal categories which have the specification [BAR 2] but which have no specifications for the features N and V. There are two places where X2 categories are used: to describe the predicative complements of *be* (X2[+PRD]) and to describe adverbials (X2[+ADV]). In each of these cases, more than one type of major category phrase can occur and cross-categorial coordinations are possible:

(123) a Kim is happy/an idiot/in Paris/laughing/loved by all (A2/N2/P2/VP/VP)
b Kim is an idiot but loved by all
c Kim is happy and laughing

(124) a Kim ate her supper in the kitchen/later than Lee/an hour ago (P2/A2/N2)
b Kim ate her supper in the kitchen and later than Lee did
c Kim ate her supper an hour ago or after Lee left.

In order to generate the cross-categorial coordinations we need rules which recognise A2s, N2s, P2s etc. as being similar in some way and we do this with unary rules which rewrite X2[+PRD] or X2[+ADV] as more specific [BAR 2] categories (ID rules PRD1–PRD4 and X2/MOD1–X2/MOD4). Once these rules exist for the coordination cases it makes sense to use them in the non-coordination cases as well since this cuts down on rule numbers. The ID rule VP/BE_PRD generates VPs headed by *be* followed by [PRD +] complements of any type and ID rule VP/MOD1 generates VPs post-modified by [ADV +] phrases of any type.

[7] Our lexicon does not distinguish between verbs such as *believe* which are generally treated as taking intensional arguments and ones such as *discover* which take extensional arguments. Simililarly, we cannot distinguish intensional transitives such as *seek* from extensional ones such as *kick*. Improvements to the lexicon would help in this situation although a well-specified knowledge-base might also be sufficient.

Semantically, X2 constituents have much the same representation as the more specified phrases which are their daughters.

6 Constructions

6.1 Passive

The grammar includes a passive metarule which takes as input active VP rules and produces as output passive versions of these. This treatment closely parallels the GPSG85 analysis of passive. The metarule is restricted in various ways so that it only matches ID rules which ought to have a passive version. The output rules lose their N2 object but contain an optional P2[by] instead. (125) and (126) are examples of input and output.

(125)
```
VP/OE_INF1 : ; persuade fido to dance (object equi)
   VP -->
   H[SUBCAT OC_INF, SUBTYPE EQUI], N2[-PRD], VP[TO, AGR N2] :
   2 = [SLASH NOSLASH],
      (lambda (Q) (Q
        (lambda (e) (lambda (x) (1 e x 2
          (3 (lambda (prop) (lambda (ta) (lambda (equa)
             (prop (uqe ((equa some) (e1) (NOTENSE (ta e1)))) 2))))))))
        (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2],
     (lambda (Q) (Q
       (lambda (e) (lambda (x) (lambda (wh) (1 e x (2 wh)
         (3 (lambda (prop) (lambda (ta) (lambda (equa)
          (prop (uqe ((equa some) (e1) (NOTENSE (ta e1)))) (2 wh))))))))))
        (lambda (e2) e2) (lambda (qu) qu))).
```

(126)
```
VP/OE_INF1(PASSIVE/-) :
   VP[PAS, -AUX, -PRO, CONJ NULL, BEGAP -, FIN @11, PAST @12, NEG @17,
      SLASH NOSLASH, COORD -, AGR N2[NFORM NORM, PER @21, PLU @22,
      COUNT @23, CASE @24], ELLIP -] -->
   V[PAS, -INV, -AUX, -PRO, SUBCAT OC_INF, CONJ NULL, H +, FIN @11,
     PAST @12, PSVE +, NEG @17, COORD @, SUBTYPE EQUI, AGR N2[NFORM NORM,
     PER @21, PLU @22, COUNT @23, CASE @24]],
   VP[TO, -PRD, -FIN, CONJ NULL, BEGAP -, PAST @, AUX @, NEG @,
      SLASH NOSLASH, PRO @, COORD @, AGR N2[NFORM NORM, PER @21,
      PLU @22, COUNT @23, CASE @24], ELLIP -] :
   (lambda (Q) (Q
    (lambda (e) (lambda (y) (1 e (uq (some (x1) (entity x1))) y
      (2 (lambda (prop) (lambda (ta155) (lambda (equa156)
        (prop (uqe ((equa156 some) (e1) (NOTENSE (ta155 e1)))) y))))))))
            (lambda (e2) e2) (lambda (qu) qu))).
```

```
VP/OE_INF1(PASSIVE/+) :
   VP[PAS, -AUX, -PRO, CONJ NULL, BEGAP -, FIN @11, PAST @12, NEG @17,
      SLASH @19, COORD -, AGR N2[NFORM NORM, PER @21, PLU @22,
      COUNT @23, CASE @24], ELLIP -] -->
   V[PAS, -INV, -AUX, -PRO, SUBCAT OC_INF, CONJ NULL, H +, FIN @11,
     PAST @12, PSVE +, NEG @17, COORD @, SUBTYPE EQUI, AGR N2[NFORM NORM,
     PER @21, PLU @22, COUNT @23, CASE @24]],
   P2[-PRD, -GER, -POSS, by, CONJ NULL, BEGAP @, NEG @, SLASH @19,
      PRO @, LOC @, WH NO, UB NO, EVER NO, COORD @, MODTYPE @],
   VP[TO, -PRD, -FIN, CONJ NULL, BEGAP -, PAST @, AUX @, NEG @,
      SLASH NOSLASH, PRO @, COORD @, AGR N2[NFORM NORM, PER @21,
      PLU @22, COUNT @23, CASE @24], ELLIP -] :
   2 = [SLASH NOSLASH],
   (lambda (Q) (Q
    (lambda (e) (lambda (y) (1 e (2 (lambda (by) (lambda (np) np))) y
      (3 (lambda (prop) (lambda (ta149) (lambda (equa150)
       (prop (uqe ((equa150 some) (e1) (NOTENSE (ta149 e1)))) y))))))))
     (lambda (e2) e2) (lambda (qu) qu))) :
   2 = [SLASH X2],
   (lambda (Q) (Q
     (lambda (e) (lambda (y) (lambda (wh)
      (1 e (2 (lambda (by) (lambda (np) np)) wh) y
       (3 (lambda (prop) (lambda (ta152) (lambda (equa153)
        (prop (uqe ((equa153 some) (e1) (NOTENSE (ta152 e1)))) y)))))))))
     (lambda (e2) e2) (lambda (qu) qu))).
```

A second metarule, PASSIVE2 generates passive versions of some object raising rules. This metarule has to be separate from the basic passive metarule because ordinary passive outputs are specified as [AGR N2[NFORM NORM]] but the passives of object raising rules must be [AGR @] in order to allow all the different subjects in (127) and (128) which are generated by ID rules VP/OR_INF(PASSIVE2) and VP/OR_AP(PASSIVE2) respectively.

(127) a Fido is believed to be a dog (by Lee)
  b It is believed to bother Lee that fido is a dog
  c There are believed to be dogs in the garden
  d That he fell is believed to bother Lee
  e (For us) to go is believed to be possible

(128) a Fido is considered likely to be a dog (by Lee)
  b It is considered likely to bother Lee that fido is a dog
  c There are considered likely to be dogs in the garden
  d That he fell is considered likely to bother Lee
  e (For us) to go is considered likely to be possible

There are a further set of passive VP rules which exist as basic ID rules instead of being generated by the passive metarules. These ID rules account for various 'odd' passives: for example, 'prepositional passives' (129), ones where the sentential complement is promoted to subject position (130), and ones which are passive versions of extraposed actives (131).

(129) We are being looked at (by everyone) (VP/PP/PASS)

(130) That fido is a dog is believed (by Lee) (VP/SFIN/PASS)

(131) Kim is bothered that Lee likes reading books (VP/NP_SFIN2B/PASS)

The semantics of passive VPs are just like the semantics of actives except that the subject of a passive VP is realised not as the first non-event argument of the verb but as the second. When the P2[by] is present its N2 complement is realised as the first non-event argument. The fact that a P2[by] is non-predicative means that it has a semantics which keeps the translation of the preposition separate from the translation of its N2 complement. The translations in passive rules utilise the N2 meaning but discard the preposition (`(2 (lambda (by) (lambda (np) np))`). When the optional P2[by] is absent, the semantics supplies an existentially quantified agent interpretation: `(uq (some (x1) (entity x1)))`. Passives with a P2[by] have the same semantic representation as the corresponding active (132) and passives without the P2[by] have the same semantic representation modulo the agent argument (133).

(132)
```
"kim helped the abbot"
"the abbot was helped by kim"

(DECL
 (HELP (uqe (some (e1) (PAST e1)))
       (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
       (dd (the (x2) (and (sg x2) (ABBOT x2))))))
```

(133)
```
"the abbot was helped"

(DECL
 (HELP (uqe (some (e1) (PAST e1)))
       (uq (some (x1) (entity x1)))
       (dd (the (x2) (and (sg x2) (ABBOT x2))))))
```

In the semantic translation of those prepositional verbs which can form prepositional passives (ones specified as [SUBTYPE PVERB]) the preposition introducing the object N2 is incorporated into a complex predicate with the verb in both the active and the passive:

(134)
```
"she slept in the bed"
(DECL
 ((CP SLEEP IN) (uqe (some (e1) (PAST e1)))
                (pro (the (x1) (and (sg x1) (female x1))))
                (dd (the (x2) (and (sg x2) (BED x2))))))

"the bed has been slept in"
(DECL
 ((CP SLEEP IN) (uqe (some (e1) (PRES (PERF e1))))
                (uq (some (x1) (entity x1)))
                (dd (the (x2) (and (sg x2) (BED x2))))))
```

6.2 Subject-Auxiliary Inversion

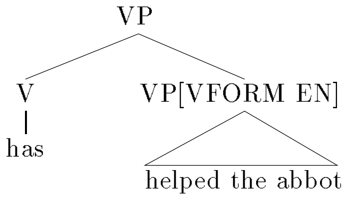
Following GPSG85 we handle subject-auxiliary inversion by means of metarules. There are two metarules for this: SAI and ELL/SAI. In this section we restrict our attention to SAI. The motivation behind ELL/SAI can be found in section 6.7.

The SAI metarule (135) takes the rules for auxiliaries and modals (VP/MODAL1a–VP/BE_CLEFT3) as input. The metarule outputs rules for generating inverted Ss headed by [AUX +] verbs. These S rules are just like the VP inputs except the mother node is an S, the auxiliary is [FIN +], and there is an extra N2[CASE NOM] daughter.

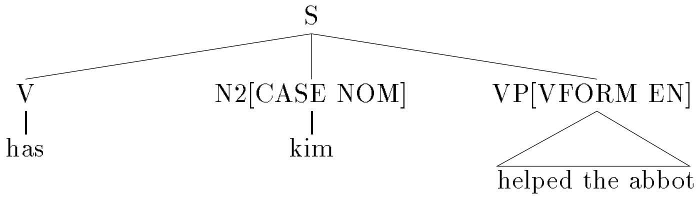
(135)
```
SAI : ; Subject Auxiliary Inversion metarule. Applies to all
      ; non-[ELLIP +] auxiliary rules except for VP/TO. The
      ; listing of possible input SUBCAT values excludes VP/TO.
      ; SLASH propagation is done here too.
   VP[+AUX, VFORM (@, NOT), ELLIP (-, @), COORD (-, @)] -->
   H[SUBCAT (DO, MODAL_BSE, MODAL_INF, FUT, HAVE, BE)],
   X2[~COMP, BEGAP (@, -)], W.
   ==> S[+INV, +FIN, COMP NORM, SLASH @s] -->
   H[+INV], X2[SLASH @s], N2[+NOM, -PRD, SLASH NOSLASH], W :
   1 = [~PAST, NEG -], 5 = [PAST -, NEG -],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (PRES (ta e2))) equa))))))) :
   1 = [~PAST, NEG +], 5 = [PAST -, NEG +],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (PRES (ta e2))) equa))))))) :
   1 = [~PAST, NEG -], 5 = [PAST +, NEG -],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (PAST (ta e2))) equa))))))) :
   1 = [~PAST, NEG +], 5 = [PAST +, NEG +],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (PAST (ta e2))) equa))))))) :
   1 = [PAST FUT, NEG -], 5 = [PAST FUT, NEG -],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (FUT (ta e2))) equa))))))) :
   1 = [PAST FUT, NEG +], 5 = [PAST FUT, NEG +],
      (lambda (s) (s (lambda (prop) (lambda (ta)
        (lambda (equa) (lambda (Q) (Q (lambda (e) (prop e 3))
          (lambda (e2) (FUT (ta e2))) equa))))))).
```

The trees in (136) and (137) indicate the structure associated with one of the input rules, VP/HAVE, and the structure associated with the output, VP/HAVE(SAI).

(136)



(137)



The SAI metarule is restricted such that only NP subjects can be inverted and not S subjects (**does that Lee reads the book bother Kim*). Inverted sentences occurring on their own are recognised as yes-no questions. They also occur as the S which is sister to a preposed wh-constituent (wh-questions, e.g. *which book did Kim read*).

In terms of semantics, the SAI metarule takes the input VP semantics and applies it to the meaning of the N2 subject and an appropriate tense operator, depending on the instantiation of the feature PAST. Apart from VP/WILL, the rules which are input do not have conditions associated with PAST so the first four clauses of the semantic part of the metarule apply to these (i.e. the clauses marked `1 = [~PAST,..]`). These clauses ensure that on the output rules the semantics will be conditional on whether the auxiliary is [PAST −] or [PAST +] and on whether it is [NEG −] (e.g. *can*) or [NEG +] (e.g. *can't*). The fifth and sixth clauses are for VP/WILL which is marked as [PAST FUT] and its output is [PAST FUT] too. To illustrate we show the rule VP/HAVE and the linearised output VP/HAVE(SAI).

(138)
```
VP/HAVE : ; have, has, had (gone etc).
   VP[+AUX, ELLIP -, COORD -] --> H[SUBCAT HAVE],
   VP[EN, PRD -, ELLIP -, COORD -] :
   2 = [SLASH NOSLASH], 1 = [NEG -],
      (2 (lambda (prop) (lambda (ta)
            (lambda (equa) (lambda (Q) (Q prop
                     (lambda (e2) (PERF (ta e2))) equa)))))) :
   2 = [SLASH NOSLASH], 1 = [NEG +],
      (2 (lambda (prop) (lambda (ta)
            (lambda (equa) (lambda (Q) (Q (lambda (e)
                        (lambda (x) (NOT (prop e x))))
                     (lambda (e2) (PERF (ta e2))) equa)))))) :
   2 = [SLASH X2], 1 = [NEG -],
      (2 (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e)
                        (lambda (x) (lambda (wh)
                              (prop e x wh))))
                     (lambda (e2) (PERF (ta e2))) equa)))))) :
```

```
      2 = [SLASH X2], 1 = [NEG +],
         (2 (lambda (prop) (lambda (ta) (lambda (equa)
                  (lambda (Q) (Q (lambda (e)
                         (lambda (x) (lambda (wh)
                               (NOT (prop e x wh)))))
                     (lambda (e2) (PERF (ta e2))) equa)))))).
```

(139)
```
VP/HAVE(SAI) :
      S[+INV, +FIN, +AUX, -PRO, CONJ NULL, VFORM @7, BEGAP -,
         PAST @12, PRD @13, COMP NORM, SLASH @133, WH @39, UB @40,
         EVER @41, COORD -,
         AGR N2[+NOM, NFORM @20, PER @21, PLU @22, COUNT @23],
         UDC -, ELLIP -] -->
      V[+INV, +AUX, -PRO, SUBCAT HAVE, CONJ NULL, VFORM @7, H +,
         FIN @11, PAST @12, PRD @13, PSVE -, NEG @17, COORD @,
         SUBTYPE NONE,
         AGR N2[+NOM, NFORM @20, PER @21, PLU @22, COUNT @23],
         CONEG @74],
      VP[PSP, -FIN, CONJ NULL, BEGAP -, PAST @, AUX @, NEG @74,
         SLASH @133, PRO @, COORD -,
         AGR N2[+NOM, NFORM @20, PER @21, PLU @22, COUNT @23],
         ELLIP -],
      N2[-PRD, -POSS, -ADV, +NOM, -NEG, +SPEC, CONJ NULL, BEGAP -,
         SLASH NOSLASH, NFORM @20, PER @21, PLU @22, COUNT @23,
         PN @, PRO @, PROTYPE @, PART -, AFORM @, DEF @, NUM @,
         WH @39, UB @40, EVER @41, COORD @, REFL @, QFEAT NO,
         DEMON @, COADV @, KIND @] :
      2 = [SLASH NOSLASH], 1 = -NEG[PAST -],
         ((2 (lambda (prop) (lambda (ta)
                     (lambda (equa) (lambda (Q) (Q prop
                              (lambda (e2) (PERF (ta e2)))
                              equa))))))
            (lambda (prop) (lambda (ta) (lambda (equa)
                     (lambda (Q) (Q (lambda (e) (prop e 3))
                        (lambda (e2) (PRES (ta e2))) equa)))))) :
      2 = [SLASH NOSLASH], 1 = -NEG[PAST +],
         ((2 (lambda (prop) (lambda (ta)
                     (lambda (equa) (lambda (Q) (Q prop
                              (lambda (e2) (PERF (ta e2)))
                              equa))))))
            (lambda (prop) (lambda (ta) (lambda (equa)
                     (lambda (Q) (Q (lambda (e) (prop e 3))
                       (lambda (e2) (PAST (ta e2))) equa)))))) :
      2 = [SLASH NOSLASH], 1 = +NEG[PAST -],
         ((2 (lambda (prop) (lambda (ta)
                     (lambda (equa) (lambda (Q) (Q
                              (lambda (e) (lambda (x)
                                    (NOT (prop e x))))
                              (lambda (e2) (PERF (ta e2)))
                              equa))))))
            (lambda (prop) (lambda (ta) (lambda (equa)
                     (lambda (Q) (Q (lambda (e) (prop e 3))
                        (lambda (e2) (PRES (ta e2))) equa)))))) :
```

```
2 = [SLASH NOSLASH], 1 = +NEG[PAST +],
   ((2 (lambda (prop) (lambda (ta)
               (lambda (equa) (lambda (Q) (Q
                        (lambda (e) (lambda (x)
                              (NOT (prop e x))))
                        (lambda (e2) (PERF (ta e2)))
                        equa))))))
      (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e) (prop e 3))
                 (lambda (e2) (PAST (ta e2))) equa)))))) :
2 = [SLASH X2], 1 = -NEG[PAST -],
   ((2 (lambda (prop) (lambda (ta)
               (lambda (equa) (lambda (Q) (Q
                        (lambda (e) (lambda (x)
                              (lambda (wh) (prop e x wh))))
                        (lambda (e2) (PERF (ta e2)))
                        equa))))))
      (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e) (prop e 3))
                  (lambda (e2) (PRES (ta e2))) equa)))))) :
2 = [SLASH X2], 1 = -NEG[PAST +],
   ((2 (lambda (prop) (lambda (ta)
               (lambda (equa) (lambda (Q) (Q
                        (lambda (e) (lambda (x)
                              (lambda (wh) (prop e x wh))))
                        (lambda (e2) (PERF (ta e2)))
                        equa))))))
      (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e) (prop e 3))
                  (lambda (e2) (PAST (ta e2))) equa)))))) :
2 = [SLASH X2], 1 = +NEG[PAST -],
   ((2 (lambda (prop) (lambda (ta)
               (lambda (equa) (lambda (Q) (Q
                        (lambda (e) (lambda (x)
                              (lambda (wh) (NOT
                                    (prop e x wh)))))
                        (lambda (e2) (PERF (ta e2)))
                        equa))))))
      (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e) (prop e 3))
                  (lambda (e2) (PRES (ta e2))) equa)))))) :
2 = [SLASH X2], 1 = +NEG[PAST +],
   ((2 (lambda (prop) (lambda (ta)
               (lambda (equa) (lambda (Q) (Q
                        (lambda (e) (lambda (x)
                              (lambda (wh) (NOT
                                    (prop e x wh)))))
                        (lambda (e2) (PERF (ta e2)))
                        equa))))))
      (lambda (prop) (lambda (ta) (lambda (equa)
               (lambda (Q) (Q (lambda (e) (prop e 3))
                  (lambda (e2) (PAST (ta e2))) equa)))))).
```

The translations here are too complex to be easily grasped but the important thing to notice is that while the VP input rule has four conditions depending on how SLASH and NEG are instantiated, the output S rule has eight conditions, again depending on SLASH and NEG, but also depending on whether the verb is [PAST +] or [PAST –]. This is in accordance with our treatment of S and VP: tense information is incorporated at the S level but not at the VP level.

The translations associated with the mother nodes in (136) and (137) are given in (140).

(140)
```
"has helped the abbot"
(lambda (Q1) (Q1
    (lambda (e1) (lambda (x1)
       (HELP e1 x1 (dd (the (x2) (and (sg x2) (ABBOT x2)))))))
    (lambda (e2) (PERF e2))
    (lambda (qu1) qu1)))


"has kim helped the abbot"
(lambda (Q1) (Q1
    (lambda (e1)
       (HELP e1 (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
                (dd (the (x2) (and (sg x2) (ABBOT x2))))))
    (lambda (e2) (PRES (PERF e2)))
    (lambda (qu1) qu1)))
```

6.3 Extraposition

GPSG85 treats extraposition by means of metarule, but we have chosen to deal with it by means of directly written basic ID rules, partly because of the small number of potential input and output rules and partly because there are a number of odd cases of extraposition not covered by the GPSG85 metarule. Extraposition verbs and adjectives occur with a number of different SUBCAT values but they are all specified as [SUBTYTPE EXTRAP]. The grammar deals with the following kinds of example:

(141)	a	That he left matters	ID rule VP/SFIN3A
		It matters that he left	ID rule VP/SFIN3B
	b	That he left bothers Lee	ID rule VP/NP_SFIN2A
		It bothers Lee that he left	ID rule VP/NP_SFIN2B
		Lee is bothered that he left	ID rule VP/NP_SFIN2B/PASS
	c	That he left matters to me	ID rule VP/PP_SFIN2A
		It matters to me that left	ID rule VP/PP_SFIN2B
	d	To annoy Lee amuses him	ID rule VP/OE_INF2A
		It amuses him to annoy Lee	ID rule VP/OE_INF2B
	e	To see them hurts	ID rule VP/INFA
		It hurts to see them	ID rule VP/INFB

(142)	a	That he left is convenient	ID rule A1/SFIN2A
		It is convenient that he left	ID rule A1/SFIN2B
	b	That he left is clear to me	ID rule A1/PP_SFINA
		It is clear to me that he left	ID rule A1/PP_SFINB
	c	That he leave was necessary	ID rule A1/SBSE2A
		It was necessary that he leave	ID rule A1/SBSE2B
	d	For him to leave was essential	ID rule A1/SINF2A
		It was essential for him to leave	ID rule A1/SINF2B
	e	To be nervous is normal	ID rule A1/VPINF1A
		It is normal to be nervous	ID rule A1/VPINF1B
	f	To be late is typical of him	ID rule A1/PP_VPINF1A
		It is typical of him to be late	ID rule A1/PP_VPINF1B
	g	It is not clear whether he left	ID rule A1/WHSA
		Whether he left is not clear	ID rule A1/WHSB
	h	It is not clear to us whether he left	ID rule A1/PP_WHSA
		Whether he left is not clear to us	ID rule A1/PP_WHSB

In the semantics, the proposition represented by the extraposable V2 complement occurs as an argument of the verb or adjective, such that extraposed/non-extraposed pairs have the same representation:

(143)
```
"that the abbey is falling down doesn't matter"
"it doesn't matter that the abbey is falling down"

(DECL
 (NOT
  (MATTER (uqe (some (e1) (PRES e1)))
   ((CP FALL DOWN) (uqe (some (e2) (PRES (PROG e2))))
    (dd (the (x1) (and (sg x1) (ABBEY x1))))))))
```

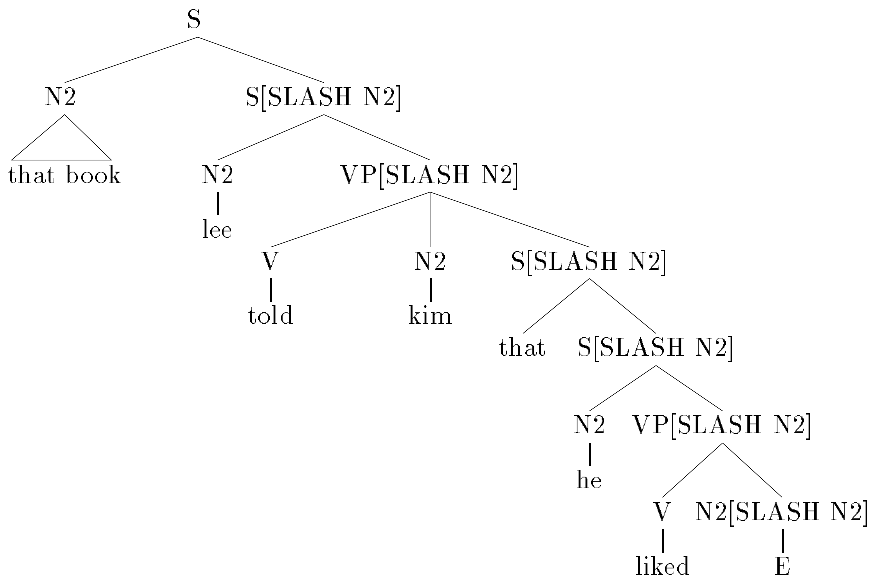
6.4 Unbounded Dependencies

6.4.1 Unbounded Dependency Syntax

Following GPSG85, our treatment of unbounded dependencies involves the use of the category-valued feature SLASH. If a constituent is preposed from its normal position, SLASH is propagated through the tree and links the preposed constituent with the 'gap' (or 'trace') that appears in its normal position. By this means, categorial identity between the preposed constituent and the gap can be maintained. Our treatment achieves much the same kind of results in terms of tree structures and distribution of gaps as the GPSG85 analysis but our method of ensuring the propagation of SLASH through trees is somewhat different.

Before describing SLASH propagation mechanisms in more detail, we demonstrate in (144) a typical example of SLASH distribution in a topicalised sentence:

(144)



A somewhat simplified description of the way in which SLASH propagates is as follows: in all non-lexical ID rules SLASH passes from a mother to a head (e.g. the rules for building the embedded S local trees in (144)). In lexical ID rules (e.g. the rules responsible for the two VPs in (144)) the lexical head cannot be specified with SLASH so it passes to a non-head. This has two consequences: firstly, the only categories which can introduce 'gaps' ([NULL +] categories) will be ones in argument positions (sisters to a lexical head). Secondly, it makes it possible to achieve the effects of some island constraints – for example, the Complex Noun Phrase Constraint which, amongst other things, prevents extractions out of relative clauses. There are further complexities involved in the GPSG85 account such as a treatment of double extractions in parasitic gap constructions but these are largely irrelevant here, although it should be noted that our grammar does not extend to an analysis of parasitic gaps.

In GPSG85 the distribution of SLASH in local trees is not generally controlled by specifications in ID rules but instead is controlled by well-formedness conditions on local trees: SLASH can be instantiated on any category in any tree so long as its presence (and its value) are consistent with the various principles of the theory, in particular the Head Feature Convention, the Foot Feature Principle and Feature Cooccurrence Restrictions. In our system all information about the well-formedness of trees has to reside in the expanded rules which generate them: we do not use principles to discriminate between well- and ill-formed local trees. As a consequence of this, we have to control the distribution of SLASH in very specific terms in the expanded rules.

In previous versions of our grammar (releases 1 and 2), the categories in ID rules were not usually specified for the feature SLASH and on their own they could not describe any long-distance dependencies—a large number of SLASH metarules were needed for this. If one thinks of the ID rules as being basic, 'unslashed' rules, then the SLASH metarules defined derived 'slashed' versions of the relevant rules. The derived rules were identical to basic rules except that the feature SLASH would appear on the mother and on one daughter with the same variable value. For example, the feature SLASH did not appear in the rule VP/NP (`VP --> V N2`) and this rule could therefore not participate in a parse of a sentence containing a preposed direct object such as *sandwiches, Sandy likes E*. In order to parse such a sentence, a derived rule which introduced SLASH onto the mother and onto the N2 object was needed; a metarule generated this new rule on the basis of the old. This approach is very much akin to early GPSG treatments as described in Gazdar et al. (1982) and the major disadvantage to it is the very large increase in the number of rules that it causes.

A more parsimonious approach to long-distance dependencies which has been discussed in the more recent computational linguistics literature is that of 'gap-threading' (e.g. Periera & Shieber, 1987). Here, SLASH must be replaced with two (or more) similar features which appear as a matter of course on all (or most) categories in rules. The distinction between basic 'unslashed' rules and derived 'slashed' rules disappears since one and the same rule can be used for both cases. A more complex pattern of variable bindings ensures that information about the gap propagates up through the tree to link up with the preposed constituent. A gap-threading approach is easily implementable in the GDE but for reasons relating to the creation of LR parse tables (Briscoe & Carroll, 1991, 1992, Carroll & Briscoe, 1992), it was decided not to adopt it for the large scale grammar. Instead a compromise between the two approaches has been adopted.

In this version of the grammar (and in the release 3 version), all non-lexical major categories are specified for the feature SLASH. A series of propagation rules, PSLASH1–PSLASH10 ensure that the value for SLASH on the mother category of an ID rule is bound to the value of SLASH on just one of the daughters in the rule. In non-lexical ID rules, this daughter will be the [BAR 1] or [BAR 2] head. In lexical ID rules which introduce just one complement daughter apart from the lexical head, the complement daughter's SLASH feature is bound to the mother. In lexical ID rules which introduce more than one complement daughter, just one of these daughters will be bound to the mother. (145) contains some examples:

(145)
```
VP/NP :          VP[SLASH @x] --> V, N2[SLASH @x]
VP/PPa :         VP[SLASH @x] --> V, P2[SLASH @x]
VP/NP_PHRA :     VP[SLASH @x] --> V, N2[SLASH @x], Prt.

VP/OE_INF1 :     VP[SLASH @x] --> V, N2[SLASH @x], VP[SLASH NOSLASH].
VP/OE_AP :       VP[SLASH @x] --> V, N2[SLASH @x], A2[SLASH NOSLASH].
VP/PP_SFIN1a :   VP[SLASH @x] --> V, P2[SLASH @x], S[SLASH NOSLASH].
```

The specification [SLASH NOSLASH] is in fact an alias for [SLASH [NOSLASH +]] and this provides a suitable category as a value for SLASH on constituents which do not contain a gap. In all the cases in (145) the rules are able to generate normal 'non-gappy' constituents where the `@x` is simple instantiated to [SLASH NOSLASH]. For the first group, these rules are also all that is needed for generating 'gappy' versions as well: if the complement daughter either contains a gap or is a gap then its value for SLASH will be instantiated to a [BAR 2] category and this will propagate up to the mother. The rules in the second group generate only a subset of possible 'gappy' examples. For example, the version of VP/OE_INF1 above will generate the sentence *Kim, Lee persuaded E to leave* where the N2 object of the verb is preposed. It does not, however, propagate SLASH between the VP complement and the mother as would be needed in *Kim, Lee persuaded Sandy to help E*. The specification [SLASH NOSLASH] on the second daughters in the second group of rules is provided by the set of default rules DSLASH1–DSLASH12.

In order to generate the second possibility, a new derived rule has to be created by means of metarules (see metarules MSLASH1a–MSLASH5b). The derived rule is one where SLASH on the mother is specified as having to be a [BAR 2] value (the alias X2) as is SLASH on the second daughter. The values for features on these X2 values are bound between mother and second daughter. The derived version of VP/OE_INF1 is:

```
(146)  VP/OE_INF1(MSLASH1a) :
         VP[-AUX, -PRO, CONJ NULL, VFORM @7, BEGAP -, FIN @11, PAST @12,
            PRD @13, NEG @17, SLASH X2[N @1, V @2, SUBJ @4, PRD @85,
            NFORM @20, PER @130, PLU @128, COUNT @129, CASE @149,
            PFORM @30, LOC @31, GERUND @32, QUA @34, ADV @37], COORD -,
            AGR N2[NFORM NORM, PER @21, PLU @22, COUNT @23, CASE @24],
            ELLIP -] -->
         V[-INV, -AUX, -PRO, SUBCAT OC_INF, CONJ NULL, VFORM @7, H +,
            FIN @11, PAST @12, PRD @13, PSVE -, NEG @17, COORD @,
            SUBTYPE EQUI, AGR N2[NFORM NORM, PER @21, PLU @22,
            COUNT @23, CASE @24]],
         N2[-PRD, -POSS, -ADV, +ACC, -NEG, +SPEC, CONJ NULL, BEGAP -,
            SLASH NOSLASH, NFORM NORM, PER @111, PLU @112, COUNT @113,
            PN @, PRO @, PROTYPE @, PART -, AFORM @, DEF @, NUM @,
            WH NO, UB NO, EVER NO, COORD @, REFL @, QFEAT NO, DEMON @,
            COADV @, KIND @],
         VP[TO, -PRD, -FIN, CONJ NULL, BEGAP -, PAST @, AUX @, NEG @,
            SLASH X2[N @1, V @2, SUBJ @4, PRD @85, NFORM @20, PER @130,
            PLU @128, COUNT @129, CASE @149, PFORM @30, LOC @31,
            GERUND @32, QUA @34, ADV @37], PRO @, COORD @,
            AGR N2[NFORM NORM, PER @111, PLU @112, COUNT @113, CASE @],
            ELLIP -] :
       (lambda (Q) (Q
          (lambda (e) (lambda (x) (lambda (wh)
            (1 e x 2 (3 (lambda (prop) (lambda (ta) (lambda (equa)
                         (prop (uqe ((equa some) (e1) (NOTENSE (ta e1))))
                               2 wh)))))))))
         (lambda (e2) e2) (lambda (qu) qu))).
```

This method of only generating derived rules where more than one propagation path through a rule is possible has led to considerable reductions in grammar size. The compiled version of the release 2 grammar had approximately 1,500 object grammar rules. The compiled version of this grammar has 782 object grammar rules.

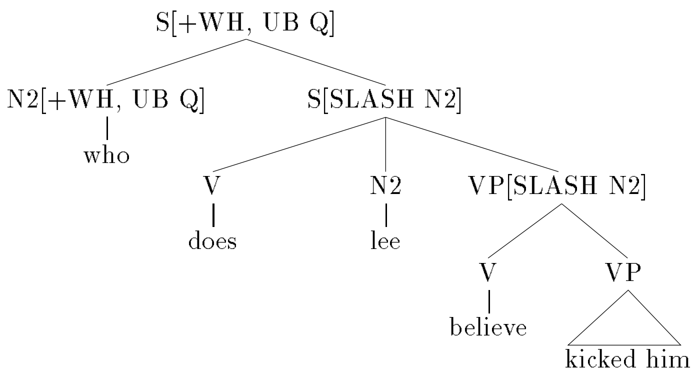
The SLASH specifications for the top and bottom parts of an unbounded dependency have to be specifically catered for. The ID rules S/NP_UDC1–S/AP_UDC2 describe the top of the dependency where a preposed N2, P2 or A2 precedes an S whose SLASH value is the same as the preposed constituent. We introduce gaps at the bottom of the dependency by means of the ID rules TRACE1–TRACE8 which allow N2s, P2s and A2s which are specified with a SLASH value of the same category as themselves to be realised as a gap. That is, N2[SLASH N2], A2[SLASH A2] and P2[SLASH P2] can be gaps. The rule in (147) is the one for N2[SLASH N2].

(147)
```
TRACE1 : ; an NP gap
    N2[+SPEC, -ADV, SLASH N2[NFORM NORM, -ADV, PLU @pl, COUNT @co,
            PER @pe, CASE @ca, PRD @pr], NFORM NORM, PLU @pl,
       COUNT @co, PER @pe, CASE @ca, PRD @pr, BEGAP +] -->
    [NULL +] : 0 = [PRD -], (lambda (wh) wh) :
    0 = [PRD +], (lambda (x) (lambda (wh) (wh x))).
```

Notice that features relevant to N2s are bound between the N2 and the category value of SLASH—this ensures that whatever restrictions exist on the gap position are propagated up to the preposed N2 (for example, if the gap is in a position which requires a predicative N2 then this will force the preposed N2 to be predicative). The feature specification [NULL +] which defines the daughter in (147) is a special specification which causes the parser to insert an 'E' node in that position.

It has been argued in the GPSG literature that preposed subjects of embedded clauses do not leave behind a gap. GPSG85 has a metarule, STM2, which terminates a SLASH path at the position of an embedded subject but which doesn't introduce a gap: our analysis follows suit (see metarules STM2a and STM2b). In sentences such as *who does Lee believe kicked him*, although the subject of *kicked* is missing, there is no empty category in this position. Instead, the embedded S complement of *believe* is replaced by a VP. The structure for this sentence is given in (148).

(148)



There are a few places in the grammar where SLASH introduction, propagation or elimination does not happen in quite the way described above. The comparative rules involve departures from the norm (see section 6.8 below). The coordination rules are also non-standard because they involve propagating SLASH onto all conjunct daughters (see section 6.7). As we explained above, SLASH cannot usually propagate to modifier constituents since these are not sisters of lexical heads. Arguably, examples such as *enthusiastically we set off for the zoo* involve preposing of a VP modifier but since a modifier receives the same interpretation whether it attaches to S or VP and whether it occurs initially or finally (as a predicate which takes the event variable as an argument), there is no motivation for producing such an analysis. There is one case, however, which does seem to involve preposing of a modifier, occurring in some it-cleft constructions: *when was it E that you last saw him.* For just these cases we have introduced special rules for terminating SLASH on modifiers (TRACE4 and TRACE5).

In the next subsection we discuss the semantics of unbounded dependencies and then in the rest of the section we outline briefly the analysis of some specific types of unbounded dependency construction.

6.4.2 Unbounded Dependency Semantics

The semantic translations associated with unbounded dependencies are intuitively quite simple although they have proved rather complex to implement. Basically, a constituent which has a non-NOSLASH value for SLASH will be looking for one argument more than its [SLASH NOSLASH] counterpart, i.e. over and above its other needs, it is looking for a constituent to fill the gap that it contains. For this reason, in all the rules where SLASH may become instantiated either to NOSLASH or to an actual category, the grammar has to provide a translation for each of those possibilities.

(149)
```
VP/NP : ; abandons his friends
   VP --> H[SUBCAT NP], N2[-PRD] :
   2 = [SLASH NOSLASH],
    (lambda (Q) (Q
       (lambda (e) (lambda (x) (1 e x 2)))
       (lambda (e2) e2)
       (lambda (qu) qu))) :
   2 = [SLASH X2],
    (lambda (Q) (Q
       (lambda (e) (lambda (x) (lambda (wh) (1 e x (2 wh)))))
       (lambda (e2) e2)
       (lambda (qu) qu))).
```

(149) demonstrates this point: the N2 object of a transitive may end up instantiated as either [SLASH NOSLASH] or as [SLASH X2] (where X2 must be realised as N2 if the whole N2 is a gap but may be any of N2, P2 or A2 if the gap is lower down inside the N2). When there is no gap, then the predicate translating the verb is still looking for an event argument and a subject argument—`(lambda (e) (lambda (x) (1 e x 2)))` (remember that the first argument of `Q` is always the predicate-argument structure). When the N2 either is a gap or contains a gap then the information about that gap is still to be found outside of the VP and the `(lambda (wh)` part of the predicate-argument structure is waiting to pick up that information. The `(lambda (wh)` associated with a gap is threaded up from the gap site with each successive translation of local trees until the antecent of the gap is reached and can be given to the `(lambda (wh)` as an argument. In (150) we show the successive translations of the local trees in *Lee Kim helps E* in order that the technique should become clear. By way of contrast, (151) shows the successive translations of its non-topicalised counterpart, *Kim helps Lee.*

(150)
```
"E"
(lambda (wh1) wh1)
```

```
"helps E"
(lambda (Q1) (Q1
   (lambda (e1) (lambda (x1) (lambda (wh1)
      (HELP e1 x1 wh1))))
   (lambda (e2) e2)
   (lambda (qu1) qu1)))

"kim helps E"
(lambda (Q1) (Q1
   (lambda (e1) (lambda (wh1)
      (HELP e1
           (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
            wh1)))
   (lambda (e2) (PRES e2))
   (lambda (qu1) qu1)))

"lee kim helps E"   (S node)
(lambda (Q1) (Q1
   (lambda (e1)
      (HELP e1
            (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
            (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))))
   (lambda (e2) (PRES e2))
   (lambda (qu1) qu1)))

"lee kim helps E"  (Top category)
(DECL
 (HELP (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))))
```

(151)

```
"lee"
(name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))

"helps lee"
(lambda (Q1) (Q1
   (lambda (e1) (lambda (x1)
      (HELP e1
            x1
           (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))))
   (lambda (e2) e2)
   (lambda (qu1) qu1)))

"kim helps lee"   (S node)
(lambda (Q1) (Q1
   (lambda (e1)
      (HELP e1
            (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
            (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))))
   (lambda (e2) (PRES e2))
   (lambda (qu1) qu1)))
```

```
"kim helps lee"   (Top category)
(DECL
 (HELP (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))))
```

The techinique of passing up the requirement for information about gaps is conceptually quite simple but it has involved a high degree of verbosity in most rules since the translations have to be made conditional on the presence or absence of a gap. It has also made very complex the statement of the semantic part of the metarules which produce alternative slashed versions of some rules. We can demonstrate the problem by considering the rule in (152).

(152)
```
VP/NP_SFIN1 : ; tells her (that) he can do it
   VP --> H[SUBCAT NP_SFIN, SUBTYPE NONE],
          N2[-PRD, NFORM NORM], S[+FIN] :
   2 = [SLASH NOSLASH],
    (lambda (Q) (Q
      (lambda (e) (lambda (x)
        (1 e x 2
           (3 (lambda (prop) (lambda (ta) (lambda (equa)
             (prop (uqe ((equa some) (e3) (ta e3)))))))))))
      (lambda (e2) e2)
      (lambda (qu) qu))) :
   2 = [SLASH X2],
    (lambda (Q) (Q
      (lambda (e) (lambda (x) (lambda (wh)
        (1 e x (2 wh)
           (3 (lambda (prop) (lambda (ta) (lambda (equa)
             (prop (uqe ((equa some) (e3) (ta e3))))))))))))
      (lambda (e2) e2)
      (lambda (qu) qu))).
```

In this rule SLASH propagates between the VP and the N2 object, but since another propagation path to the S complement is possible, a second slashed version of the rule is generated by metarule and has the syntax in (153).

(153)
```
VP/NP_SFIN1(MSLASH1b) :
   VP[-AUX, -PRO, CONJ NULL, VFORM @7, BEGAP -, FIN @11, PAST @12,
      PRD @13, NEG @17, SLASH X2[N @1, V @2, SUBJ @4, PRD @85,
      NFORM @20, PER @111, PLU @112, COUNT @113, CASE @114,
      PFORM @30, LOC @31, GERUND @32, QUA @34, ADV @37], COORD -,
      AGR N2[NFORM NORM, PER @21, PLU @22, COUNT @23, CASE @24],
      ELLIP -] -->
   V[-INV, -AUX, -PRO, SUBCAT NP_SFIN, CONJ NULL, VFORM @7, H +,
      FIN @11, PAST @12, PRD @13, PSVE -, NEG @17, COORD @,
      SUBTYPE NONE, AGR N2[NFORM NORM, PER @21, PLU @22,
      COUNT @23, CASE @24]],
   N2[-PRD, -POSS, -ADV, +ACC, -NEG, +SPEC, CONJ NULL, BEGAP -,
      SLASH NOSLASH, NFORM NORM, PER @, PLU @, COUNT @, PN @,
      PRO @, PROTYPE @, PART -, AFORM @, DEF @, NUM @, WH NO,
      UB NO, EVER NO, COORD @, REFL @, QFEAT NO, DEMON -,
      COADV @, KIND @],
```

```
S[-INV, +FIN, CONJ NULL, VFORM NOT, BEGAP -, PAST @, PRD @,
   AUX @, COMP @, SLASH X2[N @1, V @2, SUBJ @4, PRD @85,
   NFORM @20, PER @111, PLU @112, COUNT @113, CASE @114,
   PFORM @30, LOC @31, GERUND @32, QUA @34, ADV @37], PRO @,
   WH NO, UB NO, EVER NO, COORD @, AGR @, UDC -, ELLIP -] :
```

Here the SLASH propagation path is between the mother and the S daughter and we a need a semantic translation to reflect this path, i.e. we need the following translation where the `wh` variable is an argument of the translation of the S.

(154)
```
(lambda (Q) (Q
    (lambda (e) (lambda (x) (lambda (wh)
        (1 e x 2
           (3 (lambda (prop) (lambda (ta) (lambda (equa)
             (prop (uqe ((equa some) (e3) (ta e3))) wh)))))))))
    (lambda (e2) e2)
    (lambda (qu) qu))).
```

The problem we have is that in metarules the output semantics is defined in terms of the input semantics. Usually the lambda calculus is sufficient to define a function which takes the input semantics and transforms it into the output but in this case, however, there is no lambda expression which we can write which can take either of the two translations in (152) and turn it into (154). To solve the problem we have had to use Lisp to define operators to achieve the appropriate transformations: slash-operator-1, slash-operator-2 and slash-operator-3.[8] The definition of the metarule MSLASH1b is as follows:

(155)
```
MSLASH1b : ; takes an input where mother and N2 share values for
           ; SLASH and produces an output where mother and some
           ; other [BAR 2] daughter are [SLASH X2] and the N2 is
           ; [SLASH NOSLASH]. See file cmeta-ops for definition
           ; of slash-operator-2.
    [SLASH @, ~INV] --> [H +, BAR 0], N2[-PRD, SLASH @],
    X2[SLASH NOSLASH, UB NO], W.
    ==> [SLASH X2] --> [H +, BAR 0], N2[-PRD, SLASH NOSLASH],
       X2[SLASH X2, BEGAP @b], W :
    2 = [SLASH NOSLASH], slash-operator-2.
```

6.4.3 Topicalisation

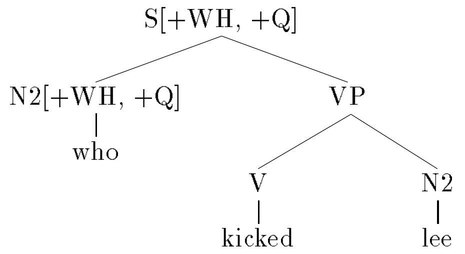
Topicalised sentences are the simplest type of unbounded dependency. The topicalised constituent is introduced at the top of a tree by the S/UDC rules and the SLASH is passed down the tree until it is terminated. The tree in (144) is an example of a topicalisation structure. Semantically, topicalisations receive exactly the same representation as their non-topicalised counterparts.

[8] The definitions of these operators can be found in the file 'cmeta-ops'. They must be loaded prior to grammar compilation.

6.4.4 Wh-questions

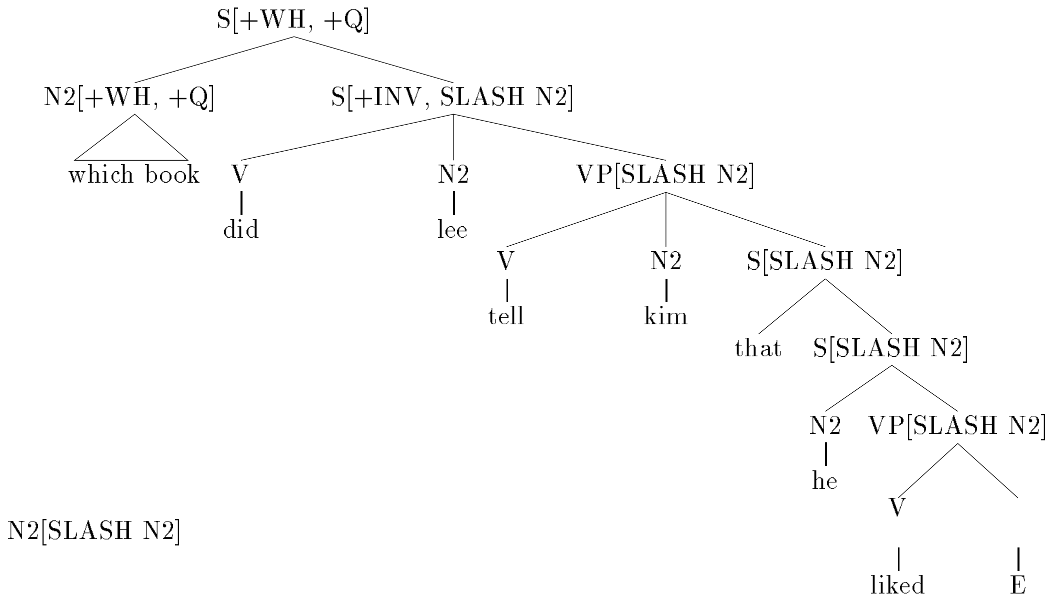
Matrix subject wh-questions such as *who kicked Lee* do not involve the SLASH feature or gaps. They receive a structure as in (156).

(156)



All other kinds of wh-questions are very similar to topicalisations in that they too involve a preposed constituent and a SLASH passed down the tree. The difference is that the preposed constituent has values other than NO for the features WH, UB and EVER which pass up to the mother node. In addition, the S[SLASH X2] which is the sister of the preposed constituent is inverted. The tree in (148) is an example of a wh-question, as is the tree in (157).

(157)

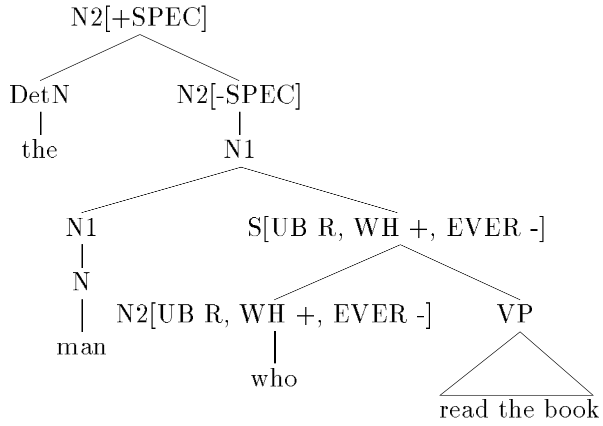


Some comment is in order about the propagation of the features WH, UB and EVER. Together, these features describe the 'wh-ness' of constituents. In releases 1 and 2, a wh version of a rule was derived from the basic non-wh version by means of metarule. This paralleled the way slashed versions of rules derived from non-slashed versions. In this grammar (and in release 3) the changes in the way WH, UB and EVER are treated parallel the changes in the treatment of SLASH. WH, UB and EVER now appear on almost all non-lexical major categories. When a constituent is not a wh-constituent, for example, the N2 *the cat*, the features WH, UB and EVER all have the value NO. A wh-N2, such as *which cat* is specified as [WH +, UB Q, EVER -]. The source of the wh-ness of a constituent is a wh-word which may appear quite deep inside that constituent: e.g. *the height of the* *letters on the covers of which*. In many rules, the path of propagation of wh-ness is quite straightforward in the sense that it can pass from just one daughter to the mother. For example, in a rule of the form `N1 --> A2, H1`, the wh-features pass from the head to the mother. The propagation rules PFOOT1–PFOOT8 set up a pattern of propagation in rules which are affected by wh-ness so that the mother and one daughter are bound. The default rules DFOOT1–DFOOT7 default other categories in rules to [WH NO, UB NO, EVER NO]. As is the case with SLASH propagation, in some rules more than one propagation pattern is possible. For example in the rule which expands an N2 as a determiner plus a nominal head, the wh-ness may derive either from the determiner (*which picture of Lee*) or from the head (*a picture of whom*). Where more than one propapagation pattern can occur, more than one version of the rule has been written (see for example, N2+/DET1a and N2+/DET2). Second versions could have been generated by metarule, as is done with SLASH, but in fact the number of rules affected is so small that it is easier to produce them by hand.

6.4.5 Relative Clauses

For most relative clauses, our account is the same as the GPSG85 account. Matrix subject relatives receive structures like the following:

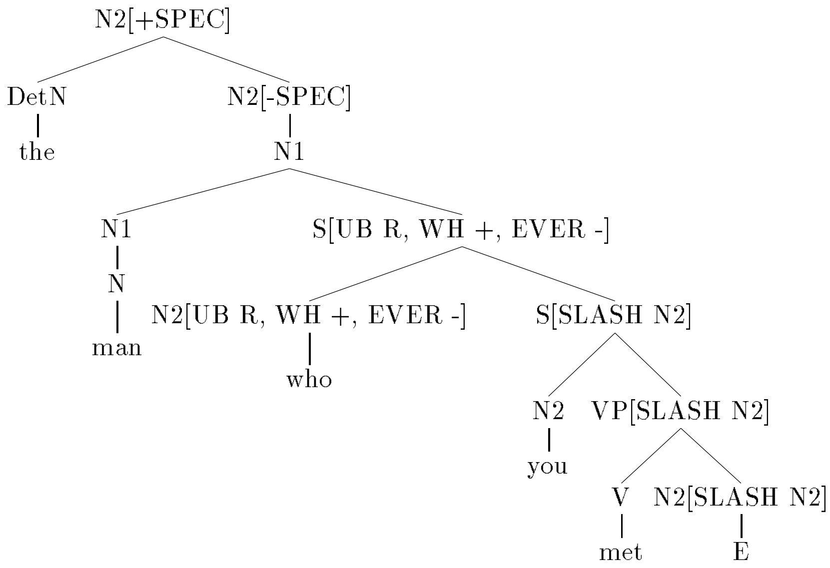
(158)



Notice that SLASH and gaps are not involved in matrix subject relatives. The relative pronoun is not preposed but simply occupies the position of the subject. A similar structure is assigned to examples with a *that* relative pronoun such as *the tree that was blown down*. The only difference is that the WH feature on the relative pronoun and the relative clause has the value −.

Non-matrix-subject relatives receive analyses like (159).

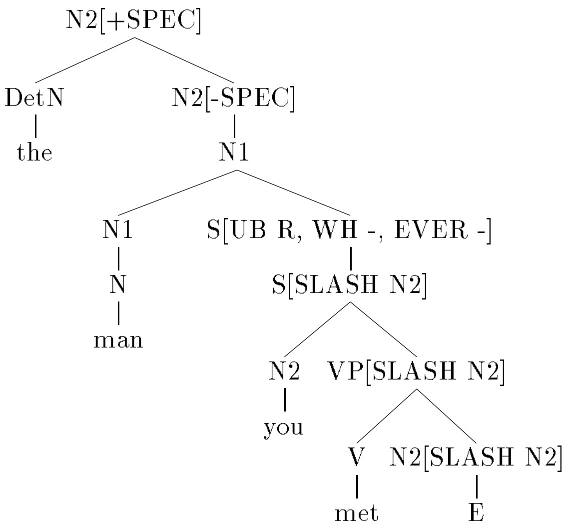
(159)



Here the relative pronoun is preposed and its sister is an S[SLASH N2]. Non-matrix-subject relatives with *that* (*the window that you broke E*) are analysed in the same way except that the WH feature has a negative value again. The grammar also analyses relatives where the relativiser is a P2 and not an N2 as in *the table on which you put the book.* The grammar is restricted so as to only allow P2[WH +] relativisers, so examples such as **the table on that you put the book* will be blocked.

'That'-less relatives receive the analysis in (160):

(160)



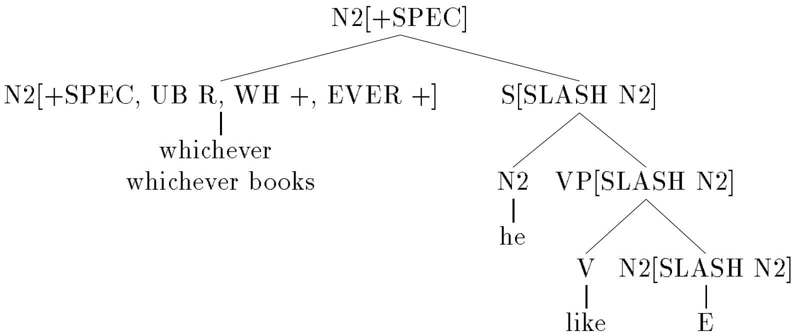
Semantically, relative clauses behave just like any other nominal modifier: they are predicates which are applied to the nominal. Although relative clauses are syntactically complete Ss, they are semantically incomplete since they are looking to be predicated of the nominal in order to be completed. In (161) we show the translation of the N2 *the man who he helped E* and in (162) we show the translations associated with relevant subparts of this N2.

```
(161)   "the man who he helped E"
        (dd (the (x1)
           (and (sg x1)
                (and (MAN x1)
                     (HELP (uqe (some (e1) (PAST e1)))
                           (pro (the (x2) (and (sg x2) (male x2))))
                           x1))))))
```

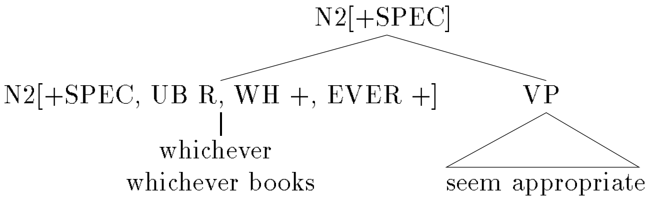
```
(162)   "who he helped E"
        (lambda (Q1) (Q1
          (lambda (e1) (lambda (x1)
              (HELP e1 (pro (the (x2) (and (sg x2) (male x2))))
                       x1)))
          (lambda (e2) (PAST e2))
          (lambda (qu1) qu1)))

        "man who he helped E"
        (lambda (x1) (and (MAN x1)
                          (HELP (uqe (some (e1) (PAST e1)))
                                (pro (the (x2) (and (sg x2) (male x2))))
                                x1)))
```

Free relatives receive the analyses in (163) and (164) (see ID rules N2+/FREEREL and N2+/FREEREL2).

(163)



(164)



The semantic translations these receive are given in (165).

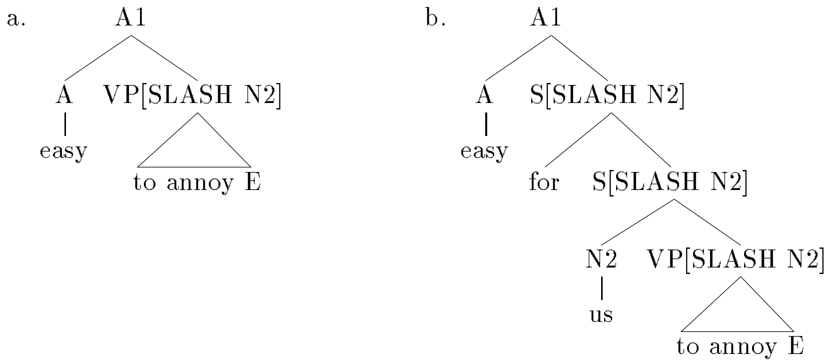
```
(165)   "whichever he likes E"
        (dd (the (_wh1)
              (and (entity _wh1)
                   (LIKE (uqe (some (e1) (PRES e1)))
                         (pro (the (x1) (and (sg x1) (male x1))))
                         _wh1))))
```

```
"whichever books he likes E"
(dd (the (_wh1)
     (and (pl _wh1)
          (BOOK _wh1)
          (LIKE (uqe (some (e1) (PRES e1)))
                (pro (the (x1) (and (sg x1) (male x1))))
                _wh1))))

"whichever seem appropriate"
(dd (the (_wh1)
      (and (entity _wh1)
           (SEEM (uqe (some (e1) (PRES e1)))
                 (BE (uqe (some (e2) (NOTENSE e2)))
                 (APPROPRIATE _wh1 (degree unknown)))))))

"whichever books seem appropriate"
(dd (the (_wh1)
      (and (pl _wh1)
           (BOOK _wh1)
           (SEEM (uqe (some (e1) (PRES e1)))
                 (BE (uqe (some (e2) (NOTENSE e2)))
                 (APPROPRIATE _wh1 (degree unknown)))))))
```

6.4.6 Tough Adjectives

The rules A1/TOUGH1 and A1/TOUGH2 introduce 'tough' adjectives which appear with a VP[SLASH N2[CASE ACC]] or an S[SLASH N2[CASE ACC]] complement. These generate examples such as *Lee is easy to annoy E*, *Lee is easy for us to annoy E* as in (166). The [CASE ACC] restriction blocks examples like **Lee is easy to believe likes cats*, **Lee is easy for us to believe likes cats.*

(166)



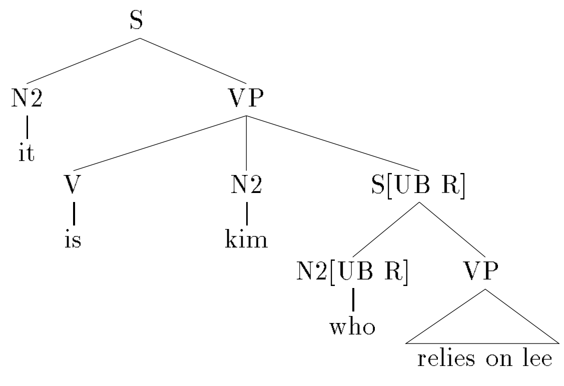
In the semantics, the subject of the 'tough' adjective serves as the antecedent of the object gap in the complement (and does not act as an argument of the adjective). When a *for*-phrase is present, its N2 is interpreted as the subject argument of the complement, and when it is absent, a suitable subject is provided (`(pro (the (z1) (entity z1)))`). (167) illustrates:

(167)
```
"lee is so easy for kim to annoy E"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EASY
   (ANNOY (uqe (some (e2) (NOTENSE e2)))
          (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
          (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))
   (degree so))))

"lee is so easy to annoy E"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EASY
   (ANNOY (uqe (some (e2) (NOTENSE e2)))
          (pro (the (z1) (entity z1)))
          (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1)))))
   (degree so))))
```

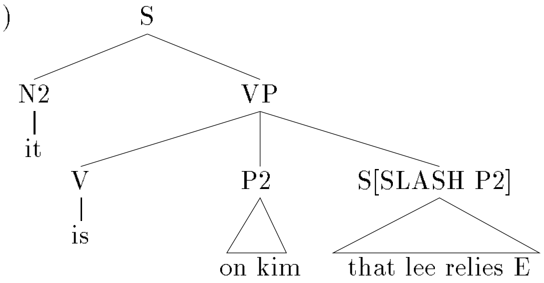
6.4.7 It-Clefts

The rules VP/BE_CLEFT1–VP/BE_CLEFT3 generate it-clefts. VP/BE_CLEFT1 is responsible for examples such as *it is Lee who/that relies on Kim*, *it is Lee who/that Kim relies on*. Here, an N2 occurs in focused position and is followed by a relative clauses to which it is related. VP/BE_CLEFT2 deals with examples such as *it is on Lee (that) Kim relies* where a P2 occurs on focused position and is followed by an S[SLASH P2]. VP/BE_CLEFT3 is responsible for examples such as *it was in the park that he found it*, *it was seldom that he saw her* and places adverbials in focused position followed by an S with no gap. Some example trees:

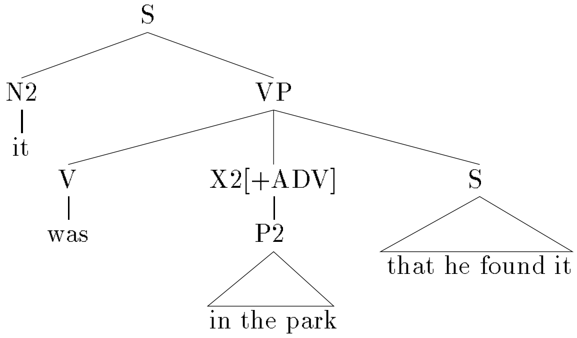
(168)



(169)



(170)



Semantically it-clefts receive the same translation as their non-clefted counterparts.

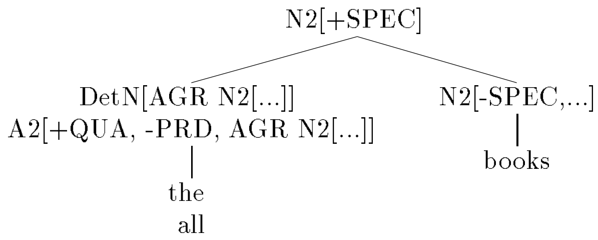
6.5 Agreement and Control

6.5.1 Syntactic Agreement

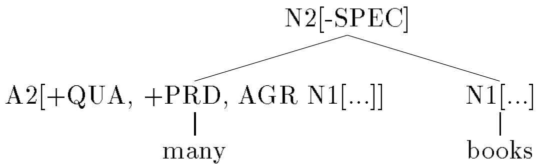
In GPSG85, the category valued AGR feature is used both to ensure subject-verb concord and to ensure an appropriate selection of subject for control predicates. The positions where AGR must be bound are defined by reference to the semantic type of a category. Once these positions have been identified, the Control Agreement Principle ensures appropriate bindings. In our grammar agreement positions must be identified by the grammar writer. Once identified, propagation rules ensure binding of features on the category value of AGR on some node to features on a sister of that node. The following agreement positions have been identified:

(171)

a.

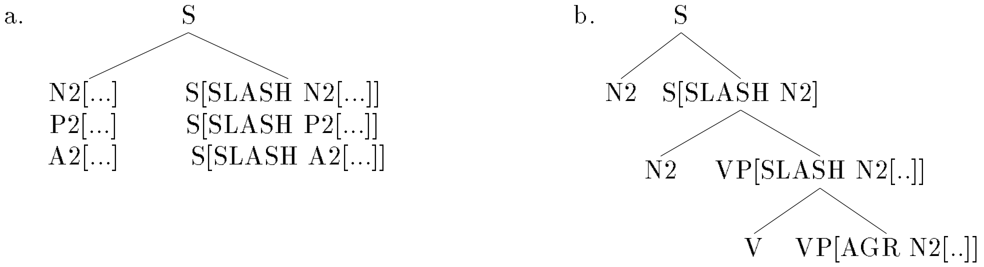


b.

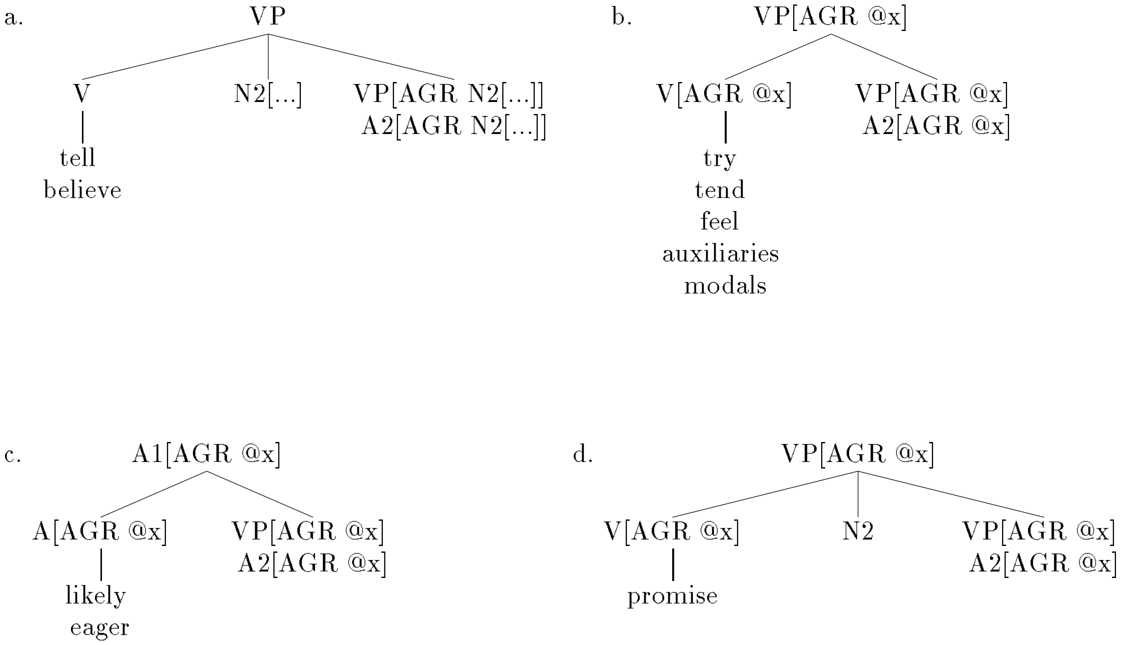


(172)

(173)



(174)



In (171a) and (171b), determiners and quantifying adjectives must agree with their nominal sisters. Propagation rule AGR/NOM ensures that the values for the features PLU and COUNT on the nominal category which is the value of AGR on the specifier are bound to the values for these features on the nominal head, thus blocking examples such as **a books*, **both book*, **many book* and **much books.*

In (172a) and (172b) there must be agreement between a VP and its N2 or V2 subject. In the case of the N2 subject, propagation rule AGR/NP_VP binds the features PER, PLU, CASE and NFORM (and somewhat redundantly, COUNT) between the AGR N2 on the VP and the N2 subject. The binding of the PER and PLU features ensures agreement whilst the binding of the NFORM feature ensures that a verb will only occur with the type of N2 subject it requires. For example, one entry for the verb *bother* is specified as [AGR N2[NFORM IT]] so the binding of AGR ensures that it will only occur with a pleonastic *it* subject. Note that in inverted sentences agreement occurs between the subject N2 and the inverted verb. Propagation rule AGR/INV achieves the appropriate binding of values. In the case of the V2 subject, the values for the features SUBJ, FIN and VFORM are bound by propagation rule AGR/V2_VP. In this way, verbs or adjectives which require one particular type of V2 subject (finite S, infinitival S or infinitival VP) will only be able to appear with that type of subject.

(173a) and (173b) are structures in which features on the category value of SLASH must be bound to features on a neighbouring category. The trees schematized in (173a) show the tops of unbounded dependency constructions where preposed N2, P2, and A2 must be linked to the value of SLASH on the sister S node. Propagation rules SLASH/AGR_NP1, SLASH/AGR_PP1 and SLASH/AGR_AP are the rules which ensure this link. (173b) is the kind of structure resulting from the slash termination metarules STM2a and STM2b mentioned in the previous section. Here a [SLASH N2[+NOM]] is not terminated by a gap but instead the finite S complement in which it would have appeared is replaced by a finite VP. In order to ensure agreement between the preposed N2 and the VP whose subject it is, features such as PLU, PER and CASE must be bound between the N2 value of SLASH and the N2 value of AGR on that VP—propogation rule SLASH/AGR_NP3 does just that. In this way we block ill-formed examples such as **those children I believe lives in our street.*

The tree in (174a) exemplifies the configuration in which object control verbs appear. Here the N2 object also functions as the subject argument of the VP or A2 complement. Accordingly, propagation rule OBJ_CONTROL binds the values for the features PER, PLU, NFORM and COUNT on the [AGR N2] on the VP or A2 to values for these features on the N2 object. The only feature that is really of significance here is NFORM since this restricts the type of N2 which can occur as object. In object equi cases (see ID rules VP/OE*) the N2 must be [NFORM NORM] so the NFORM value on the AGR N2 on the VP or A2 must also be NORM. This ensures that examples such as *persuade Lee to read* will be parsed whilst **persuade it to bother Lee that Kim reads* will not. In the case of object raising (see ID rules VP/OR*), the NFORM value on the N2 is unspecified with the result that all of *believe Lee to be a fool*, *believe it to bother Lee that Kim reads*, *believe there to be elephants in the garden* will be parsed. A number of different types of VP complement are possible: infinitival ones (*believe Lee to like bananas*); -ing form ones (*see Lee running to the shop*); base form ones (*make Lee run to the shop*) and passive ones (*get Lee arrested*). The verbs which head object control VPs determine the form of their complement by being specified differently for their SUBCAT feature. The verbs just given as examples are specified as [SUBCAT OC_INF], [SUBCAT OC_ING], [SUBCAT OC_BSE] and [SUBCAT OC_PASS] respectively. Similarly, it is the verbs which determine whether the construction is a raising or equi construction: raising verbs are [SUBTYPE RAIS] and equi verbs are [SUBTYPE EQUI].

The trees in (174b) and (174c) are subject control environments. Here there is no N2 object to be interpreted as the subject of the VP or A2 complement. Instead, the subject of the mother VP or A1 must be interpreted as the subject of the complement – the '@x' in the trees indicates that they share the same AGR value and hence the same subject. The SUBJ_CONTROL propagation rules ensure this identity of values. In the case of subject equi, the control predicate (such as *try*, *feel* or *eager*) is specified as [AGR N2[NFORM NORM]] and this means that the only kind of VP or A2 complement that can occur here will also have to be [AGR N2[NFORM NORM]]. Examples such as **it tries to bother Lee that Kim reads*, **there feel likely to be elephants in the garden*, **that Kim reads is eager to bother Lee* will therefore be blocked. In the case of subject raising predicates (such as *tend*, *likely* and the auxiliaries and modals) there is no restriction on the type of subject that these predicates can occur with and hence no restriction on the AGR value of their VP or A2 complements. Subjects can be V2s of any type or N2s of any type—in a sense these predicates simply inherit the agreement properties of their complements. The result is that sets of sentences such as *Kim seems to read*, *it seems to bother Kim that Lee reads*, *there seem to be elephants in the garden*, *that Kim reads seems to bother Lee*, *for us to go seems to be possible*, *to go seems to be possible* are all accepted by the grammar. As with the object control verbs, it is subject control verbs or adjectives which determine the form of their controlled complement. Subject control verbs may select infinitival VP complements (*try to help*—[SUBCAT SC_INF]); -ing form VP complements (*like helping*—[SUBCAT SC_ING]); base form VP complements (*dare be rude*—[SUBCAT SC_BSE]); passive VP complements (*get arrested*–[SUBCAT SC_PASS] and A2 complements (*seem stupid*–[SUBCAT SC_AP]). Subject control adjectives may select infinitival VP complements (*eager to help*—[SUBCAT SC_INF]); -ing form VP complements (*busy helping*—[SUBCAT SC_ING]) and A2 complements (*scared silly*–[SUBCAT SC_AP]). As with object control verbs, subject control predicates also determine whether the control relationship is a raising one ([SUBTYPE RAIS]) or an equi one ([SUBTYPE EQUI]).

The tree in (174d) is the one in which the verbs *promise* and *strike as* appear. These verbs are peculiar in that although they occur in an environment typical of object control verbs, they are in fact subject control verbs. The N2 object is not to be construed as the subject of the VP complement. The ID rules introducing *promise* (VP/SE_NP_INF) and *strike as* (VP/SR_NP_AP) are correctly bound by the propagation rule SUBJ_CONTROL5 and not by the OBJ_CONTROL one.

A few more control ID rules, VP/SR_NP, VP/SR_NP_PHR, VP/SE_NP, VP/SE_NP_PHR, VP/OR_NP and VP/OE_NP generate examples such as *seem a fool*, *turn out a fool*, *look a fool*, *end up a fool*, *consider Lee a hero* and *make Kim a hero*. These are clearly a kind of control environment but syntactically there is no agreement process since nominal categories are not specified with an AGR feature.

6.5.2 The Semantics of Control

The syntactic aspects of control constructions are mirrored by the semantic translations assigned to them. Here, the distinction between raising predicates and equi ones is crucial. The controlled complement has a translation as a proposition which still needs a subject argument. In the case of equi predicates, one of their other arguments plays two roles: firstly as an argument of the equi predicate and secondly as the missing subject argument of the embedded proposition. To give an example, the VP complement of *try* in *Kim tries to help often* has the translation in (175).

```
(175)  "to help often"
       (lambda (Q1) (Q1
           (lambda (e1) (lambda (x1) (HELP e1 x1)))
           (lambda (e2) e2)
           (lambda (qu1) often)))
```

The first argument of Q1, `(lambda (e1) (lambda (x1) (HELP e1 x1)))`, is the predicate argument structure associated with the VP and it is still in need of information both about its event argument (`(lambda (e1)..`) and its subject argument (`(lambda (x1) ..`). The rule which combines this VP with a subject equi verb, VP/SE_INF, makes sure that both the matrix VP and the embedded VP will receive the interpretation of the matrix subject as their subject argument. At the same time it puts together the tense and aspect and quantifier information about the VP (which up until now has been kept separate as the second and third arguments of Q1) in order to provide a complete proposition interpretation as an argument to the equi predicate.[9] The rule is given in (176), the translation of the VP *tries to help often* in (177) and the final translation of *Kim tries to help often* in (178).

```
(176)  VP/SE_INF : ; try to dance (subject equi)
          VP --> H[SUBCAT SC_INF, SUBTYPE EQUI], VP[TO] :
          2 = [SLASH NOSLASH],
             (lambda (Q) (Q
                 (lambda (e) (lambda (x)
                    (1 e x (2 (lambda (prop)
                                (lambda (ta) (lambda (equa)
                                  (prop (uqe ((equa some) (e1)
                                     (NOTENSE (ta e1)))) x))))))))
                 (lambda (e2) e2)
                 (lambda (qu) qu))) :
          2 = [SLASH X2],
             (lambda (Q) (Q
                 (lambda (e) (lambda (x) (lambda (wh)
                    (1 e x (2 (lambda (prop)
                                (lambda (ta) (lambda (equa)
                                  (prop (uqe ((equa some) (e1)
                                     (NOTENSE (ta e1)))) x wh)))))))))
                 (lambda (e2) e2)
                 (lambda (qu) qu))).
```

```
(177)  "tries to help often"
       (lambda (Q1) (Q1
            (lambda (e1) (lambda (x1)
               (TRY e1 x1 (HELP (uqe (often (e2) (NOTENSE e2))) x1))))
            (lambda (e3) e3)
            (lambda (qu1) qu1)))
```

[9]Our treatment differs from a property theoretic approach (Chierchia 1985, Chierchia & Turner 1988) which provides a property as an argument to the control predicate rather than a complete proposition, i.e. it does not resolve the control relation at the level of compositional interpretation. See Dowty (1985) for an evaluation of the differences between the two approaches.

(178)
```
"kim tries to help often"
(DECL
 (TRY (uqe (some (e1) (PRES e1)))
      #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
      (HELP (uqe (often (e2) (NOTENSE e2))) #1#)))
```

The analysis of raising predicates is very similar to the analysis of equi ones except that the constituent from the matrix clause which provides the subject information for the controlled complement's semantics does not actually play a role with respect to the matrix clause semantics. The subject raising example, *Kim seems to help often*, has an analysis which is almost the same as the equi example, *Kim tries to help often*, just described. The translation for the controlled complement *to help often* is as given in (175), the rule in (179) is the one which builds the VP *seems to help often* and the translations for this and the complete sentence are given in (180).

(179)
```
VP/SR_INF : ; appear to be crazy (subject raising).
   VP --> H[SUBCAT SC_INF, SUBTYPE RAIS], VP[TO] :
   2 = [SLASH NOSLASH],
      (lambda (Q) (Q
          (lambda (e) (lambda (x)
             (1 e (2 (lambda (prop)
                       (lambda (ta) (lambda (equa)
                         (prop (uqe ((equa some) (e1)
                            (NOTENSE (ta e1)))) x))))))))
          (lambda (e2) e2)
          (lambda (qu) qu))) :
   2 = [SLASH X2],
      (lambda (Q) (Q
          (lambda (e) (lambda (x) (lambda (wh)
             (1 e (2 (lambda (prop)
                       (lambda (ta) (lambda (equa)
                         (prop (uqe ((equa some) (e1)
                            (NOTENSE (ta e1)))) x wh)))))))))
          (lambda (e2) e2)
          (lambda (qu) qu))).
```

(180)
```
"seems to help often"
(lambda (Q1) (Q1
     (lambda (e1) (lambda (x1)
        (SEEM e1 (HELP (uqe (often (e2) (NOTENSE e2))) x1))))
     (lambda (e3) e3)
     (lambda (qu1) qu1)))
```

(181)
```
"kim seems to help often"
(DECL
 (SEEMS (uqe (some (e1) (PRES e1)))
        (HELP (uqe (often (e2) (NOTENSE e2)))
              (name (the (x1) (and (sg x1)
                    (named x1 KIM) (animate x1)))))))
```

If we compare these examples with the ones for *try*, we see that the predicate SEEM has one argument fewer than TRY—its subject is a semantic argument of the controlled complement but plays no role with respect to SEEM itself.

Object control predicates have a similar semantics to subject control ones except that it is an object of the matrix predicate which is interpreted as the subject of the controlled complement. The equi/raising distinction is treated in the same way: object equi predicates have one more argument than corresponding raising ones because their object plays a role both in the matrix clause and the controlled complement while the objects of object raising predicates only participate in the semantics of the controlled complement. The examples in (182) demonstrate the analysis of object equi and object raising predicates respectively.

(182)
```
"kim tells lee to help"
(DECL
 (TELL (uqe (some (e1) (PRES e1)))
       (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
       #1=(name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
       (HELP (uqe (some (e2) (NOTENSE e2))) #1#)))

"kim believes lee to help"
(DECL
 (BELIEVE (uqe (some (e1) (PRES e1)))
          (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
          (HELP (uqe (some (e2) (NOTENSE e2)))
                (name (the (x2) (and (sg x2)
                      (named x2 LEE) (animate x2)))))))
```

The auxiliary and modal rules behave syntactically as if they are control verbs. In the case of the tense/aspect auxiliaries the relationship is like a raising relationship since the auxiliary acquires the type of subject which its complement needs (*it had been raining*, *there is a book on the table*, *that Lee won that race will annoy Lee* etc.) Semantically, however, these auxiliaries are not like raising verbs because they do not contribute a predicate of their own to the translation—they only contribute tense and aspect information:

(183)
```
"it had been raining"
(DECL
  (RAIN (uqe (some (e1) (PAST (PERF (PROG e1)))))))
```

Modal verbs tend to be ambiguous between an epistemic meaning (*this must be Glasgow*) and a deontic meaning (*he must behave himself*). When they have an epistemic reading they behave syntactically like raising verbs in that they occur with whichever subject their complement needs: *it must be raining*, *that Kim won the race must annoy you.* Like the tense/aspect auxiliaries, however, they do not behave like raising predicates semantically because they do not contribute a predicate to the translation —instead they contribute a modal operator scoping over the whole proposition (184a). When modals have a deontic reading they behave syntactically like equi verbs since they can only take an N2[NFORM NORM] subject—examples with non-normal subjects only ever have an epistemic reading and never a deontic one. Semantically we treat the deontic interpretations in the same way as equi predicates: the modal contributes a predicate to the translation and the subject plays a role with respect both to the modal and to the complement predicate (184b).

(184) "he may stand up"

```
a.  (DECL (POS ((CP STAND UP)
                 (uqe (some (e1) (PRES e1)))
                 (pro (the (x1) (and (sg x1) (male x1)))))))

b.  (DECL (MAY (uqe (some (e1) (PRES e1)))
               #1=(pro (the (x1) (and (sg x1) (male x1))))
               ((CP STAND UP)
                  (uqe (some (e2) (NOTENSE e2)))
                  #1#)))
```

We keep only one lexical entry for modal verbs and therefore achieve only one parse of a sentence involving a modal. The semantic translation, however, is organised so that two readings will result from the parse. In order to get two translations, the entries for ambiguous modals have a semantics which keeps both the epistemic operator and the deontic predicate available: the semantics for the entry for *must* is `(lambda (int) (int NEC MUST))`. The modal rules, VP/MODAL1a–VP/MODAL2, give two interpretations for VPs which are [AGR N2[NFORM NORM]] using, in one, the epistemic operator and, in the other, the deontic predicate. When AGR is instantiated in other ways only the epistemic translation is given. Apart from this we are unable to disambiguate between the two readings.

6.6 Negation

The grammar deals with forms of negation involving the presence in a sentence of the word *not*, either free-standing or cliticized to an auxiliary or modal (e.g. *can't*, *doesn't*). The syntax of cliticized *not* is quite simple: the contracted forms, which are all finite, have lexical entries marked as [NEG +] and are able to be picked up by the normal auxiliary and modal rules. In general, wherever a non-negative finite [AUX +] verb can occur, so can a negative one (*Kim can swim*, *Kim can't swim*). The distribution of free-standing *not* is controlled by means of both a metarule and some ID rules.

(185)
```
NEGATION : ; takes an aux or modal rule and returns a new
           ; version with a "not" between head and complement.
           ; The semantics of the new rules are equivalent to
           ; the [NEG +] semantics on the input rules.
   V2[+AUX, FIN (@, +), PRO -, ~TAG] --> H[NEG @a, CONEG @b], W.
   ==> V2[+AUX, +FIN] --> H[+FIN, NEG -, CONEG @c], [NEG +], W :
   1 = [NEG +, ~PAST], (lambda (s) s) :
   1 = [NEG +, PAST -], 4 = [PAST -], (lambda (s) s) :
   1 = [NEG +, PAST +], 4 = [PAST +], (lambda (s) s) :
   1 = [NEG +, PAST FUT], 4 = [PAST FUT], (lambda (s) s).
```

The metarule NEGATION, as shown in (185), takes [AUX +] VP rules and derives a new set of rules from them which are the same except that a *not* occurs between the head and the VP complement and the new rules are restricted to be finite. This permits examples such as *has not eaten*, *could not help it* etc. The metarule accounts for a good many occurrences of *not* but does not cover all possibilities. For example, it does not cover cases such as *may have not eaten* since the *not* occurs with a non-finite auxiliary. Most other cases are accounted for by the rule VP/NEG as shown in (186).

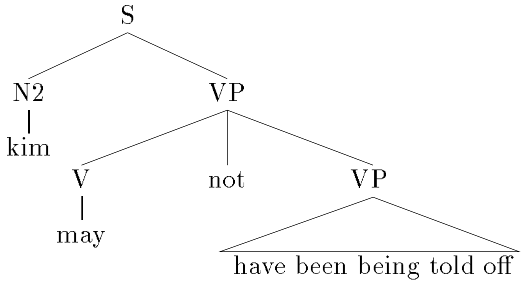
(186)
```
VP/NEG : ; (does) not dance. -FIN prevents 'not dances'
   VP[NEG +] --> [NEG +], H2[SUBJ -, FIN -, NEG -] :
   2 = [SLASH NOSLASH],
      (2 (lambda (prop) (lambda (ta) (lambda (equa)
           (lambda (Q) (Q (lambda (e) (lambda (x)
             (NOT (prop e x)))) ta equa)))))) :
   2 = [SLASH X2],
      (2 (lambda (prop) (lambda (ta) (lambda (equa)
           (lambda (Q) (Q (lambda (e) (lambda (x) (lambda (wh)
             (NOT (prop e x wh))))) ta equa)))))).
```

This rule allows *not* to precede any non-finite VP and therefore permits all of the following:

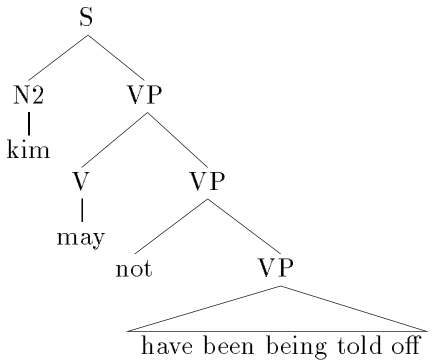
(187) a Kim may not have been being told off.
 b Kim may have not been being told off.
 c Kim may have been not being told off.
 d Kim may have been being not told off.

The careful reader will have noticed that (187a) can also be generated as a result of the metarule in (185): in fact (187a) does get two parses which receive different semantic interpretations (see below). The two structures are:

(188)



(189)



Examples involving finite tense/aspect auxiliaries do not so easily give rise to an ambiguity as ones with modals and so we block the possibility of a parse involving the VP/NEG rule in such cases. To do this we use a slightly complex interaction of the two features NEG and

CONEG (as described in section 3.1.1.1) which results in us blocking the use of VP/NEG when the auxiliary is [NEG –] (*Kim does not leave*) but permitting it when the auxiliary is [NEG +] (*Kim doesn't not leave*), since the NEGATION metarule will not handle this case.

There are one or two places where *not* can appear in constituents other than VP and for these we have provided specific rules. ID rule A2/NOT permits *not* before a quantifier (*not many*, *not often*); PS rule NEG/DETA allows *not* before a degree specifier (*not too*, *not so*); ID rules N2+/NEG, A2/NEG and P2/NEG permit *not* to occur inside coordinations of N2, A2 and P2 (*a cat but not a dog*, *happy and not tired*, *under the table and not on it*); PS rule P/NEG allows *not* in coordinations of P (*under and not on the table*); ID rule VP/PRO/NEG allows *not* to occur after a [PRO +] verb (*but he did not*).

Semantically, negation is usually treated by means of the propositional operator NOT:

(190)
```
"the abbot didn't abdicate"
(DECL
 (NOT
  (ABDICATE (uqe (some (e1) (PAST e1)))
   (dd (the (x1) (and (sg x1) (ABBOT x1)))))))

"lee was not helping"
(DECL
 (NOT
  (HELP (uqe (some (e1) (PAST (PROG e1))))
   (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1)))))))
```

In a few cases the question arises whether the operator NOT should scope over a higher proposition or an embedded one, or whether it should scope over another propositional operator or not. In the case of modals, two parses are produced, one by means of the NEGATION metarule and one by means of ID rule VP/NEG (as described above), and each parse corresponds to a different scoping. Since modals are already ambiguous between epistemic and deontic readings, four interpretations usually result from a sentence with a modal and non-cliticised negation:

(191)
```
"kim must not be helping the abbot"
(DECL
 (NOT
  (NEC
   (HELP (uqe (some (e1) (PRES (PROG e1))))
    (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
    (dd (the (x2) (and (sg x2) (ABBOT x2))))))))

(DECL
 (NOT
  (MUST (uqe (some (e1) (PRES e1)))
   #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (HELP (uqe (some (e2) (NOTENSE (PROG e2)))) #1#
    (dd (the (x2) (and (sg x2) (ABBOT x2))))))))
```

```
(DECL
 (NEC
  (NOT
   (HELP (uqe (some (e1) (PRES (PROG e1))))
    (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
    (dd (the (x2) (and (sg x2) (ABBOT x2))))))))

(DECL
 (MUST (uqe (some (e1) (PRES e1)))
  #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (NOT
   (HELP (uqe (some (e2) (NOTENSE (PROG e2)))) #1#
    (dd (the (x2) (and (sg x2) (ABBOT x2))))))))
```

Of course, in many cases only one or two of these interpretations are actual possibilities but we do not have access to sufficient information (such as whether an event is a state or an activity) to disambiguate at this point.

In a few cases the negative marker NOT is used not as a propositional operator but as part of predicate or a quantifier:

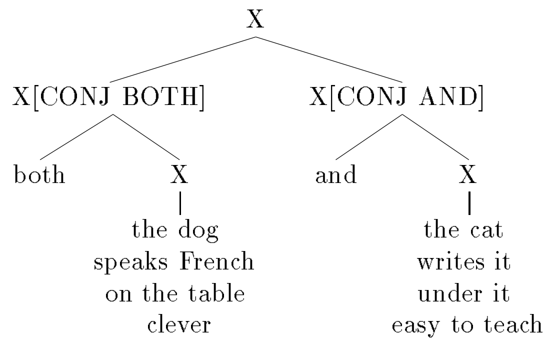
(192)
```
"not many abbeys fall down"
(DECL
 ((CP FALL DOWN) (uqe (some (e1) (PRES e1)))
  (uq ((NOT many) (x1) (and (pl x1) (ABBEY x1))))))
```

(193)
```
"kim is not very eager"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EAGER (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree (NOT very)))))

(DECL
 (NOT
  (BE (uqe (some (e1) (PRES e1)))
   (EAGER (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
    (degree very)))))
```

6.7 Coordination

Because our system uses fixed-arity term unification (instead of graph-theoretic unification with extension), no general elegant treatment of coordination, such as the GPSG coordination schemata, is available to us. Instead, coordination rules have to be stated for each of the major category types. All of the coordination rules induce broadly the same analysis, however, and a schematic example is given in (194).

(194)



Coordinators are treated as being in constituency with the conjuncts that follow them. Where there is no coordinator, as with the first conjunct in *the dog and the cat*, the conjunct has the feature specification [CONJ NULL]. No special rules are need for this: all major category constituents are marked as [CONJ NULL] to indicate that they can appear as a coordinator-less conjunct. The order in which conjuncts can occur is controlled by LP rule LP9 which allows the following possibilities: NULL...AND, NULL...OR, NULL...BUT, BOTH...AND, NEITHER...NOR, EITHER...OR.

Rules have been provided for coordinations of N2 (*the dog and the cat*), N1 (*the clever dogs and stupid cats*), N (*the clever dogs and cats*), S (*Lee sings and Kim dances*), VP (*reads books and understands them*), V (*reads and understands books*), P2 (*on the table and under it*), P (*on and under the table*), A2 (*exceptionally clever and clearly capable of learning*), A1 (*so clever but so arrogant*) and A (*too clever and arrogant*).

Because GPSG uses a more complex category matching technique than ours, it is able to treat all conjuncts as heads. This is not possible in our grammar, however, since it would entail that all conjuncts had the same values for their head features, and counterexamples to this are numerous. As a result, apart from coordinations of V and A, conjuncts are not treated as heads. Instead our analysis treats them as partial heads—for each type of coordination a subset of head features have been identified which must be shared between conjuncts. Various propagation rules ensure correct bindings for these features (see the propagation rules named COORD/... and the sets CONJ_NOMHEAD, CONJ_ADJHEAD etc.).

The rules for coordinations of nominal categories are rather complex since an attempt has been made to describe the propagation of number (the feature PLU) in these cases. For example, the grammar reflects the fact that a coordination of N2s with *and* will always be [PLU +], irrespective of the PLU values of the conjuncts, whilst the PLU value of a coordination with *or* is dependent on the PLU values of the conjuncts. A correct statement of person (PER) propagation, on the other hand, proved too difficult to implement so PER is left underspecified on the mother of a coordination.

While it is generally the case that only like categories can be coordinated, there are well known exceptions to this rule. There are particular positions where unlike categories may be coordinated. These positions are, firstly, after copula *be* where any X2[+PRD] category may be coordinated with any other X2[+PRD] (*Lee is in the bath and singing*, *Lee is a fool and unable to learn* etc.) and secondly, in modifier of V2 positions where any X2[+ADV] categories may be coordinated (*Lee sings tunelessly but with great gusto*, *Lee works at a fast rate but carelessly*). ID rules PRD/COORD1–PRD/COORD6 and CONJ/PRD allow for coordinations of X2[+PRD] categories, whilst ID rules MOD/COORD1–MOD/COORD6, CONJ/MOD1 and CONJ/MOD2 allow for coordinations of adverbial categories. The fact that these are coordinations of special X2 categories (as described in sections 3.1.1.5, 3.1.1.6 and 5.6) means that their distribution is limited to the positions outlined above.

Extractions out of coordinations may only occur in an 'across-the-board' fashion. That is, if one conjunct contains a gap, then so must all the other conjuncts (*who is it that John likes E but Bill hates E* vs **who is it that John likes E but Bill hates Mary*). Propagation rule PSLASH8 binds SLASH on the mother of a coordination to SLASH on all of the daughters thereby ensuring an across-the-board pattern of extraction. In addition, though conjuncts can contain gaps, they cannot *be* gaps: **who did you meet John and E*, **who did you meet E and E*. To reflect this, conjuncts are marked as [BEGAP –].

Semantically, coordination is usually simply reflected by semantic conjunction or disjunction. We use upper case `AND` and `OR` instead of the lower case `and` and `or` which occur elsewhere in our semantic representations, in order to preserve the information that this conjunction/disjunction is the result of syntactic coordination. Some examples of the semantic representations of coordinations of N2, A2 and P2 are given in (195).

(195)
```
"kim and the abbots arrived"
(DECL
 (ARRIVE (uqe (some (e1) (PAST e1)))
  (AND (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
       (dd (the (x2) (and (pl x2) (ABBOT x2)))))))

"kim put the butter either in the cupboard or in the fridge"
(DECL
 (PUT (uqe (some (e1) (PAST e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  #1=(dd (the (x2) (and (ms x2) (BUTTER x2))))
  (OR (IN #1# (dd (the (x3) (and (sg x3) (CUPBOARD x3)))))
      (IN #1# (dd (the (x4) (and (sg x4) (FRIDGE x4))))))))

"a friendly and very helpful person"
(uq
 (some (x1)
  (and (sg x1)
   (and (PERSON x1)
        (AND (FRIENDLY x1 (degree unknown))
             (HELPFUL x1 (degree very)))))))
```

The method of building the semantic representations for coordinated VPs and Ss is quite complicated. When sentences are coordinated there must be a separate event argument for each conjunct, for example, *Kim left and Lee arrived* describes two events, the event of Kim leaving and the event of Lee arriving. To understand why the translation of coordinated S should be complex, consider how the translation of an S is usually built:

(196)
```
"kim left" --- S node
(lambda (Q1) (Q1
  (lambda (e1) (LEAVE e1
      (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))))
  (lambda (e2) (PAST e2))
  (lambda (qu1) qu1)))

"kim left" --- TOP node
(DECL
 (LEAVE (uqe (some (e1) (PAST e1)))
        (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))))
```

(197)
```
T1 : ; root symbol for the parser. It is here that the information
     ; that root sentences are always finite is encoded. This
     ; means that the parser gives two parses for finite sentences
     ; (one as T, one as S) whilst only one parse for non-finite
     ; sentences (as S)
    [T S] --> S[H +, COMP NORM, +FIN] :
    1 = [VFORM NOT, INV -], (DECL
          (1 (lambda (prop) (lambda (ta) (lambda (equa)
                        (prop (uqe ((equa some) (e) (ta e)))))))))) :
    1 = [VFORM NOT, INV +],
       (YNQU (1 (lambda (prop) (lambda (ta) (lambda (equa)
                        (prop (uqe ((equa some) (e) (ta e)))))))))) :
    1 = [VFORM BSE],
       (IMP (1 (lambda (prop) (lambda (ta) (lambda (equa)
                        (prop (uqe ((equa some) (e) (ta e)))))))))).
```

The translation of an S node is one where the parts of the proposition it denotes are present but where they have not yet been put together to form the completed proposition. The reason for not putting them together is that further information from S modifiers may need to be incorporated (e.g. *(angrily S(Kim left))*) and so the complete proposition is only put together when it is certain that the S is syntactically complete, i.e. when it occurs as a daughter of some other constituent. It occurs, for example, as the daughter of the TOP node category and the rule that builds the translation for this (reproduced in (197)) takes the parts and puts them together in the appropriate way. The rule takes the tense and aspect information (`(lambda (e2) (PAST e2))` and `(lambda (qu1) qu1)`, the second and third arguments of `Q`) and builds an event argument out of them which it then gives to the predicate-argument structure (`(lambda (e1) (LEAVE e1 (pro (the (x1) (and (sg x1) (male x1))))))`, the first argument of `Q`).

In the case of coordinated S, we want each conjunct to have a separate event argument, but the rules which put an S together only hand one event to the predicate argument structure. To bypass this problem we complete each S conjunct, giving each a distinct event argument, as part of the process of building the coordination. However, we have to return a translation for the whole which is of the form that rules taking S as a daughter expect. We therefore build a representation which has an already complete form embedded inside vacuous lambda abstractions. The example in (198) demonstrates:

(198)
```
"kim left"
(lambda (Q1) (Q1
    (lambda (e1) (LEAVE e1
        (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))))
    (lambda (e2) (PAST e2))
    (lambda (qu1) qu1)))

"and lee arrived"
(lambda (Q1) (Q1
  (lambda (e1) (ARRIVE e1
      (name (the (x1) (and (sg x1) (named x1 LEE) (animate x1))))))
  (lambda (e2) (PAST e2))
  (lambda (qu1) qu1)))


"kim left and lee arived" --- S node
(lambda (F1)
 (F1
  (lambda (lose1)
   (AND
    (LEAVE (uqe (some (e1) (PAST e1)))
     (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
    (ARRIVE (uqe (some (e2) (PAST e2)))
     (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))))
  lta lequa))


"kim left and lee arrived" --- TOP node
(DECL
 (AND
  (LEAVE (uqe (some (e1) (PAST e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
  (ARRIVE (uqe (some (e2) (PAST e2)))
   (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))))
```

We achieve a similar treatment for coordinated VPs whereby each conjunct has a separate event argument although they all have the same subject argument. A similar method has to be used as with S coordination, assigning a coordinate VP a translation which is already more complete than usual and which involves vacuous lambda abstraction at the top of the coordination.

The treatment of coordinated VP and S involves more complications than achieving separate event arguments however. There are many places where some item in a sentence containing an S or a VP coordination has be integrated into the translation of each conjunct even though it occurs only once outside the coordination. For coordinations of S, premodifiers such as *often* and *angrily* as in *often Lee helped Kim and Kim helped Lee*, *angrily Lee accused Kim and Kim left the room*, can be given syntactic scope over both conjuncts and need a corresponding semantic scope too. This involves adding the information provided by the modifier into the translation of each conjunct:

(199) "often kim helped lee and lee helped kim"

```
(DECL
 (AND
  (HELP (uqe (often (e1) (PAST e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))
  (HELP (uqe (often (e2) (PAST e2)))
   (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3))))
   (name (the (x4) (and (sg x4) (named x4 KIM) (animate x4)))))))
```

In the case of VPs there are many more items which scope over both conjuncts and whose translations have to be integrated with each conjunct. With sequences of auxiliaries, for example, a coordination may happen at the main verb level but tense and aspect information from higher up scopes over both conjuncts and has to be incorporated into each proposition that translates the conjuncts. For example, in *Kim will have been reading that book and thinking about it*, tense information from *will* and aspect information from *have* and *been* occurs outside of the coordination but needs to be integrated into the translation of the conjuncts, as in (200).

(200) "kim will have been reading that book and thinking about it"

```
(DECL
 (AND
  (READ
     (uqe (some (e1) (FUT (PERF (PROG e1)))))
     #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
     (dd (the (x2) (and (sg x2) (distal x2) (BOOK x2)))))
  ((CP THINK ABOUT)
     (uqe (some (e2) (FUT (PERF (PROG e2)))))
     #1#
     (pro (the (x3) (and (or (sg x3) (ms x3)) (nonhuman x3)))))))
```

In other cases, quantifiers and modifiers scope over coordinations of VPs and have to be integrated into all conjuncts:

(201) "kim frequently cooks and washes up"

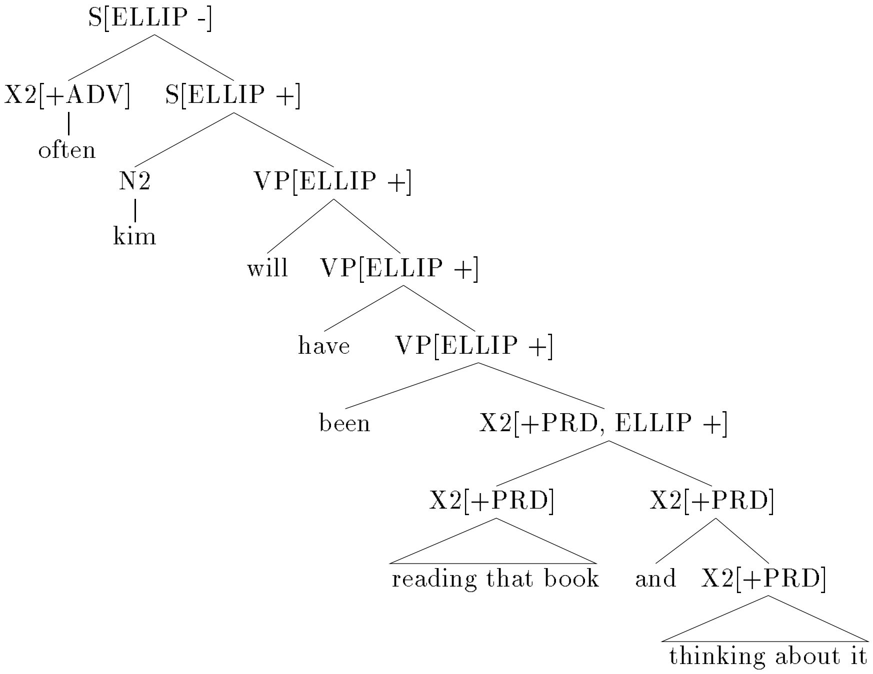
```
(DECL (AND
  (COOK (uqe (frequently (e1) (PRES e1)))
   #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
  ((CP WASH UP) (uqe (seldom (e2) (PRES e2))) #1#)))
```

"kim sang and danced beautifully"

```
(DECL (AND
  (and
   (SING #1=(uqe (some (e1) (PAST e1)))
    #2=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
   (#3=(BEAUTIFULLY) #1#))
  (and
   (DANCE #4=(uqe (some (e2) (PAST e2))) #2#)
   (#3# #4#))))
```

What we do in the semantics, in effect, is reconstruct the full sentences that an ellipsis or deletion account of coordination would claim underlies the VP coordination.[10] We achieve this reconstruction at the cost of complicating still further a part of the grammar which is already quite complex enough. We use a feature ELLIP to indicate whether a constituent is one which is looking for information which it needs to complete itself. For example, in the sentence *often Kim will have been reading that book and thinking about it*, ELLIP distributes as follows:

(202)



We have separate rules for [ELLIP +] constituents and [ELLIP −] ones. Rules whose name starts with 'ELL/' are ones for building and translating [ELLIP +] constituents. The translations are done using complex lambda expressions which look ouside of the constituent for information needed to complete it. There is also a metarule callked ELL/SAI which achieves subject-auxiliary inversion for the [ELLIP +] auxiliary rules ELL/VP/DO1, ELL/VP/HAVE, ELL/VP/WILL, ELL/VP/BE_PRD. The result is a rather complex set of rules which achieve the desired translation in most cases but which are sometimes unreliable. We have not by any means addressed all of the issues which arise in the semantics of coordination.

[10]We are aware that there are situations where it would be more desirable to think of a coordination of VPs as representing just one event, not two (or more) but we have chosen to implement a 'reconstruct the ellipsis' account since this is what is needed in a large number of cases and since it is more difficult to achieve than the alternative. There is no reason why the alternative should not be added to the grammar as well with the result that VP coordinations would always be ambiguous. Finding disambiguation proceedures, however, might not be easy.

6.8 Comparatives

6.8.1 Syntax

Our analysis of comparatives is loosely based on an account in Gazdar (1980). It is not exhaustive, but it recognises all of the following:

Nominal comparatives:

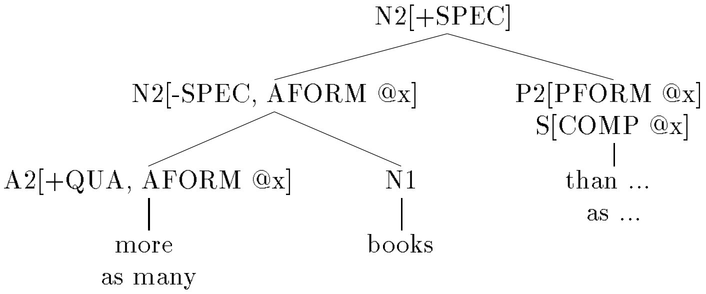
(203)	a	Kim reads more books than Lee	N2+/COMPAR1
	b	Kim reads as many books as Lee	
(204)	a	Kim reads more books than Lee reads	N2+/COMPAR3
	b	Kim reads as many books as Lee reads	
(205)	a	Kim reads more books than you'd think/than Lee does	N2+/COMPAR4
	b	Kim reads as many books as he claims/as Lee does	
(206)	a	Kim reads more books than magazines	N2+/COMPAR2
	b	as many children as grown-ups read this	
(207)	a	Kim reads more books than Lee reads magazines	N2+/COMPAR5
	b	Kim reads as many books as Lee reads magazines	

Adjectival comparatives:

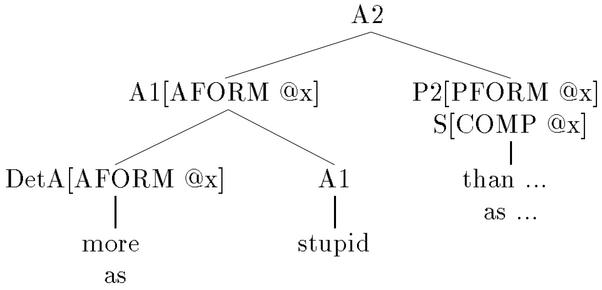
(208)	a	Kim is taller/more stupid than Lee	A2/COMPAR1
	b	Kim is as tall as Lee	
(209)	a	Kim is taller than Lee is	A2/COMPAR3a
	b	Kim is as tall as Lee is	
(210)	a	Kim plays more aggressively than Lee plays	A2/COMPAR3b
	b	Kim plays as aggressively as Lee plays	
(211)	a	Kim is more aggressive than I thought	A2/COMPAR4a
	b	Lee played as aggressively as I expected	
(212)	a	Kim plays more often than Lee does	A2/COMPAR4b
	b	Kim ate as quickly as he could	
(213)	a	Kim is more stupid than wicked	A2/COMPAR2
	b	Kim is as much stupid as wicked	
(214)	a	Kim is more stupid than Lee is wicked	A2/COMPAR5
	b	Kim is as stupid as Lee is wicked	

The (a) examples are comparatives whilst the (b) ones are equatives. Since the latter receive the same treatment as the former the illustrations will refer mostly to comparatives. *Than* and *as* have entries both as prepositions and as complementisers. The preposition occurs in (203), (206), (208) and (213) and the complementiser in all the others. In all the nominal examples, the basic structure assigned is one where an N2[+SPEC] rewrites as an N2[−SPEC] followed by the *than/as* constituent. The value for AFORM on the N2[−SPEC] (which it inherits from its quantifier) is bound to the value for PFORM or COMP on the other daughter to ensure that *than* is paired with *more* or *+er* and *as* is paired with *as*. Similarly, in the adjectival rules, an A2 dominates an A1 head and a *than/as* constituent. AFORM on the A2 is bound to PFORM or COMP on the other daughter. The basic structures are therefore as follows:

(215)



(216)



In all cases, the *than/as* constituent can be thought of as semantically elliptical (see below) and in some of the rules the use of SLASH reflects this. The range of possible *than/as* constituents are as follows:

(217)	a	more books than Lee	P2
	b	more books than Lee reads	S[SLASH N2]
	c	more books than you'd think	S[SLASH S]
	d	more books than Lee does	S[SLASH VP]
	e	more books than magazines	P2[SLASH +QUA]
	f	more books than Lee reads magazines	S[SLASH +QUA]

(218)	a	more stupid than Lee	P2
	b	taller than Lee is	S[SLASH A2]
	c	more aggressively than Lee plays	S
	d	more aggressive than I thought	S[SLASH S]
	e	more often than Lee does	S[SLASH VP]
	f	more stupid than wicked	A2[SLASH +QUA]
	g	more stupid than Lee is wicked	S[SLASH +QUA]

The P2 in (217a) and (218a) is a standard non-predicative P2. The S[SLASH N2] and S[SLASH A2] in (217b) and (218b) are standard gappy constituents, containing an N2 gap and an A2 gap respectively (*than Kim reads E*, *than Kim is E*). The S in (218c) is an unslashed version of (218b) specially provided for when the A2 is a modifier rather than a complement (we do not have modifier gaps). The S[SLASH S] and S[SLASH VP] in (217c), (217d), (218d) and (218e) are instances of a category that only ever appears in comparative constructions. There are no gaps corresponding to [SLASH VP] and [SLASH S]—instead [SLASH VP] is terminated by the rule in (219) which rewrites a VP[SLASH

VP] as a [PRO +] auxiliary. [SLASH S] is terminated by means of the special metarules STM3a and STM3b which generate new versions of rules with finite S complements which lack the S complement. STM3a and STM3b are shown in (220).

(219)
```
VP/PRO/SLASH : ; to terminate the [SLASH V2] introduced by
               ; N2+/COMPAR4 and A2/COMPAR4 -- lee helps more
               ; abbots than kim did/will/can etc. SUBCAT @s,
               ; SUBTYPE DEO allows both special modal entries
               ; (SUBCAT MODAL_BSE, SUBTYPE DEO) and ordinary
               ; non-modal +PRO auxs (SUBCAT NULL, SUBTYPE @) to
               ; match - semantics differs according to which.
   VP[+AUX, AGR N2[NFORM NORM], SLASH VP] -->
   H[SUBCAT @s, SUBTYPE DEO, PRO +] :
   1 = [SUBCAT NULL], (lambda (Q)
         (Q (lambda (e) (lambda (x) (lambda (wh) (wh e x))))
            (lambda (e2) e2) (lambda (qu) qu))) :
   1 = [SUBCAT MODAL_BSE],
      (lambda (Q) (Q (lambda (e) (lambda (x) (lambda (wh)
                     (1 e x (wh (uqe (some (e1) NOTENSE e1)) x)))))
            (lambda (e2) e2) (lambda (qu) qu))).
```

(220)
```
STM3a : ; terminates the [SLASH S] introduced by N2+/COMPAR4 and
        ; A2/COMPAR4a. Doesn't leave a gap - it just removes the
        ; sentential complement.
   [SLASH @, AGR N2[NFORM NORM]] --> [H +, BAR 0],
   S[+FIN, SLASH @, COMP (@, THAT)], W.
   ==> [SLASH S] --> [H +, BAR 0], W :
   2 = [SLASH X2], (lambda (s) (s
            (lambda (prop) (lambda (ta) (lambda (equa)
                     (lambda (Q) (Q
                           ((lambda (2) prop) (lambda (f) (f
                                    (lambda (lose1)
                                       (lambda (lose2) wh)) NIL
                                    NIL))) ta equa))))))).

STM3b : ; same as STM3b except inputs are outputs of previous
        ; MSLASH metarules and are [SLASH X2] rather than [SLASH
        ; @].
   [SLASH X2, AGR N2[NFORM NORM]] --> [H +, BAR 0],
   S[+FIN, SLASH X2], W. ==> [SLASH S] --> [H +, BAR 0], W :
   (lambda (s)
         (s (lambda (prop) (lambda (ta) (lambda (equa) (lambda (Q)
                        (Q
                           ((lambda (2) prop) (lambda (f) (f (lambda
                                       (lose1) (lambda (lose2) wh))
                                    NIL NIL))) ta equa))))))).
```

The P2[SLASH +QUA], S[SLASH +QUA] and A2[SLASH +QUA] constituents in (217e), (217f), (218f) and (218g) are also special categories which only appear with comparatives. The [SLASH +QUA] is terminated, without a gap, by means of the rules in (221).

(221)
```
N2+/COMPAR6 : ; rather than use a metarule, this terminates the
              ; [SLASH [QUA +]] introduced by idrules N2+/COMPAR2
              ; and N2+/COMPAR5. Notice that no empty node is
              ; introduced. The binding of values for PLU and
              ; COUNT on the daughter make sure that only plural
              ; (PLU +, COUNT +) or mass (PLU -, COUNT -) nouns
              ; will match (*kim eats more apples than orange).
   N2[SLASH [QUA +, ADV -, PRD -], +SPEC, WH NO, UB NO, EVER NO] -->
   H1[SLASH NOSLASH, PRO -, PLU @pc, COUNT @pc] :
   1 = [PLU +], (lambda (x) (and (pl x) (1 x))) :
   1 = [PLU -], (lambda (x) (and (ms x) (1 x))).

A2/COMPAR6 : ; rather than use a metarule, this terminates the
             ; SLASH +QUA introduced by idrules A2/COMPAR5 and
             ; A2/COMPAR2. Notice that no empty node is
             ; introduced.
   A2[SLASH [QUA +, ADV -, PRD -], WH NO, UB NO, EVER NO, QUA -] -->
   H1[AFORM NONE, SLASH NOSLASH, GRADE +] :
   1 = [QUA -, ADV -], (lambda (x)
         (lambda (wh) (1 (lambda (dword) (degree wh)) x))) :
   1 = [QUA -, ADV +], (lambda (x) (lambda (wh) (1 x wh))) :
   1 = [QUA +], (lambda (x) (lambda (wh) (1 x wh))).
```

6.8.2 Semantics of Nominal Comparatives

We produce semantic representations for nominal comparatives which are very similar to those described in Pulman (1991). The translation of the *than/as* phrase forms a complex quantifier with the *more/as*, as illustrated in (222).

(222)
```
"kim eats more apples than lee eats"
(DECL (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq (((more than)
     (lambda (c1) (and (and (pl c1) (APPLE c1))
       (EAT (uqe (some (e2) (PRES e2)))
        (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))))
    (x2) (and (pl x2) (APPLE x2))))))

"kim eats more apples than lee does"
(DECL (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq (((more than)
     (lambda (c1) (and (and (pl c1) (APPLE c1))
       (PROVP (uqe (some (e2) (PRES e2)))
        (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))))
    (x2) (and (pl x2) (APPLE x2))))))
```

```
"kim eats more apples than lee"
(DECL (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq (((more than)
     (lambda (c1) (and (and (pl c1) (APPLE c1))
       (PROPRED (uqe (some (e2) (NOTENSE e2)))
        (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))))
    (x2) (and (pl x2) (APPLE x2))))))
```

The three examples in (222) are produced by means of different rules (N2+/COMPAR3, N2+/COMPAR4 and N2+/COMPAR1 respectively). The translation of the first example, *Kim eats more apples than Lee eats*, is the most complete representation of the three. It contains the complex quantifer

```
((more than) (lambda (c1) (and (and (pl c1) (APPLE c1))
(EAT (uqe (some (e2) (PRES e2)))
(name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))))
```

where `(more than)` takes as an argument the set described by

```
(lambda (c1) (and (and (pl c1) (APPLE c1))
(EAT (uqe (some (e2) (PRES e2)))
(name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))).
```

This set is the set of apples such that Lee ate them. The interpretation of the `((more than) ..)` quantifier can therefore be glossed as 'a quantity which is greater than the size of the set `(lambda (c1)...)`'. Although the second and third examples in (222) mean much the same as the first, they do not explicitly contain a predicate in the *than/as* phrase since it has proved too complex to try to inherit this information from the main clause as part of the process of translation. Our alternative is to provide a 'pro' predicate whose antecedent must be found by a resolution component. In the case of the third example, the only thing we know is that there is a predicate which has at least three arguments, *Lee*, an unknown quantity of apples and an event argument with unknown tense properties. We represent this with the atom PROPRED. In the case of the second example, we know a little more—we know that *Lee* is the subject of the missing predicate and we know its tense. Continuing our usual treatment of VP ellipsis we represent this with PROVP.

(223) illustrates our treatment of examples which involve terminating a [SLASH S] by removing the finite S complement of a verb. Again we use PROPRED but here it represents the missing complement of THINK. All that we can infer about this predicate is that one of its argument is *apples* and one is an event argument (with no tense information).

(223)

```
"kim ate more apples than he thought"
(DECL
 (EAT (uqe (some (e1) (PAST e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq
   (((more than)
     (lambda (c1)
      (and (and (pl c1) (APPLE c1))
       (THINK (uqe (some (e2) (PAST e2)))
        (pro (the (x3) (and (sg x3) (male x3))))
        (PROPRED (uqe (some (e3) (NOTENSE e3))) c1)))))
    (x2) (and (pl x2) (APPLE x2))))))
```

Nominal comparatives with a [SLASH +QUA] *than/as* phrase are ones where only a quantifier is missing. Our analysis currently gives the translation in (224) for *Kim eats more apples than Lee eats pears* but this is not the translation we would want to achieve. Ideally we would want a translation which paralleled the examples in (222) (as shown in (225)) but this would involve adding additional clauses to all the VP rules and this is too high a cost for these relatively infrequent examples. Post-processing of the semantic representations could achieve the same effect.

(224)
```
"kim eats more apples than lee eats pears"   (actual interpretation)
(DECL
 (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq
   (((more than)
     (lambda (c1)
      (EAT (uqe (some (e2) (PRES e2)))
       (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3))))
       (and (pl c1) (PEAR c1)))))
    (x2) (and (pl x2) (APPLE x2))))))
```

(225)
```
"kim eats more apples than lee eats pears"   (desired interpretation)
(DECL
 (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq
   (((more than)
     (lambda (c1)
      (and (and (pl c1) (PEAR c1))
       (EAT (uqe (some (e2) (PRES e2)))
        (name (the (x3) (and (sg x3) (named x3 LEE) (animate x3)))) c1))))
    (x2) (and (pl x2) (APPLE x2))))))
```

The other kind of [SLASH +QUA] nominal comparative, for example, *Kim eats more apples than pears* is the only one whose interpretation does not involve a proposition inside the quantifier. Here the quantifier simply compares quantities of two entities:

(226)
```
"kim eats more apples than pears"
(DECL
 (EAT (uqe (some (e1) (PRES e1)))
  (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
  (uq
   (((more than) (lambda (c1) (and (pl c1) (PEARS c1)))) (x2)
    (and (pl x2) (APPLES x2))))))

"more men than women drive"
(DECL
 (DRIVE (uqe (some (e1) (PRES e1)))
  (uq
   (((more than) (lambda (c1) (and (pl c1) (WOMAN c1)))) (x1)
    (and (pl x1) (MAN x1))))))
```

6.8.3 Semantics of Adjectival Comparatives

The representations which translate *than/as* phrases in nominal comparatives have a similar form in adjectival comparatives but while they form part of a complex quantifier in the nominal case, they form part of a complex degree modifier in the adjectival case.

(227)
```
"kim is more crazy than lee is"
(DECL (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree
    ((more than)
     (lambda (d1)
      (BE (uqe (some (e2) (PRES e2)))
       (CRAZY (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
        (degree d1)))))))))


"kim is more crazy than lee"
(DECL (BE (uqe (some (e1) (PRES e1)))
  (CRAZY (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree
    ((more than)
     (lambda (d1)
      (BE (uqe (some (e2) (NOTENSE e2)))
       (CRAZY (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
        (degree d1)))))))))
```

The two examples in (227) have almost exactly the same translation except that the former has tense information from the *is* that the latter does not have. Following von Stechow (1984), Rayner and Banks (1990) and many other authors we treat gradable adjectives as relations between a subject and a degree.[11] The comparative expression becomes part of the specification of the degree and involves a comparison of two degrees rather than two entities.

The other kinds of adjectival comparatives have the following translations:

(228)
```
"kim helps more eagerly than lee does"
(DECL (and
  (HELP #1=(uqe (some (e1) (PRES e1)))
   (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1)))))
  (#2=(EAGER LY) #1#
   (degree
    ((more than)
     (lambda (d1)
      (and
       (PROVP #3=(uqe (some (e2) (PRES e2)))
        (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2)))))
       (#2# #3# (degree d1)))))))))
```

[11]When there is no degree modifier we supply an indication of that fact: *crazy* translates as `(lambda (x) (CRAZY x (degree unknown)))`.

```
"kim is more crazy than I thought"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (CRAZY #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree
    ((more than)
     (lambda (d1)
      (THINK (uqe (some (e2) (PAST e2)))
       (pro (the (x2) (and (sg x2) (speaker x2))))
       (BE (uqe (some (e3) (NOTENSE e3))) (CRAZY #1# (degree d1))))))))))

"kim is more eager than crazy"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EAGER #1=(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree
    ((more than)
     (lambda (d1)
      (BE (uqe (some (e2) (NOTENSE e2))) (CRAZY #1# (degree d1)))))))))

"kim is more eager than lee is crazy"
(DECL
 (BE (uqe (some (e1) (PRES e1)))
  (EAGER (name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))
   (degree
    ((more than)
     (lambda (d1)
      (BE (uqe (some (e2) (PRES e2)))
       (CRAZY (name (the (x2) (and (sg x2) (named x2 LEE) (animate x2))))
        (degree d1)))))))))
```

7 Semantic Scoping and Post-Processing Module

The module consists of two parts. The first is a simple quantifier scoper based on Vestre (1991). The second, applying to the output of this, is a post-processor which carries out a set of simplifications. The scoper produces a single scoping of an SR, where the order of scoped quantifiers simply reflects their left to right order in the input. The only constraint is that a quantified event (marked with `uqe`) associated with a proposition or embedded proposition is assigned narrower scope than all the other quantifiers associated with that proposition.

Scoping is local to an embedded proposition in that the quantifier which is the translation of an embedded clause is not scoped outside of the predicate of the higher clause. For example, the SR assigned by the grammar to the sentence *an abbot knows an acquaintance helps* is:

```
(DECL
   (KNOW (uqe (some (e1) (PRES e1)))
      (uq (some (x1) (and (sg x1) (ABBOT x1))))
      (HELP (uqe (some (e2) (PRES e2)))
         (uq (some (x2) (and (sg x2) (ACQUAINTANCE x2)))))))
```

which receives the scoping:

```
(DECL
   (some (x1) (and (sg x1) (ABBOT x1))
      (some (e1) (PRES e1)
         (KNOW e1 x1
            (some (x2) (and (sg x2) (ACQUAINTANCE x2))
               (some (e2) (PRES e2) (HELP e2 x2)))))))
```

Representations of noun phrases deriving from quantifiers other than `the`, `some` or `every` have a definiteness specification `def` inserted when appropriate. The default is assumed to be indefinite. Likewise, representations of kind interpretations have a `generic` specification added.

7.1 Post-Processing

The post-processor takes a scoped SR and performs a set of simplifications on it. The simplifications applied are:

- A formula representing a named person is replaced by just the name itself, e.g. `(name (the (x1) (and (sg x1) (named x1 KIM) (animate x1))))` becomes `KIM`.

- A formula representing a number is replaced by the number itself, e.g. the representation of the string "three hundred and sixty five", `(NN (+ (* \3 \100) (+ \60 \5)))`, becomes `365`. A number formula can also be a range, e.g. `(NN (RANGE (NN \2) (NN \3)))`, which becomes `(RANGE 2 3)`.

- A non-atomic formula representing a complex predicate (typically a phrasal verb) is replaced by an appropriate atomic predicate name, e.g. `(CP BRING BACK)` becomes `BRING-BACK`.

- A non-atomic formula representing a compound is replaced by an appropriate atomic name, e.g. `(compound BUTTER MOUNTAIN)` becomes `BUTTER_MOUNTAIN`.

- A formula that is recognised as representing a possession relation is replaced by a `POSSESSED-BY` predicate, e.g. `(lambda (R1) (R1 x1 KIM))` becomes `(POSSESSED-BY x1 KIM)`.

- A formula produced from the scoping of a complex quantifer corresponding to a comparative or adjective of degree, such as "more abbots than ..."

```
(((more than)
    (lambda (c1)
       (and (pl c1) (ABBOT c1) (some (e2) ...))))
 (x2) (and (pl x2) (ABBOT x2)) ...)
```

  is replaced by

```
(((more than)
    (?quant (c1) (and (pl c1) (ABBOT c1)) (some (e2) ...)))
 (x2) (and (pl x2) (ABBOT x2)) ...)
```

- Nested `and`s are simplified.

8 Conclusions and Further Work

The resulting grammar has very wide syntactic and semantic coverage: it is arguably the widest coverage grammar which has been developed for English in a declarative unification-based formalism, though it lacks the depth of analysis of a system such as the Core Language Engine (Alshawi 1992). Nevertheless there are many inadequacies which remain; these include:

Punctuation and Tokenisation The grammar does not take account of punctuation, taking unpunctuated and tokenised text as input. Since many aspects of punctuation do not reflect syntactic distinctions, this is appropriate in general (e.g. Nunberg, 1990). What is required is a text preprocessor capable of tokenising input for lexical access and 'parcelling' units for syntactic and compositional semantic analysis. In this way problems of parentheticals and other sub-sentential constituents which are related via discourse relations, such as elaboration, and not by syntactic relations can be recognised and parsed individually. However, some limited aspects of punctuation, such as the occurrence of commas in coordination and after preposed constituents, need to be integrated with the grammatical description.

Ambiguity The grammar, particularly in conjunction with the large lexicon, assigns large numbers of analyses to many naturally-occurring sentences. A number of these analyses are the result of errors in the lexicon and can be ameliorated by production of a specialised lexicon for a specific application. However, in general some technique will always be required for selecting amongst alternative analyses. Briscoe & Carroll (1991, 1992) and Carroll & Briscoe (1992) describe a technique based on probabilistic selection of the syntactically most likely parse. However, in domain-dependent applications semantic filtering of the possible analyses may be feasible and preferable (e.g. Alshawi, 1992).

Semantic Framework The semantic approach of rule-to-rule translation into formulae of the lambda calculus results in verbose, multiple specifications of the semantics of individual rules and very complex formulae associated with sub-analyses. A more perspicuous semantics might be developed by constructing well-formed formulae using unification (Pollard & Sag 1987, Alshawi 1992).

Coverage Inadequacies of coverage are continuously emerging and most are straightforward to remedy by addition of a few simple rules. However areas in which the current framework is possibly inadequate but certainly inelegant, and where the current grammar is deficient include: coordination, adjectival and adverbial modification, multiple and echo wh-questions, gapping, sluicing, and complex ellipsis.

9 References

Alshawi, H., D. Carter, J. van Eijck, R. Moore, D. Moran, F. Pereira, A. Smith & S. Pulman (1988) *Interim Report on the SRI Core Language Engine,* CCSRC-005, SRI Cambridge Research Centre.

Alshawi, H. (ed) (1992) *The Core Language Engine,* MIT Press, Cambridge, MA.

Bach, E. (1986) 'The Algebra of Events', *Linguistics and Philosophy, vol.9,* pp. 5–16.

Barwise, J. & R. Cooper (1981) 'Generalized Quantifiers and Natural Language', *Linguistics and Philosophy, vol.4,* pp. 159–219.

Boguraev, B., Briscoe, E., Carroll, J., Carter, D. and Grover, C. (1987) 'The derivation of a grammatically indexed lexicon from the Longman Dictionary of Contemporary English', *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics,* Stanford, California, pp. 193–200.

Borsley, R. (1983) 'A Welsh Agreement Process and the Status of VP and S' in Gazdar, G., E. Klein & G. Pullum (eds.), *Order, Concord and Constituency,* Foris, Dordrecht.

Boguraev, B., J. Carroll, E. Briscoe & C. Grover (1988) 'Software Support for Practical Grammar Development', *Proceedings of the Proceedings of 12th International Conference of Computational Linguistics,* Budapest, Hungary, pp. 54–58.

Briscoe, E. & J. Carroll (1991) *Generalised Probabilistic LR Parsing of Natural Language (Corpora) Using Unification-based Grammars,* University of Cambridge, Computer Laboratory, TR-234.

Briscoe, E. & J. Carroll (1992, in press) 'Generalised Probabilistic LR Parsing of Natural Language (Corpora) Using Unification-based Grammars', *Computational Linguistics,*.

Briscoe, E., C. Grover, B. Boguraev & J. Carroll (1987a) 'Feature Defaults, Propagation and Reentrancy' in Klein, E. and van Benthem, J. (eds.), *Categories, Polymorphism and Unification,* Centre for Cognitive Science, University of Edinburgh, pp. 19–34.

Briscoe, E., C. Grover, B. Boguraev & J. Carroll (1987b) 'A Formalism and Environment for the Development of a Large Grammar of English', *Proceedings of the 10th International Joint Conference on Artificial Intelligence,* Milan, Italy, pp. 703–708.

Carroll, J. (1993, in prep.) *Parsing Techniques for Unification-based Grammars,* Doctoral Dissertation, Cambridge University, Computer Laboratory.

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

Carroll, J., E. Briscoe & C. Grover (1991) *A Development Environment for Large Natural Language Grammars,* Technical Report No. 233, Computer Laboratory, University of Cambridge.

Carroll, J. & C. Grover (1989) 'The Derivation of a Large Computational Lexicon for English from LDOCE' in Boguraev, B. & E. Briscoe (eds.), *Computational Lexicography for Natural Language Processing,* Longman, London.

Chierchia, G. (1985) 'Formal Semantics and the Grammar of Predication', *Linguistic Inquiry, vol.16,* pp. 417–443.

Chierchia, G. & R. Turner (1988) 'Semantics and Property Theory', *Linguistics and Philosophy, vol.11,* pp. 261–302.

Davidson, D. (1967) 'The Logical Form of Action Sentences' in N. Rescher (eds.), *The Logic of Decision and Action,* Univeristy of Pittsburgh Press, Pittsburgh.

Dowty, D. (1985) 'On recent analyses of the semantics of control', *Linguistics and Philosophy, vol.8,* pp. 291–331.

Dowty, D., R. Wall & S. Peters (1981) *Introduction to Montague Semantics,* Reidel, Dordrecht.

Gazdar, G. (1980) 'A Phrase Structure Syntax for Comparative Clauses' in Hoekstra, T., H. van der Hulst & M. Moortgat (eds.), *Lexical Grammar,* Foris, Dordrecht.

Gazdar, G., E. Klein, G. Pullum & I. Sag (1982) 'Coordinate structure and unbounded dependencies' in Barlow, Flickinger & Sag (eds.), *Developments in Generalized Phrase Structure Grammar,* Stanford Working Papers in Grammatical Theory: Indiana University Linguistics Club.

Gazdar, G., E. Klein, G. Pullum & I. Sag (1985) *Generalized Phrase Structure Grammar,* Blackwell, Oxford.

Grover, C., E. Briscoe, J. Carroll & B. Boguraev (1989) *The Alvey Natural Language Tools Project Grammar (Second Release): a Large Computational Grammar of English,* Technical Report No. 162, Computer Laboratory, University of Cambridge.

Hobbs, J. & S. Shieber (1987) 'An Algorithm for Generating Quantifier Scopings', *Computational Linguistics, vol.13,* pp. 47–63.

Horrocks, G. (1987) *Generative Grammar,* Longman, London.

Kilbury, J. (1986) 'Category Cooccurrence Restrictions and the Elimination of Metarules', *Proceedings of the Proceedings of 11th International Conference of Computational Linguistics,* Bonn, Germany, pp. 50–55.

Link, G. (1983) 'The Logical Analysis of Plurals and Mass Terms: A Lattice-Theoretical Approach' in Bäuerle, R., *et al.* (eds.), *Meaning, Use and Interpretation of Language,* De Gruyter, Berlin.

Link, G. (1987) 'Generalized Quantifiers and Plurals' in Gärdenfors, P. (eds.), *Generalized Quantifiers,* Reidel, Dordrecht.

Nunberg, G. (1990) *The Linguistics of Punctuation,* Univ. of Chicago Press, Chicago.

Parsons, T. (1990) *Events in the Semantics of English: a study in subatomic semantics,* MIT Press. Cambridge, MA.

Pereira, F. & S. Shieber (1987) *An Introduction to Natural Language Processing in Prolog,* University of Chicago Press.

Phillips, J. (1986) 'A Simple, Efficient Parser for Phrase-Structure Grammars', *SSAISB Quarterly Newsletter, vol.59,* pp. 14–18.

Phillips, J. & H. Thompson (1987) 'A Parser and an Appropriate Computational Representation for GPSG' in Klein, E. and Haddock, N. (eds.), *Cognitive Science Working Papers 1,* Centre for Cognitive Science, University of Edinburgh.

Pollard C, Sag I (1987) *Head-Driven Phrase Structure Grammar,* CSLI Lecture Notes 12, Stanford, California.

Pulman, S. (1991) 'Comparatives and Ellipsis', *Proceedings of the Proceedings of the 5th European ACL,* Berlin, Germany, pp. 2–7.

Pulman, S., G. Russell, G. Ritchie & A. Black (1988) 'Computational Morphology of English', *Linguistics, vol.26,* pp. 545–560.

Rayner, M. & A. Banks (1990) 'An Implementable Semantics for Comparative Constructions', *Computational Linguistics, vol.16,* pp. 86–112.

Ritchie, G., A. Black, S. Pulman & G. Russell (1987) *The Edinburgh/Cambridge Morphological Analyser and Dictionary System (Version 3.0) User Manual,* Software Paper No. 10, Department of Artificial Intelligence, University of Edinburgh.

Ritchie, G., G. Russell, A. Black & S. Pulman (1992) *Computational Morphology: Practical Mechanisms for the English Lexicon,* MIT Press, Cambridge, Mass..

Russell, G., S. Pulman, G. Ritchie & A. Black (1986) 'A Dictionary and Morphological Analyser for English', *Proceedings of the 11th International Conference on Computational Linguistics,* Bonn, Germany, pp. 277–279.

Sells, P. (1985) *Lectures on Contemporary Syntactic Theories,* CSLI, Stanford.

von Stechow, A. (1984) 'Comparing Semantic Theories of Comparison', *Journal of Semantics, vol.3,* pp. 1–77.

Vestre, E.J. (1991) 'An Algorithm for Generating Non-redundant Quantifier Scopings', *Proceedings of the Proceedings of the 5th European ACL,* Berlin, Germany, pp. 251–256.

Warner, A. (1984) 'The Structuring of English Auxiliaries: A Phrase Structure Grammar' in *York Papers in Linguistics,* Department of Language and Linguistic Science, University of York.