AIMS

Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung
Lehrstuhl für Theoretische Computerlinguistik Universität Stuttgart

AIMS • 2000 • VOL. 6 • NO. 1

YAP:
Parsing and Disambiguation With Feature–Based Grammars

Helmut Schmid

YAP:
Parsing and Disambiguation
With Feature-Based Grammars

Helmut Schmid

Januar 2000

Institut für maschinelle Sprachverarbeitung der Universität Stuttgart

Previous Issues of AIMS:

Vol. 1 (1) 1994: *Wigner Distribution in Speech Research.* Master thesis by Wolfgang Wokurek (in German), papers by Grzegorz Dogil, Wolfgang Wokurek, and Krysztof Marasek (in English), and a bibliography.

Vol. 2 (1) 1995: *Sprachentstörung.* Doctoral Dissertation. University of Vienna, 1994 by Wolfgang Wokurek (in German with abstract in English). Full title: Sprachentstörung unter Verwendung eines Lautklassendetektors (Speech enhancement using a sound class detector).

Vol. 2 (2) 1995: *Word Stress.* Master thesis by Stefan Rapp (in German) and papers mostly by Grzegorz Dogil, Michael Jessen, and Gabriele Scharf (in English).

Vol. 2 (3) 1995: *Language and Speech Pathology.* Master theses by Gabriele Scharf and by Jörg Mayer (in German) and papers by Hermann Ackermann, Ingo Hertrich, Jürgen Konczak, and Jörg Mayer (mostly in German).

Vol. 3 (1) 1997: *Tense versus Lax Obstruents in German.* Revised and expanded version of Ph.D. Dissertation, Cornell University, 1996 by Michael Jessen (in English). Full title: Phonetics and phonology of the tense and lax obstruents in German.

Vol. 3 (2) 1997: *Electroglottographic Description of Voice Quality.* Habilitationsschrift, University of Stuttgart, 1997 by Krysztof Marasek (in English).

Vol. 3 (3) 1997: *Aphasie und Kernbereiche der Grammatiktheorie (Aphasia and core domains in the theory of grammar).* Doctoral Dissertation, University of Stuttgart, 1997 by Annegret Bender (in German with abstract in English).

Vol. 3 (4) 1997: *Intonation and Bedeutung (Intonation and meaning).* Doctoral Dissertation, University of Stuttgart, 1997 by Jörg Mayer (in German with abstract in English).

Vol. 3 (5) 1997: *Koartikulation und glottale Transparenz (Coarticulation and glottal transparency).* Doctoral Dissertation, University of Bielefeld, 1997 by Kerstin Vollmer (in German with abstract in English).

Vol. 3 (6) 1997: *Der TFS-Repräsentationsformalismus und seine Anwendung in der maschinellen Sprachverarbeitung (The TFS Representation Formalism and its Application to Natural Language Processing).* Doctoral Dissertation, University of Stuttgart, 1997 by Martin C. Emele (in German).

Vol. 4 (1) 1998: *Automatisierte Erstellung von Korpora für die Prosodieforschung (Automated generation of corpora for prosody research).* Doctoral Dissertation, University of Stuttgart, 1998 by Stefan Rapp (in German with abstract in English).

Vol. 4 (2) 1998: *Theoriebasierte Modellierung der deutschen Intonation für die Sprachsynthese (Theory-based modelling of German intonation for speech synthesis).* Doctoral Dissertation, University of Stuttgart, 1998 by Gregor Möhler (in German with abstract in English).

Vol. 4 (3) 1998: *Inducing Lexicons with the EM Algorithm.* Papers by Mats Rooth, Stefan Riezler, Detlef Prescher, Sabine Schulte im Walde, Glenn Carroll, and Franz Beil. Chair for Theoretical Computational Linguistics, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart.

Vol. 5 (1) 1999: *Probabilistic Constraint Logic Programming: Formal Foundations of Quantitative and Statistical Inference in Constraint-Based Natural Language Processing* Doctoral Dissertation, University of Tübingen, 1999 by Stefan Riezler.

Vol. 5 (2) 1999: *Timing bei Dysarthrophonie (Timing in Dysarthria)* Doctoral Dissertation, University of Stuttgart, 1999 by Gabriele Scharf (in German with abstract in English).

Vol. 5 (3) 1999: *Semantikkonstruktion (Semantic Construction)* Doctoral Dissertation, University of Stuttgart, 1999 by Michael Schiehlen (in German with abstract in English).

YAP:
Parsing and Disambiguation With Feature-Based Grammars

von der Fakultät Philosophie der Universität Stuttgart
zur Erlangung der Würde eines Doktors der Philosophie (Dr. phil.)
genehmigte Abhandlung

vorgelegt von Helmut Schmid aus Reutlingen

Hauptberichter: Prof. Ph. D. Mats Rooth
Mitberichter: Prof. Dr. phil. Christian Rohrer
Tag der mündlichen Prüfung: 9. Februar 1999

Januar 2000

Institut für maschinelle Sprachverarbeitung der Universität Stuttgart

Helmut Schmid – YAP: Parsing and Disambiguation With Feature-Based Grammars

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Helmut Schmid:
YAP: Parsing and Disambiguation With Feature-Based Grammars /
Helmut Schmid – Stuttgart, 2000. AIMS - Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung, Vol. 6, No. 1, 2000, Stuttgart, Germany.
ISSN 1434-0763

Copyright © 2000 by Helmut Schmid

Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung
Lehrstuhl für Theoretische Computerlinguistik
Azenbergstraße 12
D – 70 174 Stuttgart

www: `http://www.ims.uni-stuttgart.de/~schmid`
email: `schmid@ims.uni-stuttgart.de`

Acknowledgements

The research presented here has been conducted while I was a member of the Graduiertenkolleg "Linguistische Grundlagen der maschinellen Sprachverarbeitung" at the Institute for Computational Linguistics of the University of Stuttgart. It would not have been possible without the help and support of my colleagues at the institute and within the Graduiertenkolleg.

Special thanks go to my supervisor Prof. Mats Rooth for his support and advice. I am also grateful to Andreas Eisele for all the discussions we had on various aspects of my work. Then I would like to thank Prof. Grzegorz Dogil who first got me interested in problems of Computational Linguistics while I was a computer science student, and Prof. Christian Rohrer who gave me the opportunity to actually work in this area.

Many thanks finally go to Esther König-Baumer, Stefan Riezler and Detlef Prescher for their comments on earlier versions of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Syntactic parsers belong to the most basic tools in natural language processing (NLP) and most NLP applications use some form of parser. If an NLP application does without a parser, the reason is often that no parser was available which was sufficiently fast, robust, and/or accurate, and which provides linguistically adequate and unambiguous analyses. There are parsers (e.g. parsers for feature-structure based grammar formalisms like HPSG and LFG) which provide sophisticated analyses, but they are often slow and not robust wrt. ill-formed input, and they either cover only a fragment of the language or their output is highly ambiguous. There are other parser (e.g. finite-state parsers) which are very fast, robust and which produce unambiguous output, but their analyses are incomplete and too simple for many linguistic phenomena. Finally there are parsers (the statistical parsers) which are highly robust, coping with all sorts of well-formed and ill-formed input, and which produce a single analysis (or a small number of likely analyses). But again, the generated analyses are too simplistic and some statistical parsers are also quite slow.

This thesis presents a new fast parsing algorithm for feature-structure based grammars, called YAP, and discusses the question of what grammar writers can do to accelerate parsing. The second part of the thesis describes an attempt to integrate statistical techniques into the YAP system in order to be able to rank analyses and to select the most likely ones.

Parsers need formal grammatical descriptions of the languages which they analyse. These descriptions are written in grammar formalisms such as Lexical Functional Grammar (LFG), Head-Driven Phrase Structure Grammar (HPSG) or Definite Clause Grammar (DCG) to name just a few. Many grammar formalisms including the formalisms just mentioned use feature structures to represent the syntactic properties of grammatical units. Efficient parsing with feature-structure based grammars is a difficult task, however. Parsing with LFGs, e.g., is undecidable in the general case, and even with suitable constraints such as the offline parsability constraint [Kaplan and Bresnan, 1982], it is still NP-complete which means that the parse time of any known algorithm grows exponentially with the length of the input in the worst case.

Fortunately, the worst case is not typical when natural languages are parsed. Natural languages seem to be only slightly more complex than context-free languages. Hence there is hope that an average parsing complexity not too far from the cubic complexity of context-free grammars could be achieved with real-world grammars and data.

The key to parsing efficiency is to avoid repeated computation of subproblems. To this end, intermediate results of parsing are usually stored in a table called a *chart*. The number of cells in the chart grows quadratically with the length n of the input sentence. In case of context-free parsing, the number of items (categories in case of the Cocke-Younger-Kasami algorithm, "dotted rules" in case of Earley's algorithm) in a cell of the chart is limited by a constant which depends only on the grammar. Each item can be built in at most n many ways, each requiring constant time wrt. sentence length. The runtime complexity of context-free parsing is therefore at worst cubic in the length of the input.[1]

In feature-structure based grammars, feature structures play the role of the category labels. So, a chart parser for feature structure-based grammars inserts feature structures rather than categories into the chart. Because the number of different feature structures is infinite, however, there is no limit to the number of items in a chart cell, and the cubic runtime complexity is lost.

Maxwell and Kaplan [Maxwell III and Kaplan, 1996] show that it is possible to circumvent this problem if constituents with different feature structures but spanning the same input are merged by pushing down the disjunction from the topmost level to the lowest possible level, as long as the feature structures do not differ in their "relevant" parts. The relevant parts of a feature structure are those which may later lead to a feature clash. If the "relevant" substructures of the feature structures of a grammar form a finite set, the grammar can even be parsed in cubic time.

The question arises, however, why syntactically irrelevant information is represented in the grammar at all, if sophisticated algorithms are required to reduce their detrimental effects on parse time. In the LFG formalism, the feature structure (called *f-structure*) of the topmost node represents the function-argument structure of the whole sentence. The f-structure is therefore a simple semantic representation. Because most local ambiguities are reflected in the predicate-argument structure, they are propagated to the dominating nodes and their feature structures. The problem disappears if the semantic representation is built in a separate step.

The YAP formalism reflects these considerations. It is a feature-based grammar formalism which is solely intended for syntactic analysis. It combines ideas from LFG, HPSG and procedural programming languages. Similarly to LFG, the formalism uses grammar rules with a context-free backbone and feature structure constraints. Like HPSG, it is a typed formalism, but the type system resembles more that of a programming language. Subcategorization and argument extraction are handled similarly as in HPSG, namely with argument cancelling from *Subcat* lists and *gap threading* via *Slash* features. A particularity of the YAP formalism is the possibility to compile features with finite value range into the context-free backbone of the grammar to enhance efficiency.

The parser for this grammar formalism uses a novel iterative method for the computation of the feature structures after the context-free backbone of the syntactic analysis has been computed by a standard context-free parser. Feature structures are represented as trees without reentrancies rather than graphs, and disjunctions are restricted to the atomic level. The feature structures are computed by iteratively solving local feature constraints. The parsing algorithm is guaranteed to terminate for cycle-free feature structures and is very

[1] A formal proof of this result is found e.g. in [Younger, 1967] and [Earley, 1970].

fast in practice. The simplicity of the data structures and operations facilitates an efficient implementation of the parser.

Another important problem in parsing is disambiguation. Many sentences are syntactically ambiguous. Some ambiguities may be solved with a simple rule preference mechanism which favours e.g. analyses with fewer traces. Other ambiguities require some sort of semantic knowledge to be resolved. Examples are PP attachment ambiguities as in the sentence `I looked at the moon through a telecope` where `through a telecope` could – according to strictly syntactic criteria – either modify `the moon` or the verb `looked`, and coordination ambiguitites as in `They met with the finance ministers of France and the United States` where either `France` and `the United States` or `the finance ministers of France` and `the United States` could coordinate.

Hindle and Rooth [Hindle and Rooth, 1993] presented a simple statistical algorithm which correctly disambiguates most PP attachment problems. It compares how likely the prepositional phrase is to attach to the different possible attachment sites and chooses the most likely attachment. This approach is extended in this thesis to resolve other types of ambiguity as well. The statistical parameters of the algorithm were determined from a 100 million word corpus which was parsed and disambiguated with a parser for head-lexicalized context-free grammars [Carroll and Rooth, 1998].

The chapters of this thesis are organized as follows. Chapter 2 presents the YAP grammar formalism in detail. A feature logic for this formalism is developed in chapter 3. This chapter also contains a proof for the correctness of the iterative method for the feature computation. Chapter 4 describes the implementation of the parsing algorithm and chapter 5 shows results of experiments with the parser. Chapter 6 presents the disambiguation method and and reports on preliminary disambiguation results. Chapter 7 summarizes the main points of this thesis and gives an outlook to future work.

Chapter 2

Grammar Formalism

YAP is a symbolic parser for a feature-based grammar formalism. It assigns a set of parse trees decorated with feature structures to sentences. Each parse tree represents one analysis of a sentence. Fig. 2.1 shows a sample parse tree which was printed with the Xmfed program [Groenendijk, 1993]. The small boxes represent "imploded" feature structures. The grammar formalism which is used to express the constraints which hold for well-formed parse trees is based on phrase structure rules. It is somewhat similar to the PATR-II formalism [Shieber, 1992], but in contrast to PATR-II the feature structures are *typed* [Carpenter, 1992]. The type of a feature structure determines its set of features.

Feature structure typing has several advantages. It allows for a more efficient representation of feature structures because feature names do not have to be represented explicitly. Instead of a list of feature-value pairs, we only store an array of feature values. Access to feature values is faster because feature names are translated to indices into the feature value array. No searching of lists is required.

Feature typing also facilitates grammar development. The compiler knows which features and feature values are appropriate in any given context and is therefore able to detect many errors at compile time, e.g. misspelled feature names or erroneous feature paths. Typing also supports a more structured approach to grammar writing.

2.1 Declarations

A grammar description in the YAP formalism starts with a set of declarations which specify the grammatical categories, the features and the feature types.

2.1.1 Category and Feature Declarations

A *category declaration* starts with the keyword `category` followed by the category name and a list of feature declarations enclosed in braces. The declaration is terminated by a semicolon.

A *feature declaration* in turn consists of the feature value type followed by the feature name and a semicolon. A feature declaration is local to a category declaration. It is possible to

use the same feature name in different category declarations, and even with different feature types.

Example:[1]

```
category NP {
   STRING Phon;
   NUMBER Number;
   GENDER Gender;
   CASE   Case;
};
```

The YAP categories are special feature value types (see the next section) which can never appear in a feature declaration, i.e. no feature value will ever have a "category" type. The only feature structures with a category type are the root nodes of feature structures and the elements of feature structure lists (see below).

2.1.2 Feature Type Declarations

There are two *predefined feature types*: `STRING` and `FS_LIST`. Features of type `STRING` may have any character string as value. `FS_LIST` features take a list of complete feature structures (i.e. feature structures whose types are categories) as values.

Enumeration Types

Besides the predefined feature types, there are two classes of *user-defined feature types*: enumeration types and structured types. The definition of an *enumeration feature type* starts with the keyword `enum` followed by the name of the feature type and the list of possible values enclosed in braces. Again, the definition is terminated by a semicolon.

Enumeration types have two advantages over STRING types: They allow for a more efficient representation of ambiguities and they provide information for error detection so that misspelled feature values are detected at compile time. Therefore enumeration types should be preferred whenever the limit of 32 possible feature values is not exceeded. Agreement feature value types are typical examples of enumeration types:

```
enum NUMBER {sg, pl};
enum GENDER {masc, fem, neut};
```

Structured Types

Structured feature types are used to group a set of features. The definition of a structured feature is analogous to that of a category, except for the new keyword `struct`.

[1] The following spelling conventions will be used throughout this book, although they are not enforced by the grammar formalism. Categories and feature types are upper case. Feature names are capitalized. Feature values and variables are lower case. Variables usually have one or two characters and feature values three or more characters.

```
struct AGR {
   NUMBER Number;
   GENDER Gender;
   CASE   Case;
};
```

The types of the features in such a definition must have been defined before. Therefore cyclic feature definitions, where a feature has the same type as one of its subordinated features, are excluded unless there is an intervening feature of type FS_LIST.

2.1.3 Variable Declarations

Variables create dependencies between features in grammar rules. A variable declaration resembles a feature declaration: it consists of the type of the variable followed by the name of the variable and a semicolon. Variable declarations are always global, i.e. there is always exactly one declaration for a given variable name. Nevertheless, the *value* of a variable is always defined locally to a rule. In other words, each rule uses its own copy of the variable. All feature types (STRING, FS_LIST, enumeration types and structured types) and the restrictor types introduced in the next section are valid *variable types*.

```
AGR    a;
STRING p1;
```

Restrictor Types

Restrictor types are special variable types. They are used to exclude certain features from being unified when two complete feature structures are unified. When the feature structures of a trace node and its filler are unified, for instance, the `Phon` feature has to be excluded because the `Phon` feature of a trace is always the empty string.

The declaration of a restrictor type starts with one of the keywords `restrictor+` and `restrictor-` followed by the name of the restrictor type, a category name enclosed in parentheses, a list of feature names enclosed in braces and a terminating semicolon. The list of features has to be a subset of the set of features defined for the given category. The difference between the two keywords is the following: `restrictor+` instructs the compiler to unify all features on the list, whereas `restrictor-` instructs the compiler to unify all features which are not on the list.

The following statements define the restrictor types `NP_R` and `NP_R2` and the variables `np` and `np2`. Variable `np` might be used to unify the feature structure of an NP argument with a feature structure on the `Subcat` list of the verb. Variable `np2` might unify the feature structure of an `NP` trace with the feature structure of its filler and variable `np3` could check agreement between a relative pronoun and the preceding noun. See the next section for two sample rules.

```
restrictor- NP_R(NP)  {};
```

```
restrictor- NP_R2(NP) {Phon};
restrictor+ NP_R3(NP) {Number, Gender};
NP_R  np;
NP_R2 np2;
NP_R3 np3;
```

Remark: The *restrictor types* only make grammar development more convenient in contrast to the *restrictor functions* in [Shieber, 1992] which are needed to guarantee termination of the parsing process.

2.2 Grammar Rules

A *grammar rule* in the YAP formalism consists of a mother node specification, followed by an arrow, a sequence of daughter node specifications and a semicolon. Each *node specification* in turn consists of a category name and a set of feature constraint equations which are enclosed in braces. One of the daughter nodes is marked as the *head daughter* with a backquote (‘) in front of the category name.

```
NP {Phon=cat(p1,p2);Number=n;Gender=g;Case=c;} ->
      DT {Phon=p1;Number=n;Gender=g;Case=c;}
      ‘N {Phon=p2;Number=n;Gender=g;Case=c;};
```

A *constraint equation* consists of a feature path, an equals sign (=), a feature value and a terminating semicolon. Possible *feature values* are variables, constants, function terms and feature structure lists. The only function currently defined is the string concatenation operator `cat`. A function term therefore consists of the function name `cat` and a comma-separated list of arguments which is enclosed in parentheses. Variables of type `STRING` are the only accepted arguments. The type of such a function term is also `STRING`.

A *Feature path* is a sequence of feature names which are separated by dots (.). Each feature must have been declared as a subfeature of the feature value type of the preceding feature in the sequence. The first feature must have been declared as a feature of the current category.

If the list of feature names in a feature path is empty, the constraint equation has to move behind the closing brace and the terminating semicolon is dropped. So, instead of NP{=np;}, we get NP {}=np.

A *feature structure list* consists of a set of comma-separated node specifications optionally followed by a vertical bar (|) and a variable of type `FS_LIST`. It is enclosed in square brackets. This notation is analogous to the notation in the Prolog programming language if we ignore the fact that Prolog variables have to be capitalized. The type of a feature structure list is `FS_LIST`.

The following grammar rule unifies the first element of the `Subcat` list with the feature structure of the argument and propagates the remaining arguments to the parent node.

```
VP {Subcat=r;} -> ‘V {Subcat=[NP{}=np|r];} NP {}=np;
```

If a node is marked with an asterisk (*) after the category name, then it is a *trace node* and it generates an empty string.

The following rule generates an `NP` trace node whose feature structure is unified with the first element of the `Subcat` list. Furthermore the `NP` feature structure minus the `Phon` feature is unified with the element on the Slash list of the parent node. The next rule generates the subject of the sentence and percolates the Slash feature. The last rule generates the filler and unifies its feature structures with the feature structure on the `Slash` list. Fig. 2.2 shows the parse tree for the clause "who Mary loves".

```
VP {Subcat=r;Slash=[NP{}=np2];} ->
   ‘V {Subcat=[NP{}=np|r];Slash=[];}
   NP*{}=np=np2;
S {Slash=r;} -> NP {}=np ‘VP {Subcat[NP{}=np];Slash=r;};
SBAR {} -> NP {}=np ‘S {Slash=[NP{}=np];};
```

The feature path on the left side and the value on the right side of a constraint equation must have the same type. If the feature path is empty, then the value must bear a restrictor type for the category of the corresponding node. If the feature path is not empty, then the type of the last feature in the path has to be identical to the type of the right side.

Structured features often simplify grammar rules. With a structured `Agr` feature as declared above and an appropriate declaration of the NP category, it is possible to simplify the above NP rule. A single constraint equation is now sufficient to enforce agreement in number, gender and case.

```
category NP {
   STRING Phon;
   AGR    Agr;
};

NP {Phon=cat(p1,p2);Agr=a;} ->
      DT {Phon=p1;Agr=a;}
      ‘N {Phon=p2;Agr=a;};
```

2.3 Lexicon Entries

A lexicon entry in the YAP formalism consists of the word form enclosed in double quotes (") followed by a colon (:), a node specification and a semicolon. If there is more than one lexicon entry for a word form, the different entries constitute alternatives.

```
"him" : NP {Agr.Number=sg;Agr.Gender=masc;Agr.Case=acc;};
```

2.4 Further Grammar Elements

So far, the essential parts of the YAP grammar formalism have been presented. The formalism contains some additional features mainly to facilitate grammar development by reducing the redundancy in grammar descriptions.

2.4.1 Templates

If a set of constraint equations is repeated in several rules, it is useful to define a *template* for it. A template definition consists of the template name, a colon (:), a node specification and a semicolon. Templates and lists of &-separated templates can replace category names in node specifications.

```
NPsg:   NP {Agr.Number=sg;};
NP3:    NP {Agr.Person=3rd;};
NPacc:  NP {Agr.Case=acc;};
NPm:    NP {Agr.Gender=masc;};
NPf:    NP {Agr.Gender=fem;};

NP3sma: NPsg & NP3 & NPacc & NPm {};
NP3sfa: NPsg & NP3 & NPacc & NPf {};

"him":  NP3sma {};
"her":  NP3sfa {};
```

The compiler expands the templates by adding the constraints from the template definition to the constraints of the current node. The scope of variables is always local to a template definition, lexicon entry or grammar rule. In other words, the compiler renames the variables used in a template before it is expanded.

2.4.2 Generic Entries

Some classes of words like numbers are difficult or even impossible to list exhaustively because there are so many of them. *Generic lexicon entries* may be used to provide lexical information for a predefined set of such word classes.

In a generic entry, the quoted word form is replaced by either <`cardinal`>, <`ordinal`>, <`propername`> or <`default`>, defining entries for cardinal numbers, ordinal numbers, proper names and other unknown word forms, respectively.

Generic entries have low priority. They are only accessed if there is no regular lexicon entry for a word. Among the generic entries, default entries have lower priority than the others. They are only accessed if no other generic entry matches.

2.4.3 Feature Inheritance

The mother node of a rule and its head daughter usually share a large set of features. In order to relieve the grammar designer from the tedious task of writing all these constraints and to keep the grammar more concise and easier to read, the YAP compiler automatically inserts a constraint whenever a feature is undefined otherwise. The corresponding *feature inheritance rule* states that

1. if the value of a feature of the head daughter is unspecified, and
2. a feature with the same name and type is declared for the mother node,

   then add a constraint which unifies the values of both features.

A similar rule applies in the other direction:

1. If the value of a feature of the mother node is unspecified, and
2. a feature with the same name and type is declared for the daughter node,

   then add a constraint which unifies the values of both features.

Sometimes, the default inheritance rule has undesired effects. Section 2.4.6 will show how it is overridden.

2.4.4 Automatic Features

There are some features which are usually declared for all categories and which are always computed in the same way. This is the case for the `Phon` feature which contains the portion of the input string which is covered by a constituent.

The compiler will automatically insert declarations for the `Phon` feature and constraints for the computation of its value if the grammar contains the command:

```
auto Phon;
```

Besides `Phon` there is another *automatic* feature called `HeadLex` which contains information about the lexical head of a constituent. The value of the HeadLex feature is automatically defined by the feature inheritance rule. In lexicon entries, however, the value of the HeadLex feature has to be specified by the grammar writer.

```
"cats": N_pl {HeadLex="cat";};
```

Since the feature inheritance rule is a default rule which applies only if a feature is undefined otherwise, it can be overridden. In the following rule for noun compounds, e.g., the value of the `HeadLex` feature of the mother node is defined as the concatenation of the `HeadLex` features of the daughter nodes.

```
N {HeadLex=cat(h1,h2);} -> N {HeadLex=h1;} `N {HeadLex=h2;};
```

2.4.5 Disjunctive Values

Linguistic entities are often ambiguous. A good example is the German determiner `die` which is either feminine singular or a plural form of any gender. Independently of this alternative, the case of `die` is either nominative or accusative (cp. table 2.1).

	masc sg	fem sg	neut sg	masc pl	fem pl	neut pl
nominative		x		x	x	x
genitive						
dative						
accusative		x		x	x	x

Table 2.1: Morphosyntactic features of the German determiner `die`

Four lexicon entries are needed for `die`:

```
"die" : DT {Gender=fem;Number=sg;Case=nom;};
"die" : DT {Gender=fem;Number=sg;Case=acc;};
"die" : DT {Number=pl;Case=nom;};
"die" : DT {Number=pl;Case=acc;};
```

The third and the fourth entry each covers three possibilities since `gender` is unspecified. Leaving the value of the `case` feature ambiguous between `nominative` and `accusative` would further reduce the number of entries to two. To this end, *disjunctive feature values* are introduced into the formalism. A disjunctive feature value is a list of comma-separated constant values which is enclosed in parentheses.

```
"die" : DT {Gender=fem;Number=sg;Case=(nom,acc);};
"die" : DT {Number=pl;Case=(nom,acc);};
```

2.4.6 Dummy Values

Sometimes it is necessary to block the feature inheritance rule. In the NP-coordination rule below, the `Number` and the `Person` feature of the coordinated NP are different from that of the head daughter. Feature inheritance is inhibited by assigning a disjunction of all possible values to the corresponding features of the daughter node.

```
NP {Number=pl;Person=3rd;} ->
       NP {} CC {HeadLex="and";}
      'NP {Person=(1st,2nd,3rd);Number=(sg,pl);};
```

Enumerating the possible values takes a lot of space and is not always possible (e.g. not for `STRING` features). The *dummy value* `*` can be used instead. It marks the respective feature value as specified without restricting it in any form.

```
NP {Number=pl;Person=3rd;} ->
       NP {} CC {HeadLex="and";}
      'NP {Person=*;Number=*;};
```

The dummy value is also used as a proxy for elements on feature structure lists. In the following rule, the `Subcat` value of the VP node is a list with one element of an arbitrary category.

```
VP {Subcat=r;Subcat=[*];} -> 'V {Subcat=[NP{}=np|r];}  NP {}=np;
```

2.4.7 Abbreviations

An expression of the form `XP{}=xp` may be abbreviated as `xp` if it is embedded, i.e. if it is an element of a feature structure list.

A set of constraint equations with the same left hand side like `f=a; f=b;` can be contracted to one equation `f=a=b;`. A set of feature or variable declarations with the same type is also contractable, e.g. `FS_LIST Subcat,Slash;` or `STRING p1,p2;`.

A shorter version of the above VP rule is:

```
VP {Subcat=[*]=r;} -> 'V {Subcat=[np|r];}  NP {}=np;
```

2.5 Missing Features in the Formalism

Although the YAP formalism is a simple grammar formalism, it is nevertheless expressive enough to allow for a linguistically motivated description of syntactic phenomena more or less along the lines of HPSG theory [Pollard and Sag, 1994]. However, the descriptions are sometimes less general in the YAP formalism because it is based on explicit phrase structure rules rather than on general principles about well-formedness. Some features which are familiar from other grammar formalisms are missing in the YAP formalism. These will be discussed now.

2.5.1 Semantic Representations

Feature structures are sometimes used to integrate phonological, syntactic, semantic and other information in a single data structure. Since all the information is available at the same time, constraints from different levels of linguistic analysis can be processed in parallel. It is even possible that one and the same constraint refers to information from linguistic levels as different as, say, phonology and pragmatics.

For several reasons, YAP does not follow this approach. First, it is contrary to the concept of *modularization*, which is generally regarded as an important technique for keeping large software projects manageable by splitting them into separate modules, which hide irrelevant information from each other. Large NLP development projects like the Verb*mobil* project [Wahlster, 1993] consequently use modular systems with separate components for

speech recognition, parsing, semantic construction, discourse analysis, transfer and generation, and specify well-defined interfaces.

Some experiments also indicate that modularization is advantageous with respect to processing speed. Kasper and Krieger [Kasper and Krieger, 1996] report a speedup of their HPSG parser resulting from delaying semantic construction to a second pass after parsing. A delay mechanism was also implemented in the Alep grammar development system [Simpkins, 1994].

There is also reason to believe that different processing strategies are required for syntactic analysis and semantic construction. During parsing it is important to eliminate invalid analyses as quickly as possible. Few analyses usually remain after parsing, and for most of them, a consistent semantic representation can be built. Therefore the size of the search space is a problem in parsing but not in semantic construction. Of course, there is often a large search space to be explored after the initial semantic representation has been built in order to find e.g. antecedents for pronouns and information satisfying presuppositions. These tasks are more or less orthogonal to the task of semantic construction, however, and are, again, best accomplished in separate steps.

On the other hand, the *size* of the feature structures is more or less constant during parsing (if the grammar is designed appropriately, s. section 2.6), whereas it grows about linearly with the number of dominated nodes during semantic construction. The *number* of alternative semantic representations for a constituent often grows exponentially with sentence length, because the semantic representation of a node reflects syntactic ambiguities. Hence an efficient representation of ambiguity in feature structures is essential for semantic construction, but not so for parsing.

Since the requirements on syntactic parsing and semantic construction are so different, it seems more promising to develop specialized processing strategies for both of them. Algorithms for semantic construction have been presented in [Schiehlen, 1996] and [Dörre, 1997].

2.5.2 Type Hierarchies

Feature structures in the YAP formalism are typed, but in contrast to many other typed grammar formalisms, the types are not ordered. In this respect, the features types of YAP resemble data types of programming languages like Modula or C. Type checking is mostly done at compile time and there is no need for type inference at runtime.

Feature types mainly serve two purposes in the YAP formalism: They extend the error detection capabilities of the compiler compared to untyped formalisms like the *Lexical Functional Grammar* and they allow a more efficient representation of the feature structures.

Feature types mainly define the structure of feature structures. If the type of a feature structure is known, it is possible to deduce the whole feature tree except for the values of terminal nodes and the number and the categories of feature structure list elements. These two define the informational content of the feature structure.

The typing system restricts the grammar designer with respect to the granularity of the grammar rules. While it easy to make a grammar rule more specific by adding further feature constraints, there is a limit to the level of abstraction because grammar rules cannot abstract over categories. Therefore it is necessary e.g. to write a coordination rule for each category

which might be coordinated instead of one general rule for all of them. Using a single category X for all feature structures is not a practical solution to this problem because it would mean that all feature structures have the same set of features. It follows that certain generalizations which are desirable from a linguistic point of view cannot be expressed in the YAP formalism.

2.5.3 Head Movement

In contrast to argument movement, the YAP formalism provides no general mechanism for head movement, though it is possible to generate a head trace together with a non-empty constituent as e.g. in the following rule:

```
VP {Subcat=r2;Slash=[v_p];} ->
        ‘V* {Aux=+;VForm=fin;Subcat=[vp_p|r2];}=v_p
         VP {Subcat=r2;Slash=[];}=vp_p;
```

It would be better to handle this type of movement more generally with a lexicon entry for verb traces and a more general VP rule which is needed anyway:

```
VP {Subcat=r2;Slash=r;} ->
        ‘V {Aux=+;VForm=fin;Subcat=[vp_p|r2];}
         VP {Subcat=r2;Slash=r;}=vp_p;
"": V {Aux=+;VForm=fin;Slash=[V{Aux=+;VForm=fin;Slash=[];}];};
```

Such a rule is problematic, however, because the context-free part of the grammar is weakened to the point where it hardly constrains anything. Just assume that the grammar also contains the following rules:

```
  VP {...} -> V {...};             % intransitive verb
   S {...} -> NP* {...} VP {...}; % Wh-movement of a subject
SBAR {...} -> NP* {...}  S {...}; % relative clause without relative
                           % pronoun, e.g. "the film I saw yesterday"
```

The context-free part of this grammar is able to generate vacuous V's, VP's, S's and SBAR's at any position of the input string. Due to this problem, a general mechanism for head movement has not been included in the YAP formalism.

2.5.4 Unrestricted Feature Structure List Elements

Feature structure lists (`FS_LIST`) are used to implement subcategorization lists and Slash features. The elements of feature structure lists are restricted to be feature structures of a `category` type. This contrasts with current HPSG theory where the elements of subcat lists are `synsem` values rather than signs in order to enforce the linguistically motivated constraint that a lexical item not refer to the phonology or constituent structure of its arguments. The otherwise unmotivated union of syntactic and semantic features to the `synsem` feature is the price paid for this.

The YAP formalism is not intended to enforce linguistically motivated constraints. The restrictor types exclude the features which are not to be unified, allowing the grammar designer to define flatter and more readable feature structures.

So far, there was no real need to allow other elements than categorial feature structures on feature structure lists.

2.6 Grammar Design Considerations

The major advantages of the YAP formalism compared to other formalisms are extensive error detection capabilities and ease of efficient implementation.

For maximal parsing efficiency, however, the grammar designer should adhere to the rule that the feature structure of a node only contains information which is relevant to its syntactic behaviour. Semantic representations have to be built by a separate module specifically designed for this task as discussed in section 2.5.1. Explicit links to the feature structures of daughter nodes (like the `Daughters` feature in HPSG) are to be avoided. They merely replicate information already contained in the chart of the parser. Finally, information about arguments should not be propagated upward in the parse tree once the argument has been generated. HPSG-style `Subcat` lists, where elements are cancelled from the list as soon as they are generated, are appropriate, whereas LFG-style argument slots are problematic because their information is propagated to the root node in order to build a simple semantic representation.

These grammar design rules do not restrict the syntactic phenomena which the grammar writer is able to describe, but merely the way they are described.

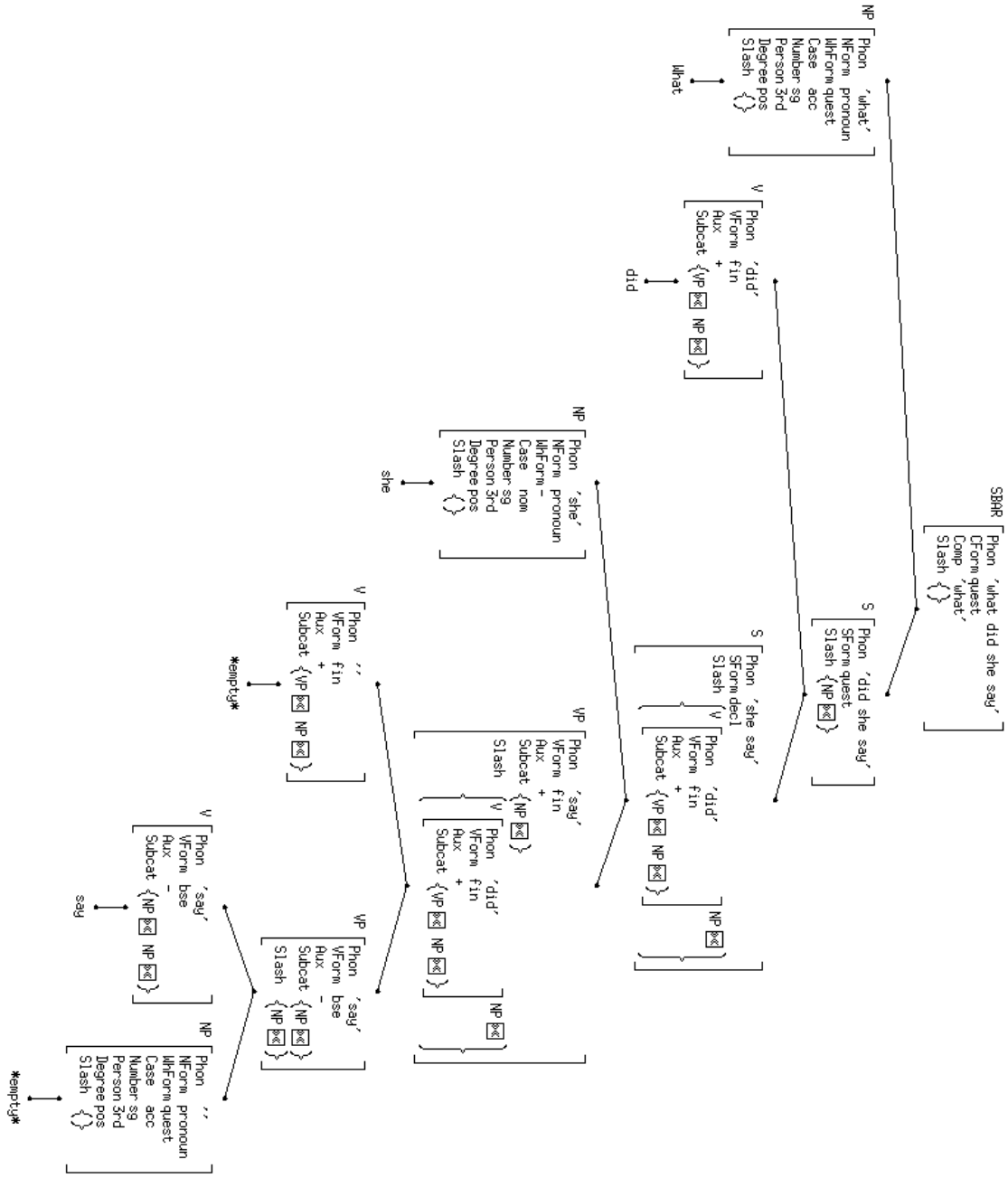

Figure 2.1: A sample parse tree

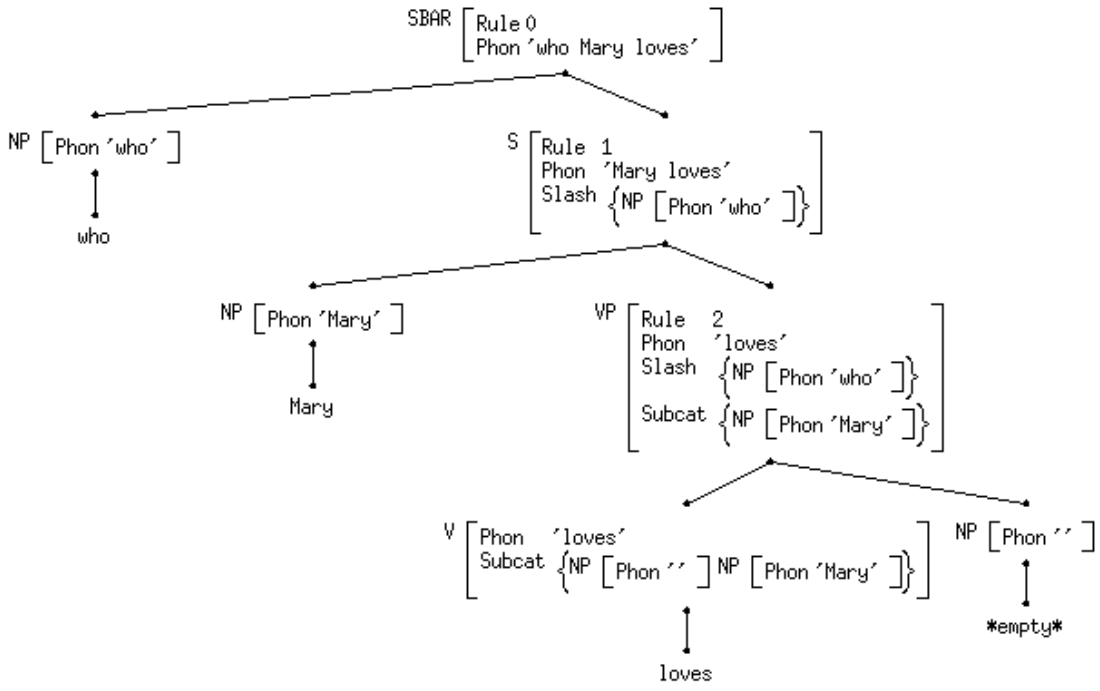

Figure 2.2: Parse tree with feature structures illustrating the Slash percolation mechanism.

Chapter 3

Theoretical Foundations

In the previous chapter, the grammar formalism of YAP has been presented. In this chapter, we will formally define a feature logic for the grammar constraints. Feature trees will be introduced as models for the feature logic and an algorithm for the computation of the feature trees will be presented and its correctness will be proved. Finally, it will be shown how YAP constraints are converted to formulas of the feature logic.

As in Shieber's dissertation, the presented feature logic is based on its own set of axioms rather than on standard predicate logic. This seems a reasonable approach since we are mainly interested in a proof of the correctness of the parsing method. All definitions and proofs which do not directly relate to the parsing algorithm can be found in similar form in [Shieber, 1992]. The purpose of this chapter is not to introduce a new feature logic but to prove the correctness of the presented parsing method. Readers familiar with Shieber's thesis may skip sections 3.1 and 3.2.

The notation in this chapter is partially adopted from [Shieber, 1992]. If nothing else is stated, p, p', q, r will denote feature paths, c, c' will denote constants and v, v' will denote entities which are either constants or feature paths. ϕ and ψ will denote single constraints and Φ and Ψ will denote sets of constraints. A set of feature constraints will be called a formula of the feature logic.

3.1 The Feature Logic

The definition of the feature logic follows the definition in [Shieber, 1992].

Definition 1: A *feature path* is a finite sequence of labels from a finite set of feature labels L. $Path$ is the set of all feature paths.

A feature path will be written as a sequence of labels which is enclosed in angle brackets, e.g. $\langle f_1 f_2 \ldots f_n \rangle$. Concatenation of paths is notated with a the center dot ('$\cdot$').

The length of a feature path p (written $|p|$) is defined in the obvious way.

Definition 2: A *constraint equation* is an equation of the form $p \doteq q$ or the form $p \doteq c$, where p and q are feature paths and c is a constant from a set of constants C.

The symbol $\doteq$ is used rather than $=$ in order to avoid confusion with the equality symbol of the meta-language.

We will now define what it means for a constraint set to be consistent. The definition formalizes the requirements that feature structures cannot be atomic and compound at the same time (constant/compound clash), and that the label of an atomic feature structure is unique (constant/constant clash).

Definition 3: A set of constraint equations is *consistent* iff it does not entail a pair of constraints of either of the following two forms:

Constant/constant clash:	$p \doteq c$ and $p \doteq c'$ where $c \neq c'$ or
Constant/compound clash:	$p \doteq c$ and $p \cdot \langle f \rangle = v$.

Entailment in the feature logic is defined by the inference rules:

Triviality:	$\vdash \langle\rangle \doteq \langle\rangle$
Symmetry:	$p \doteq q \vdash q \doteq p$
Reflexivity:	$p \cdot r \doteq v \vdash p \doteq p$
Substitutivity:	$q \doteq p,\ p \cdot r \doteq v \vdash q \cdot r \doteq v$

$\Phi \vdash \phi$ means that ϕ is deducible from Φ by means of the above inference rules.

Definition 4: A set of constraint equations Φ is called *cyclic* if two feature paths p and q exist such that $q \neq \langle\rangle$ and $\Phi \vdash p \cdot q \doteq p$.

We will now prove that the length of the feature paths in all constraints which are deducible from an acyclic set of feature constraints is limited by a constant. This lemma will help us to prove that the feature computation algorithm to be presented in section 3.4 terminates.

Lemma 1: If Φ is a finite, acyclic set of constraint equations, then there is an $N \in \mathcal{N}$ such that for all constraints $\phi = p \doteq v$ which are deducible from Φ, the length of path p is shorter than N.

Proof: We prove this lemma with a pigeon hole argument and set $N := 2 + \sum_{p \doteq q \in \Phi} |p| + |q| - 1 + \sum_{p \doteq c \in \Phi} |p|$.

We will first show that at most $N - 1$ many different feature paths are defined, i.e. there are at most $N - 1$ many feature paths $p_1, p_2, \ldots, p_{N-1}$ such that $\forall_{1 \leq i < N} [\exists_v \Phi \vdash p_i \doteq v$ and $\forall_{1 \leq j < N; i \neq j} \Phi \nvdash p_i \doteq p_j]$.

Each constraint equation $p \doteq q \in \Phi$ introduces two paths, namely p and q. Because of $p \doteq q$, the constraint adds at most one *new* path to the set of different paths. Each constraint equation $p \doteq c \in \Phi$ also adds at most one new path to the set of different paths.

The Triviality inference rule adds at most one new path, namely the empty path $\langle\rangle$ to the set of different paths. The Symmetry rule does not add any new paths. The Reflexivity rule adds for a rule of the form $\langle f_1 f_2 \ldots f_n \rangle \doteq c$ at most $n - 1$ new paths to the set of different paths. (The empty path has already been added by the Triviality rule and $\langle f_1 f_2 \ldots f_n \rangle$ has

been added as well.) Similarly, the Reflexivity rule adds at most $|p| + |q| - 2$ new nodes for a constraint $p \doteq q$.

The Substitutivity rule $q \doteq p, p \cdot r \doteq v \vdash q \cdot r \doteq v$ does not add any new paths to the set of different paths because for any $q \cdot r'$ with $r = r' \cdot r''$ (including $r = r'$), it follows that $q \cdot r' \doteq p \cdot r'$. (Proof: $q \doteq p, p \cdot r \doteq v \vdash p \cdot r' \doteq p \cdot r' \vdash q \cdot r' \doteq p \cdot r'$)

Therefore there are at most $N - 1 = 1 + \sum_{p \doteq q \in \Phi}(|p| + |q| - 1) + \sum_{p \doteq c \in \Phi} |p|$ many different paths. Now assume that $\Phi \vdash p \doteq v$ for some p with $|p| > N$, then $q \doteq q$ follows by the Reflexivity rule for all q such that $q \cdot r = p$ for some r. There are at least N many such q. Because there are less than N many different paths, it follows that $q = q \cdot r$ with $r \neq \langle\rangle$ for two prefixes $q, q \cdot r$ of p. Hence the set of constraints must be cyclic in contradiction to the assumption that it is not. □

3.2 Feature Structure Models

In this section, we will define models for feature constraint sets. Stuart Shieber considers in his dissertation [Shieber, 1992] a range of feature structure models and assesses them in terms of denotational soundness, logical soundness, logical completeness, minimal model existence, categoricity, finiteness of minimal models and computability of operations. These properties of models are defined as follows:

Property 1: (DENOTATIONAL SOUNDNESS) For all formulas Φ from a feature logic $\mathcal{L}$, if there is a model M in the set of models $\mathcal{M}$ such that $M \models \Phi$, then Φ is consistent.

Denotational soundness implies that models exist only for consistent feature constraints.

Property 2: (LOGICAL COMPLETENESS) For all formulas Φ and Φ' of $\mathcal{L}$, if all models of Φ are also models of Φ', then $\Phi \vdash \Phi'$.

Logical completeness requires that logical inferences hold for all semantically entailed formulas.

Property 3: (DENOTATIONAL COMPLETENESS) For all formulas Φ of $\mathcal{L}$, if Φ is consistent, then there is a model $M \in \mathcal{M}$ such that $M \models \Phi$.

There must be a model for each consistent set of formulas.

Property 4: (LOGICAL SOUNDNESS) For all formulas Φ and Φ' of $\mathcal{L}$, if $\Phi \vdash \Phi'$, then all models of Φ are also models of Φ'.

Property 5: (CATEGORICITY) For any two distinct models M and M', there exists a formula Φ such that either $M \models \Phi$ and $M' \not\models \Phi$ or vice versa.

Categoricity requires that any two models are distinguished by some formula. We now define an ordering on models (called subsumption) such that a model subsumes another model if it has less information, i.e. it satisfies fewer formulas.

Definition 5: A model M *subsumes* another model M' (written $M \leq M'$) iff for all formulas Φ of $\mathcal{L}$, $M' \models \Phi$ whenever $M \models \Phi$.

For computational purposes, it is useful to have a unique minimal model for each consistent formula. All computations will be carried out on these minimal models.

Property 6: (MINIMAL MODELS) if Φ is a consistent formula, then there is a model M such that $M \models \Phi$ and for all M' such that $M' \models \Phi$, it is the case that $M \leq M'$.

There is a class of models which has all the above properties, namely the *feature graphs*. Feature graphs are rooted, directed graphs with labelled arcs and labelled terminal nodes. They form canonical models for the feature logic in the sense that two sets of feature constraints are logically equivalent if and only if their minimal models are equivalent. Therefore it is possible to replace operations on constraint sets by operations on the feature graphs which model the constraint sets. In particular, it is possible to compute a model for the union of two constraint sets by unifying their minimal feature graph models.

Graph models are well-suited models for the feature logic, but they are not the only possible models. One alternative are *finite trees*. Finite trees are trees with labelled arcs and terminal nodes. They are the simplest feature structure models that Shieber considers. He identifies the following problems with finite tree models (feature trees):

- Feature trees are not *denotationally complete* because they fail to model cyclic feature structures. There is no feature tree, e.g., which satisfies the constraint $\langle\rangle \doteq \langle f \rangle$. (A feature *graph* satisfying this constraint is shown in fig. 3.1)
- Feature trees are not *logically complete* (if the set of labels is finite). Each model for the formula $\Phi = \{\langle f\, x\rangle \doteq a | f \in \{f_1, f_2\}, x \in L\}$ also models $\phi = \langle f_1 \rangle \doteq \langle f_2 \rangle$. However, $\Phi \not\models \phi$. This problem arises because feature equality in feature trees is extensional (two features are equal if they have the same value) whereas equality in the feature logic is intensional (two features p and q are only equal if an explicit equality constraint $p \doteq q$ is entailed by the constraint set).

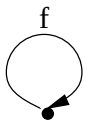

Figure 3.1: Cyclic feature structure

As Shieber notes, the denotational completeness of feature trees is recovered if cyclic constraints and infinite constraints are disallowed. Because neither of these types of constraints seems useful as a syntactic constraint[1], this restriction is acceptable for syntactical analysis.

Logical completeness of feature trees is recovered if feature trees contain additional information about which features have been set equal. Shieber calls this type of model *Eqtree*. We will not consider them here because they offer no advantages over feature graphs.

Although simple feature trees are not the ideal models for feature constraints from a theoretical point of view, they are nevertheless useful to implement a parser because their representation and manipulation is very efficient. Using feature trees to represent feature structures has some consequences, however:

[1]Cyclic constraints might be useful to encode the semantics of so called "liar" sentences like ''`This sentence is false`'' [Barwise and Etchemendy, 1988]. But we are not concerned with semantics, here.

- Constraints defining cyclic feature structures have to be disallowed. It is possible to detect cyclic feature constraints within a single rule at compile time, but it is not clear how this could be done for the grammar as a whole. A simple heuristic can be used in practice to detect the other cases at runtime. This heuristic limits the depth of feature structures to a fixed maximal value, e.g. 10. If a feature structure exceeds this limit, the parser will stop and report an error. This heuristic is not applicaple, of course, if the grammar is known to generate feature structures of arbitrary depth. But this is unlikely for an appropriately designed grammar (comp. section 2.6).

- It is not possible to distinguish whether two features values are incidentally the same or because some constraint enforces agreement. Some syntactic theories make use of this distinction in the analysis of reflexives. Here, it would be necessary to move the application of the respective constraints to semantic construction.

- It is not possible to compute a model for the union of two constraint sets simply by unifying the feature trees which form minimal models for the two constraint sets.

- Since feature trees cannot represent equality constraints, a problem arises in case of disjunctive feature values: If there are two features in a feature tree with the same disjunctive value, then we are only allowed to choose the values from the disjunction independently for both features if no equality constraint exists. However, since the equality information is missing in feature trees, there is no way to decide which combinations of feature values are valid.

The third point is related to the fact that feature trees are not logically complete. Feature tree (a) in figure 3.2, for example, is a minimal model for $\Phi_1 = \{\langle f \rangle \doteq \langle g \rangle\}$ and feature tree (b) is a minimal model for $\Phi_2 = \{\langle gf \rangle \doteq \langle gf \rangle\}$. Feature tree (c) results from unifying (a) and (b), but it is not a model for the union of constraints $\Phi = \Phi_1 \cup \Phi_2$ because (c) violates the constraint $\langle f \rangle \doteq \langle g \rangle$. The minimal feature tree model for Φ is shown in figure 3.2 as (d).

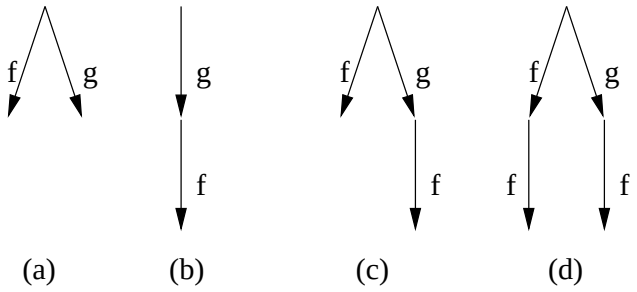

Figure 3.2: Sample feature trees

The problem arises because feature tree (a) fails to represent the constraint that the values of f and g have to be equal. Hence feature f is not updated together with g as it ought to be. Note however, that (c) subsumes (d), the minimal model for Φ. This is always true. If the tree unification had failed, then Φ would have been inconsistent. YAP exploits this property by computing a sequence of increasingly larger feature trees which subsume the minimal model for the constraints at a node and finally converge to this minimal model.

Feature trees will now be formally defined as partial functions from the set of labels to the set of feature structures. We follow the presentation in [Shieber, 1992]. From now on, the terms "feature structure" and "feature tree" will be used interchangeably.

Definition 6: The set $\mathcal{T}_{L,C}$ of *feature structures* is defined as $\mathcal{T}_{L,C} = \bigcup_{i \geq 0} \mathcal{T}_i$, where $\mathcal{T}_0 = C$ is the set of atomic feature structures, and $\mathcal{T}_i$ is the set of all partial functions from L to $\bigcup_{j<i} \mathcal{T}_j$. $\perp$ is the feature structure with $dom(f) = \emptyset$.

$F(f)$ is the value of feature f in feature structure F. This notation is extended to paths by defining $F(\langle f_1 \dots f_n \rangle) := F(f_1) \dots (f_n)$.

$\perp$ is the most general feature structure and contains no information.

Definition 7: A feature structure F *subsumes* another feature structure F', written $F \leq F'$, iff either

- $F = c$ and $F' = c$, with $c \in C$ or
- $\forall_{f \in dom(F)}\ f \in dom(F') \wedge F(f) \leq F'(f)$

Shieber calls this the *natural partial ordering*. It does not exactly match the definition of subsumption in definition 5 because there are feature trees F and F' such that $F \leq F'$, but $F \models \phi$ and $F' \not\models \phi$ for some constraint ϕ. For example, feature tree (a) in figure 3.2 subsumes feature tree (c), but feature tree (a) satisfies the constraint $\langle f \rangle \doteq \langle g \rangle$ whereas feature tree (c) does not.

Definition 8: The *unification* of two feature trees F and G (written $F \sqcup G$) is defined as follows:

- $F \sqcup G = F$, if $F \in C$ and $G \leq F$ (Note that $\perp \leq F$ according to def. 7.)
- $F \sqcup G = G$, if $G \in C$ and $F \leq G$
- $F \sqcup G = \top$, if either
  - $F \in C$ and $G \not\leq F$, or
  - $G \in C$ and $F \not\leq G$, or
  - $F(f) \sqcup G(f) = \top$ for some feature $f \in dom(F) \cup dom(G)$
- otherwise, $dom(F \sqcup G) = dom(F) \cup dom(G)$ and
  - $(F \sqcup G)(f) = F(f)$ if $f \notin dom(G)$
  - $(F \sqcup G)(f) = G(f)$ if $f \notin dom(F)$
  - $(F \sqcup G)(f) = F(f) \sqcup G(f)$ if $f \in dom(F) \cup dom(G)$

A feature structure is a model for a set of constraints if it satisfies those constraints. We will now define what it means when a feature structure is said to *satisfy* a constraint.

Definition 9: A feature structure F *satisfies* an equation $p \doteq v$ (written $F \models p \doteq v$) iff either

- v is a path $q \in Path$, and $F(p)$ as well as $F(q)$ are defined, and $F(p) = F(q)$ or
- v is a constant $c \in C$ and $F(p)$ is defined and $F(p) = c$.

A feature structure F satisfies a set of constraint equations Φ iff it satisfies each equation $\phi \in \Phi$.

Now, we define an operator which creates a new feature structure from a set of feature structures by linking the feature structures as feature values to the root node of the new feature structure.

Definition 10: The *embedding* $\bigsqcup_{f \in I} F_f \setminus f$ is the partial function F with domain $dom(F) = I$ and $F(f) = F_f$ for all $f \in I$.

If I contains only one element, the abbreviation $F_f \setminus f$ will be used.

This complex embedding operator is defined directly rather than in terms of an elementary embedding operator '$\setminus$' and a unification operator '$\sqcup$' to simplify some proofs.

Definition 11: A set of constraint equations Φ will be called a *description* of a feature structure F, written $desc(F)$, if $\Phi = \{p \doteq c \mid F(p) = c,\ c \in C\} \cup \{p \doteq p \mid F(p) \text{ is defined}\}$.

The description of a feature structure contains all constraints which are satisfied by the feature structure except for equality constraints. It is a comprehensive description of the information contained in the feature structure. (Remember that equality constraints are not represented in feature trees.)

Definition 12: The size of a feature tree F, $size(F)$, is recursively defined as follows:

$$size(F) = \begin{cases} 1 & \text{if } F \in C \\ \sum_{f \in dom(F)} size(F(f)) + 1 & \text{otherwise} \end{cases}$$

We will now show that the subsumption relation on feature trees is reflexive, antisymmetric and transitive, and that it is therefore a partial order.

Lemma 2: (REFLEXIVITY) For all feature structures F it is the case that $F \leq F$.

Proof: This follows from definition 7. □

Lemma 3: (ANTISYMMETRY) Feature structure F subsumes feature structure F' and F' subsumes F iff $F = F'$.

Proof:

'$\Leftarrow$': $F = F'$ entails $F \leq F'$ and $F' \leq F$ according to the definition of subsumption.

'$\Rightarrow$': $F \leq F'$ and $F' \leq F$ entails $F = F'$.

Induction hypothesis: Lemma is valid for feature structures from $\bigcup_{i=0}^{m} \mathcal{T}_i$ where $m < n$.

$\underline{F \in \mathcal{T}_0}$: F is an atom $c \in C$. From $F \in C$ and $F' \leq F$ follows $F' = c$ according to the definition of subsumption.

$\underline{F \in \mathcal{T}_n:}$ F is a partial function. From $F \leq F'$ follows according to the definition of subsumption that $\forall_{f \in dom(F)}\ f \in dom(F') \wedge F(f) \leq F'(f)$ holds. From $F' \leq F$ follows that $\forall_{f \in dom(F')}\ f \in dom(F) \wedge F'(f) \leq F(f)$ holds. Therefore $dom(F) = dom(F')$ and $\forall_{f \in dom(F)} F(f) \leq F'(f) \wedge F'(f) \leq F(f)$. With the induction hypothesis we can conclude that $\forall_{f \in dom(F)} F(f) = F'(f)$ and therefore $F = F'$. □

Lemma 4: (TRANSITIVITY) If $F \leq F'$ and $F' \leq F''$, then $F \leq F''$.

Proof: There are two cases. If F is a constant $c \in C$, then $F = F'$ and $F' = F''$ by definition 7. Otherwise, $F(f) \leq F'(f)$ and $F'(f) \leq F''(f)$ for all $f \in dom(F)$. $F(f) \leq F''(f)$ follows by induction over the length of the longest path in F. □

Now we prove some lemmata which will be needed later to prove the correctness of the parsing algorithm.

Lemma 5: If feature structure F subsumes feature structure F' and $F(p)$ with $p \in Path$ is defined then $F'(p)$ is defined and $F(p) \leq F'(p)$.

Proof by induction over the length of path p.

Lemma 6: If feature structure F does not subsume feature structure F', then a constraint equation $p \doteq v$ exists with $p \in Path$ and $v = p$ or $v \in C$ such that $F \models p \doteq v$ and $F' \not\models p \doteq v$.

Proof:

Induction hypothesis: Lemma is valid for feature structures from $\bigcup_{i=0}^{m} \mathcal{T}_i$ where $m < n$.

$\underline{F \in \mathcal{T}_0:}$ We choose $p := \langle\rangle$ and $v := F$ with $F \in C$ (remember $\mathcal{T}_0 = C$). $F \models p \doteq v$ follows because of $F(\langle\rangle) = F = v$ (see def. of feature structures) and from $F \not\leq F'$ follows that $F' \neq v$ and $F' \not\models \langle\rangle \doteq v$.

$\underline{F \in \mathcal{T}_n:}$ F is a partial function from L to $\bigcup_{i=0}^{n-1} \mathcal{T}_i$ and because of $F \not\leq F'$ there is a feature $f \in dom(F)$ with $f \notin dom(F')$ or $F(f) \not\leq F'(f)$. If $f \notin dom(F')$, then $\langle f \rangle \doteq \langle f \rangle$ is a constraint equation which is satisfied by F but not by F'.

Otherwise $F(f) \not\leq F'(f)$ and since $F(f) \in \bigcup_{i=0}^{n-1} \mathcal{T}_i$, there is an equation $p \doteq v$ with $p \in Path$ and $v = p$ or $v \in C$ such that $F(f) \models p \doteq v$ and $F'(f) \not\models p \doteq v$ (induction hypothesis).

If $v = p$, then $F(f)(p)$ exists, but $F'(f)(p)$ does not. Therefore $F \models \langle f \rangle \cdot p \doteq \langle f \rangle \cdot p$ and $F' \not\models \langle f \rangle \cdot p \doteq \langle f \rangle \cdot p$.

If $v \in C$, then $F(f)(p) = v$ and $F \models \langle f \rangle \cdot p \doteq v$. Because of $F'(f) \not\models p \doteq v$ we conclude that $F'(f)(p) \neq v$ and therefore $F' \not\models \langle f \rangle \cdot p \doteq v$. □

Lemma 7: F subsumes F' iff $F' \models desc(F)$.

Proof:

'⇒': If F subsumes F' then $F' \models desc(F)$

Otherwise there are feature structures F, F' with $F \leq F'$ and an equation $p \doteq v \in desc(F)$ such that $F' \not\models p \doteq v$. According to the definition of *desc*, v is either equal to p or a constant from C. Since $p \doteq v \in desc(F)$, F must satisfy $p \doteq v$ and therefore $F(p)$ must be defined.

From $F \leq F'$ and lemma 5, it follows that $F'(p)$ is defined as well. However, if $v = p$ then $F' \models p \doteq v$ in contrast to the above assumption. Therefore v must be a constant from C and $F(p) = v$. However, since F subsumes F' and $F(p) = v$, it follows that $F'(p) = v$ and therefore $F' \models p \doteq v$, again in contrast to the above assumption.

'$\Leftarrow$': If $F' \models desc(F)$ then F subsumes F'

Otherwise there are feature structures F, F' such that $F' \models desc(F)$ and $F \not\leq F'$. From $F \not\leq F'$ follows according to lemma 6 that a constraint $p \doteq v$ with $v = p$ or $v \in C$ exists such that $F \models p \doteq v$ and $F' \not\models p \doteq v$. However, it follows that $p \doteq v \in desc(F)$ and therefore $F' \not\models desc(F)$ in contrast to the initial assumption. □

Lemma 8: $F(p) \models \Phi$ iff $F \models \Phi[\langle\rangle \rightarrow p]$ where $\Phi[p \rightarrow q]$ denotes the replacement of all paths in Φ of the form $p \cdot r$ with $q \cdot r$.

Proof: see [Shieber, 1992] page 89.

Lemma 9: $F \models \Phi$ iff $F \setminus f \models \Phi[\langle\rangle \rightarrow \langle f \rangle]$.

Proof: Replacing F with $F \setminus f$ and p with $\langle f \rangle$ in lemma 8, we get $(F \setminus f)(\langle f \rangle) \models \Phi$ iff $F \setminus f \models \Phi[\langle\rangle \rightarrow \langle f \rangle]$. From the definition of embedding follows $(F \setminus f)(\langle f \rangle) = F$. □

Lemma 10: If feature structure F is defined, then a feature path p exists such that $F(p) = v$ with $v \in C \cup \{\bot\}$.

Proof:

Induction hypothesis: Lemma is valid for feature structures from $\bigcup_{i=0}^{m} \mathcal{T}_i$ where $m < n$.

$\underline{F \in \mathcal{T}_0:}$ In this case, $F = c$ with $c \in C$ and we choose $p := \langle\rangle$ so that $F(p) = F(\langle\rangle) = F = c$.

$\underline{F \in \mathcal{T}_n:}$ Here F is a partial function from L to $\bigcup_{i=0}^{n-1} \mathcal{T}_i$. If $dom(F) = \emptyset$, then we choose $p := \langle\rangle$ so that $F(p) = F(\langle\rangle) = \bot$ holds. Otherwise, there is a feature $f \in dom(F)$ such that $F(f)$ is defined. According to the induction hypothesis, there is a feature path r such that $F(f)(r) = c'$ for some $c' \in C \cup \{\bot\}$. In this case, $F(p) = c$ holds for $p := \langle f \rangle \cdot r$ and $c := c'$. □

Lemma 11: (LOGICAL SOUNDNESS) For all $F \in \mathcal{T}_{L,C}$ and all sets of equations Φ, if $F \models \Phi$ and $\Phi \vdash \phi$ then $F \models \phi$.

Proof: see [Shieber, 1992] page 90.

Lemma 12: The size of a feature structure F, $size(F)$, is always finite.

Proof by induction over the depth of F.

Lemma 13: If feature structure F subsumes feature structure F', then $size(F) \leq size(F')$.

Proof:

Induction hypothesis: Lemma holds for feature structures in $\bigcup_{i=0}^{m} \mathcal{T}_i$ where $m < n$.

$\underline{F \in \mathcal{T}_0:}$ In this case, $F = c$ with $c \in C$ and, according to the definition of subsumption $F' = c$. So, $size(F) = 1$ and $size(F') = 1$ and therefore $size(F) \leq size(F')$.

$\underline{F \in \mathcal{T}_n}$: In this case, $size(F) = \sum_{f \in dom(F)} size(F(f))$. Now $F(f) \leq F'(f)$ holds for all $f \in dom(F)$ by definition of subsumption, and because $F(f) \in \bigcup_{i=0}^{n-1} \mathcal{T}_i$ it follows that $size(F(f)) \leq size(F'(f))$ holds for all $f \in dom(F)$ (induction hypothesis). It follows that $size(F) \leq \sum_{f \in dom(F)} size(F'(f))$ and because the size function is always non-negative $size(F) \leq \sum_{f \in dom(F')} size(F'(f)) = size(F')$. □

Definition 13: Feature structure $F = tm(\Phi)$, the *tree model* of a finite and consistent set of acyclic constraint equations Φ, is defined as follows:

1. $F(p) = c$ with $c \in C$ iff $\Phi \vdash p \doteq c$
2. $F(p) = \bot$ iff $\Phi \vdash p \doteq p$ and $\forall_{r \in Path} \Phi \vdash p \cdot r \doteq v \;\Rightarrow\; r = \langle\rangle \wedge v \notin C$
3. $f \in dom(F(p))$ and $F(p)(f) = F(p \cdot \langle f \rangle)$ iff $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$.

Lemma 14: If $F = tm(\Phi)$ and $F(p)$ is defined then $\Phi \vdash p \doteq p$.

Proof:

According to definition 13, there is a constraint ϕ of the form $p \doteq c$ or $p \doteq p$ or $p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ such that $\Phi \vdash \phi$. $\Phi \vdash p \doteq p$ follows immediately by the reflexivity rule. □

Because Φ is a finite set of acyclic constraints and the number of feature labels is finite, it follows that the tree models have finite size.

Proof:

For each node in F, there is a constraint ϕ of the form $p \doteq c$ or $p \doteq p$ or $p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ such that $\Phi \vdash \phi$ (definition 13). Because Φ is acyclic, there is a constant N such that $|p| < N$ for all feature paths (lemma 1). Therefore, if $F(p)$ is defined then $|p| < N$. □

Lemma 15: (DENOTATIONAL COMPLETENESS) For all sets of constraints Φ, if Φ is consistent, then there is a $F \in \mathcal{T}_{LC}$ such that $F \models \Phi$.

Proof: (Compare Shieber's proof for the denotational completeness of infinite trees in [Shieber, 1992], page 94.)

We prove that $F = tm(\Phi)$ as defined in definition 13 is a feature tree and satisfies Φ.

F is a function: Suppose F were not a function, that is F assigned two distinct values G and G' to a single path p. If $G, G' \in C$, then $\Phi \vdash p \doteq G$ and $\Phi \nvdash p \doteq G'$, so Φ is inconsistent, contra assumption. If $G \in C$ and $G' = \bot$, then $\Phi \vdash p \doteq G$ and $\Phi \nvdash p \doteq G$, again a contradiction.

If $G \in C$ and $G' \notin C \cup \{\bot\}$, then $\Phi \vdash p \doteq G$ and $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ for some feature f. But then Φ is inconsistent due to a constant/compound clash.

If $G = \bot$ and $G' \notin C \cup \{\bot\}$, then $\Phi \nvdash p \cdot r \doteq v$ for non-empty r and $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$, a contradiction.

Finally, we have to consider the case where $G \notin C \cup \{\bot\}$ and $G' \notin C \cup \{\bot\}$. When $F(p)$ is defined and $F(p) \notin C \cup \{\bot\}$ then $F(p)$ is a function whose range and feature values are uniquely defined by the third point of def. 13. Hence, g and G' must be identical.

F is prefix-closed: This follows from the definition of tm and the reflexivity rule.

F **satisfies** Φ: Any constraint equation $\phi \in \Phi$ is of one of two forms. If $\phi = p \doteq c$, then $F(p) = c$ by definition of tm and $F \models \phi$.

If $\phi = p \doteq q$, we must show that $F(p)$ and $F(q)$ are defined and that $F(p) = F(q)$.

1. If $\Phi \vdash p \doteq c$ for some $c \in C$, then $\Phi \vdash q \doteq c$ follows immediately and by definition of tm, $F(p)$ and $F(q)$ are defined and $F(p) = F(q) = c$.

2. From $\Phi \vdash p \doteq q$ follows $\Phi \vdash p \doteq p$ and $\Phi \vdash q \doteq q$. If $(\Phi \vdash p \cdot r \doteq v) \Rightarrow (r = \langle\rangle \wedge v \notin C)$, then $(\Phi \vdash q \cdot r \doteq v) \Rightarrow (r = \langle\rangle \wedge v \notin C)$ holds because otherwise $\Phi \vdash p \cdot r \doteq v$ would follow for some non-empty path r, a contradiction to the assumption that $(\Phi \vdash p \cdot r \doteq v) \Rightarrow (r = \langle\rangle \wedge v \notin C)$. Therefore $F(p)$ and $F(q)$ are defined by definition of tm and $F(p) = F(q) = \bot$.

3. None of the previous cases applies. Then there must be a feature f and a feature path r such that $\Phi \vdash p \cdot \langle f \rangle \cdot r \doteq v$ (follows from def. 13). $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ follows by the reflexivity rule. From $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ follows that $F(p)$ is defined and that $f \in dom(F(p))$. The same follows for q. So $F(p)$ as well as $F(q)$ is defined and their domains are equal. Otherwise, there is a feature f which is in only one of the domains. Without loss of generality we assume $f \in dom(F(p))$. According to the def. of tm, $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ and by the substitutivity rule $\Phi \vdash q \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ and by the reflexivity rule $\Phi \vdash q \cdot \langle f \rangle \doteq q \cdot \langle f \rangle$ follows. But then $F(q)(f)$ is defined as well because $F \models \Phi$. We know that $\Phi \vdash p \cdot \langle f \rangle \doteq q \cdot \langle f \rangle$ holds. By induction over the height of the feature trees, it is easy to prove that $F(p \cdot \langle f \rangle) = F(q \cdot \langle f \rangle)$.

Lemma 16: (EXISTENCE OF MINIMAL MODELS) If Φ is a finite, consistent, acyclic set of constraints, then a feature structure F, called *minimal model* of Φ, exists such that $F \models \Phi$ and for all F' such that $F' \models \Phi$, it is the case that $F \leq F'$.

Proof:

According to lemma 15, there is a feature structure $F = tm(\Phi)$ such that $F \models \Phi$.

F is the unique minimal model of Φ. Otherwise, another feature tree F' existed such that $F' \models \Phi$ and $F \not\leq F'$. From $F \not\leq F'$ follows according to lemma 6 that a constraint equation $p \doteq v$ with $v = p$ or $v \in C$ must exist such that $F \models p \doteq v$ and $F' \not\models p \doteq v$. Because $F' \models \Phi$ and because feature trees are logically sound (lemma 11), it follows that $\Phi \not\vdash p \doteq v$.

However, if $v \in C$, then $F(p) = v$ follows by definition of constraint satisfaction and $\Phi \vdash p \doteq v$ follows by definition of tm which is a contradiction to $\Phi \not\vdash p \doteq v$.

Otherwise, if $v = p$ holds, then $F \models p \doteq p$ and $F' \not\models p \doteq p$ and $\Phi \not\vdash p \doteq p$. It follows from $F \models p \doteq p$ and the definition of constraint satisfaction that $F(p)$ is defined. This entails by definition of tm that $\Phi \vdash p \doteq c$ or $\Phi \vdash p \doteq p$ or $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$ for some feature f. In all three cases, $\Phi \vdash p \doteq p$ follows immediately, a contradiction to $\Phi \not\vdash p \doteq p$. Hence F is indeed the unique minimal model of Φ. □

The minimal model of Φ is written $MM(\Phi)$.

Lemma 17: Inconsistent constraints have no feature tree model.

Proof:

Assume there is an inconsistent set of constraints Φ and a feature structure F which models Φ. Because Φ is inconsistent it is either possible to derive two constraints $p \doteq c$ and $p \doteq c'$ where $c, c' \in C$ and $c \neq c'$ or two constraints $p \doteq c$ and $p \cdot r \doteq v$ where $c \in C$ and r a non-empty path. In the first case, $F(p) = c$ and $F(p) = c'$ follows from the definition of constraint satisfaction which is impossible. In the latter case, $F(p)$ is a constant and a function at the same time, which is impossible according to the definition of feature trees. □

Lemma 18: If F and G are feature structures and if $F \sqcup G \neq \top$, then $F \sqcup G = MM(desc(F) \cup desc(G))$ holds.

Proof:

Looking at the definitions of unification (definition 8) and subsumption (definition 7), it is obvious that $F \leq F \sqcup G$ and $G \leq F \sqcup G$ hold. It follows with lemma 7 that $F \sqcup G \models desc(F)$ and $F \sqcup G \models desc(G)$. Because of the logical soundness of feature trees, $F \sqcup G$ also satisfies all formulas which are entailed by $desc(F) \cup desc(G)$. Therefore $F \sqcup G$ is a model for $desc(F) \cup desc(G)$.

If $F \sqcup G$ were not the minimal model for $desc(F) \cup desc(G)$, then another model F' existed such that $F \sqcup G \not\leq F'$. In this case, a constraint $p \doteq v$ with $p \in Path$ and either $v = p$ or $v \in C$ exists, according to lemma 6, such that $F \sqcup G \models p \doteq v$ and $F' \not\models p \doteq v$. If $v \in C$, then $(F \sqcup G)(p) = v$ and either $F(p) = v$ or $G(p) = v$. in this case, $p \doteq v \in desc(F) \cup desc(G)$ and therefore $F'(p) = v$.

If $v = p$, then $F \sqcup G \models p \doteq p$ and therefore $(F \sqcup G)(p)$ is defined. From the definition of unification follows that $(F \sqcup G)(p)$ is only defined, if either $F(p)$ or $G(p)$ is defined. Therefore $p \doteq p \in desc(F) \cup desc(G)$ holds. We conclude that no such feature structure F' exists and that $F \sqcup G$ is the minimal model for $desc(F) \cup desc(G)$. □

3.3 Parse Trees

After defining the feature structures, we will now formally define grammar rules and parse trees.

Definition 14: A *grammar rule* is a pair $\langle k, \Phi \rangle$, where k is the arity of the rule and Φ is a set of constraint equations. All feature paths in Φ start with a label $l \in \{0, 1, \ldots k\}$ and $\langle i \rangle \doteq \langle i \rangle \in \Phi$ for all $0 \leq i \leq k$.

Definition 15: A *lexical rule* is a pair $\langle w, \Phi \rangle$, where w is a word and Φ is a set of constraint equations. All feature paths in Φ start with the label 0 and $\langle 0 \rangle \doteq \langle 0 \rangle \in \Phi$.

Definition 16: A *terminal parse tree node* n is a triple $\langle i, F, r \rangle$, where $i \in \mathcal{N}$ is an index, F is a feature structure and r is a lexical rule.

i is called the index of node n. F is called the feature structure of n.

Definition 17: A *nonterminal parse tree node* is a quadruple $\langle i, F, r, D\rangle$, where $i \in \mathcal{N}$ is an index, F is a feature structure, $r = \langle k, \Phi\rangle$ is a grammar rule and $D = \langle d_1 \dots d_k\rangle$ is the list of indices of the daughter nodes.

i is called the index of node n. F is called the feature structure of n.

Definition 18: A *parse tree node* is either a terminal parse tree node or a nonterminal parse tree node.

Definition 19: A parse tree node n *immediately dominates* another node n' iff n is a nonterminal node $\langle i, F, \langle k, \Phi\rangle, \langle d_1, d_2, \dots, d_k\rangle\rangle$ and there is a j with $0 < j \leq k$ such that d_j is the index of n'.

Definition 20: A parse tree node n *dominates* another node n' iff $n = n'$ or there is a node n'' such that n immediately dominates n'' and n'' dominates n'.

Definition 21: A *parse tree* is a pair $\langle PN, k\rangle$, where PN is a set of parse tree nodes and $k \in \mathcal{N}$ is the index of a node $n \in PN$ called root node and PN is the set of nodes dominated by n and n is not dominated by any node and any other node from NP is dominated by exactly one node.

Definition 22: A *valid parse tree* is a parse tree $\langle PN, k\rangle$ in which

- for each terminal node $\langle i, F, \langle w, \Phi\rangle\rangle \in PN$ it is the case that $F \setminus 0 \models \Phi$ and
- for each nonterminal node $\langle i, F_0, \langle k, \Phi\rangle, \langle d_1, \dots, d_k\rangle\rangle \in PN$ and its daughter nodes $\langle d_j, F_j, \dots\rangle \in PN$ with $0 < j \leq k$ it is the case that $\bigsqcup_{i=0}^{k} F_i \setminus i \models \Phi$.

3.4 Feature Computation

We will now present an algorithm which computes the features trees when the context-free parse tree is given. This algorithm is a simplification of the algorithm which is used in YAP (see section 4.3.3). The following simplifications have been made:

1. unambiguous parse trees as input (rather than "parse forests")
2. no feature typing
3. no disjunctive feature values
4. no feature operators (like string concatenation in the YAP formalism)

This algorithm will be used to prove that iterative computation of feature structures with a tree representation indeed returns the correct result.

3.4.1 The Algorithm

The algorithm is presented as a set of inference rules. It is assumed that the context-free parse tree is given as a set of tuples of the form $\langle i, \langle X \rightarrow w, \Phi\rangle\rangle$ in case of terminal nodes and the form $\langle i, \langle X \rightarrow Y_1 \ldots Y_k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle$ in case of nonterminal nodes, where i is a unique index which identifies the node and $\langle X \rightarrow Y_1 \ldots Y_k, \Phi\rangle$ is a grammar rule with a context-free backbone $X \rightarrow Y_1 \ldots Y_k$ and a set of feature constraints Φ, and d_1 through d_k are the indices of the daughter nodes. $\langle X \rightarrow w, \Phi\rangle$ is a lexicon entry with feature constraints. It is further assumed that the root node of the parse tree has index 0.

The feature computation algorithm uses the following auxiliary definition.

Definition 23: The *most general extension* $MGE(\Phi, i, T_1, \ldots, T_k)$ of a feature tree $T_i, 1 \leq i \leq k$ with respect to a set of feature constraints Φ and a set of other feature trees $\{T_j | 1 \leq j \neq i \leq k\}$ is defined as:

$$MGE(\Phi, i, T_1, \ldots, T_k) = \begin{cases} MM(\bigcup_{j=0}^{k} desc(T_j)[\langle\rangle \rightarrow \langle j\rangle] \ \cup \ \Phi)) \ (i) & \text{if MM(...) exists} \\ \top & \text{otherwise} \end{cases} \tag{3.1}$$

The most general extension operation extends a set of features structures (by adding new features or by making feature values more specific) such that the local constraints of a rule or lexical entry are satisfied.

Initialisation

$$\frac{\langle i, \langle X \rightarrow w, \Phi\rangle\rangle}{\langle 0, \langle i, \bot, \langle w, \Phi\rangle\rangle\rangle} \qquad \frac{\langle i, \langle X \rightarrow Y_1 \ldots Y_k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle}{\langle 0, \langle i, \bot, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle} \tag{3.2}$$

Bottom-Up

$$\frac{\langle j, \langle i, T, \langle w, \Phi\rangle\rangle\rangle}{\langle j+1, \langle i, MGE(\Phi, 0, T), \langle w, \Phi\rangle\rangle\rangle} \quad j \text{ even} \tag{3.3}$$

$$\frac{\langle j, \langle i, T_0, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle \quad \langle j+1, \langle d_1, T_1, \ldots\rangle\rangle \ \ldots \ \langle j+1 \langle d_k, T_k, \ldots\rangle\rangle}{\langle j+1, \langle i, MGE(\Phi, 0, T_0, T_1, \ldots, T_k), \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle} \quad j \text{ even} \tag{3.4}$$

Top-Down

$$\frac{\langle j, \langle 0, T, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle}{\langle j+1, \langle 0, T, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle} \quad j \text{ odd} \tag{3.5}$$

$$\frac{\langle j+1, \langle i, T_0, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle\rangle \quad \langle j, \langle d_1, T_1, \ldots\rangle\rangle \ \ldots \ \langle j, \langle d_k, T_k, \ldots\rangle\rangle}{\langle j+1, \langle d_1, MGE(\Phi, 1, T_0, \ldots, T_k), \ldots\rangle\rangle \ldots \langle j+1, \langle d_k, MGE(\Phi, k, T_0, \ldots, T_k), \ldots\rangle\rangle} \quad j \text{ odd} \tag{3.6}$$

Termination Criterion

Failure:

$$\exists_i \quad \langle j, \langle i, \top, \langle w, \Phi \rangle \rangle \rangle \tag{3.7}$$

$$\exists_i \quad \langle j, \langle i, \top, \langle k, \Phi \rangle, \langle d_1, \ldots, d_k \rangle \rangle \rangle \tag{3.8}$$

Success:

$$\forall_i \quad \langle j, \langle i, T, \langle w, \Phi \rangle \rangle \rangle \Rightarrow \langle j+1, \langle i, T, \langle w, \Phi \rangle \rangle \rangle \tag{3.9}$$

$$\forall_i \quad \langle j, \langle i, T, \langle k, \Phi \rangle, \langle d_1, \ldots, d_k \rangle \rangle \rangle \Rightarrow \langle j+1, \langle i, T, \langle k, \Phi \rangle, \langle d_1, \ldots, d_k \rangle \rangle \rangle \tag{3.10}$$

The initialisation step sets the feature structures to $\perp$ and initialises the generation counters to 0.

The bottom-up step updates the feature structures from bottom to top. The first rule applies to terminal nodes where the constraints of the lexical entry have to be satisfied. The second rule applies to non-terminal nodes whose feature structures are updated to reflect changes in the daughter node feature structures. The MGE operation also computes new values for the daughter node feature structures. But these are not retained.

The top-down step updates the feature structures from top to bottom. The feature structure of the root node (index 0) is just copied. The other feature structures are updated to reflect changes in the feature structures of the mother node and the sister nodes. The MGE operation also computes a new feature structure for the mother node which is not retained.

The algorithm terminates if one of the constraints cannot be satisfied or if no feature structure changed in the last iteration.

Note that the feature structure for a particular node and generation is not guaranteed to be unique if the input structure is a graph rather than a tree. A slightly different algorithm ist therefore needed to compute the feature structures of parse forests.

To show the correctness of the algorithm, we prove first that any valid solution to the parsing problem is subsumed by all intermediate results of the algorithm.

Lemma 19: If $\langle \{n_j\}, k \rangle$ is a valid parse tree with $n_j = \langle j, F_j, r_j, D_j \rangle$ and if $\langle \{n_j^t\}, k \rangle$ with $n_j^t = \langle j, F_j^t, r_j, D_j \rangle$ is the parse tree resulting after t iterations of the algorithm, then $\forall_i \; F_i^t \leq F_i$.

Proof:

Induction hypothesis: $F_i^s \leq F_i$ for all i and $s < t$

$\underline{t = 0:}$ $F_i^0 = \perp \; \leq F_i$ for all i

$\underline{t > 0:}$ Three cases have to be considered:

1. F_i^t is the feature structure of the root node in a top-down pass and hence unchanged. $F_i^t = F_i^{t-1} \leq F_i$ (induction hypothesis)

2. F_i^t is the feature structure of a terminal node $n_i^t = \langle i, F_i^t, \langle w, \Phi \rangle \rangle$ resulting after a bottom-up pass of the algorithm and $F_i^t = MM(desc(F_i^{t-1})[\langle\rangle \rightarrow \langle 0 \rangle] \cup \Phi))$ (0).

According to lemma 7, $F_i^{t-1} \leq F_i$ (induction hypothesis) implies that $F_i \models desc(F_i^{t-1})$. From lemma 9 it follows that $F_i \setminus 0 \models desc(F_i^{t-1}[\langle\rangle \rightarrow \langle 0\rangle])$. From the definition of valid parse trees follows that $F_i \setminus 0 \models \Phi$. From $F_i \setminus 0 \models desc(F_i^{t-1}[\langle\rangle \rightarrow \langle 0\rangle]) \cup \Phi$ follows $MM(desc(F_i^{t-1}[\langle\rangle \rightarrow \langle 0\rangle]) \cup \Phi) \leq F_i \setminus 0$ and therefore $F_i^t \leq F_i$ (lemma 5).

3. F_i^t is the feature structure of some other node which was updated either during bottom-up or during top-down parsing. In other words, there is a non-terminal node $n_{d_0}^{t-1} = \langle d_0, F_{d_0}^{t-1}, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle$ and a set of daughter nodes $n_{d_l}^{t-1} = \langle d_l, F_{d_l}^{t-1}, \ldots\rangle$ with $0 < l \leq k$ such that $i = d_j$ for some j in $0 \leq j \leq k$. $F_i^{t-1} = F_{d_j}^{t-1}$ has been replaced with $F_i^t = MM(\bigcup_{l=0}^{k} desc(F_{d_l})[\langle\rangle \rightarrow \langle l\rangle] \cup \Phi))$ (j).

   According to lemma 7, $F_i^{t-1} \leq F_i$ (induction hypothesis) implies that $F_i \models desc(F_i^{t-1})$ and according to lemma 9 it follows that $F_i \setminus l \models desc(F_i^{t-1}[\langle\rangle \rightarrow \langle l\rangle])$. From the definition of multiple embedding follows that $\bigsqcup_{l=0}^{k} F_{d_l} \setminus l \models desc(F_{d_m}^{t-1}[\langle\rangle \rightarrow \langle m\rangle])$ for all $0 \leq m \leq k$. From the definition of valid parse trees further follows that $\bigsqcup_{l=0}^{k} F_{d_l} \setminus l \models \Phi$. Therefore $\bigsqcup_{l=0}^{k} F_{d_l} \setminus l \models \bigcup_{l=0}^{k} desc(F_{d_l})[\langle\rangle \rightarrow \langle l\rangle] \cup \Phi$ holds which entails $MM(\bigcup_{l=0}^{k} desc(F_{d_l})[\langle\rangle \rightarrow \langle l\rangle] \cup \Phi) \leq \bigsqcup_{l=0}^{k} F_{d_l} \setminus l$ and therefore $F_i^t \leq F_i$ (lemma 5). □

The local rule constraints which are associated with the nodes in a parse tree together form the constraint system which the parser has to solve. We now define a translation of this system of local constraints into an equivalent global constraint system Ψ. The translation replaces the relative indices (0 for the mother node, 1 for the first daughter etc.) in the constraint formulas with the absolute indices of the respective feature structures. The translated constraint system has the advantage that its satisfiability is defined without reference to the parse tree.

Lemma 20: Parse tree $\langle\{n_i\}, k\rangle$ with $n_i = \langle i, F_i, \ldots\rangle$ is a valid parse tree iff $G = \bigsqcup_i F_i \setminus i$ satisfies the constraint set Ψ which is defined as follows:

1. For each terminal node $n_i = \langle i, F_i, \langle w, \Phi\rangle\rangle$ of the parse tree and for each $\phi \in \Phi$, if $\phi = \langle 0\rangle{\cdot}p \doteq \langle 0\rangle{\cdot}q$, then $\phi' = \langle i\rangle{\cdot}p \doteq \langle i\rangle{\cdot}q \in \Psi$ else if $\phi = \langle 0\rangle{\cdot}p \doteq c$, then $\phi' = \langle i\rangle{\cdot}p \doteq c \in \Psi$. Label i will be called the translation of label 0 at n_i, written $i = itr_i(0)$.

2. For each non-terminal node $n_{d_0} = \langle d_0, F_{d_0}, \langle k, \Phi\rangle, \langle d_1, \ldots, d_k\rangle\rangle$ with a set of daughter nodes $\langle d_l, F_{d_l}^{t-1}, \ldots\rangle$ with $0 < l \leq k$ and for each $\phi \in \Phi$, if $\phi = \langle i\rangle \cdot p \doteq \langle k\rangle \cdot q$, then $\phi' = \langle d_i\rangle \cdot p \doteq \langle d_k\rangle \cdot q \in \Psi$ else if $\phi = \langle i\rangle \cdot p \doteq c$, then $\phi' = \langle d_i\rangle \cdot p \doteq c \in \Psi$. Label d_l will be called the translation of label l at n_i, written $d_l = itr_i(l)$.

ϕ' is called the *translation* of ϕ at n_i, written $\phi' = tr_i(\phi)$. This notation is extended to constraint sets by defining $tr_i(\Phi) = \{tr_i(\phi) | \phi \in \Phi\}$.

Proof:

Since the definition of G assigns a unique value F_i to each feature $i \in dom(G)$ and since F_i is well-defined, G is a well-defined function as well.

For each constraint ϕ in the parse tree, there is by definition of Ψ exactly one corresponding constraint ψ in Ψ.

'⇒': We prove that G satisfies ψ if the parse tree satisfies ϕ.

For each non-terminal node $\langle d_0, F_{d_0}, \langle k, \Phi\rangle, \langle d_1 \dots d_k\rangle\rangle$ in the parse tree with daughter nodes $\langle d_j, F_{d_j}, \dots\rangle$ for $0 < j \leq k$, it is the case that $\bigsqcup_{j=0}^{k} F_{d_j} \setminus j \models \Phi$. Similarly, it is for each terminal node $\langle d_0, F_{d_0}, \langle w, \Phi\rangle\rangle$ the case, that $\bigsqcup_{j=0}^{k} F_{d_j} \setminus j \models \Phi$ where $k = 0$. In both cases, an equation $\phi \in \Phi$ is either of the form $\langle j\rangle \cdot p \doteq \langle l\rangle \cdot q$ or $\langle j\rangle \cdot p \doteq c$.

If ϕ is of the first form, then $\bigsqcup_{m=0}^{k} F_{d_m} \setminus m \models \langle j\rangle \cdot p \doteq \langle l\rangle \cdot q$ because the parse tree satisfies Φ. Therefore $F_{d_j}(p) = F_{d_l}(q)$ and, according to the definition of G, $G(d_j)(p) = G(d_l)(q)$. From $\langle j\rangle \cdot p \doteq \langle l\rangle \cdot q \in \Phi$ follows according to the definition of Ψ that $\psi = \langle d_j\rangle \cdot p \doteq \langle d_l\rangle \cdot q$ is the corresponding equation of ϕ in Ψ and that $G \models \psi$.

It is easy to prove with similar arguments that if ϕ is of the form $\langle j\rangle \cdot p \doteq c$, then $G \models \psi$ for the translated constraint ψ.

'$\Leftarrow$': We prove that the parse tree satisfies ϕ if G satisfies $\psi = tr_i(\phi)$, the translation of ϕ at some node n_i.

According to the definition of tr_i, there is either a terminal node $\langle d_0, F_{d_0}, \langle w, \Phi\rangle\rangle$ or a non-terminal node $\langle d_0, F_{d_0}, \langle k, \Phi\rangle\langle d_1 \dots d_k\rangle\rangle$ such that for some $\phi \in \Phi$, $\psi = tr_i(\phi)$.

If $\psi = \langle j\rangle \cdot p \doteq \langle l\rangle \cdot q$, then $j = d_m$ for some $0 \leq m \leq k$ and $l = d_n$ for some $0 \leq n \leq k$. Hence $\phi = \langle m\rangle \cdot p \doteq \langle n\rangle \cdot q$.

Since $G \models \psi$, it is the case that $G(j)(p) = G(l)(q)$ and therefore $F_{d_m}(p) = F_{d_n}(q)$ and $\bigsqcup_{s=0}^{k} F_{d_s} \setminus s \models \langle m\rangle \cdot p \doteq \langle n\rangle \cdot q$. Therefore the parse tree satisfies ϕ.

It is easy to prove with analogous arguments that if ψ is of the form $\langle j\rangle \cdot p \doteq c$ then $\bigsqcup_{s=0}^{k} F_{d_s} \setminus s \models \phi$ for the corresponding ϕ with $\psi = tr_i(\phi)$. □

Lemma 21: If $\Phi \vdash \phi$ and $tr_i(\Phi)$ and $tr_i(\phi)$ are defined, then $tr_i(\Phi) \vdash tr_i(\phi)$.

Proof:

It is possible to put an order on the constraints which are deducible from Φ such that $\Phi \vdash_1 \phi_1$ and $\Phi \cup \{\phi_1\} \vdash_1 \phi_2$ and $\Phi \cup \{\phi_1, \phi_2\} \vdash_1 \phi_3$ and so on, where $\Phi \vdash_1 \phi$ means that ϕ follows from Φ by application of a single deduction rule. We define $\Phi_n = \Phi \cup \{\phi_1, \dots, \phi_n\}$ for $n \geq 0$.

Induction hypothesis: $\Psi \vdash tr(\phi)$ holds for all $\phi \in \Phi_m$ where $tr(\phi)$ is defined if $m < n$. ϕ is of the form $\langle\rangle \doteq \langle\rangle$ or the form $\langle i\rangle \cdot p \doteq \langle j\rangle \cdot q$ or the form $\langle i\rangle \cdot p \doteq c$ where $c \in C$.

<u>$n = 0$:</u> From $\phi \in \Phi_0$ and $\Phi_0 = \Phi$, follows $tr(\phi) \in tr(\Phi) = \Psi$ and therefore $\Psi \vdash tr(\phi)$.

<u>$n > 0$:</u> If $\phi \in \Phi_n$, then either $\phi \in \Phi_{n-1}$ and therefore $\Psi \vdash tr(\phi)$ by the induction hypothesis, or ϕ follows from Φ_{n-1} by one of the following inference rules:

1. Triviality: $\Phi \vdash \langle\rangle \doteq \langle\rangle$.

   Since $tr(\langle\rangle \doteq \langle\rangle)$ is undefined, there is nothing to prove. Furthermore, $\langle\rangle \doteq \langle\rangle$ cannot be used to derive any new constraints. Hence we can ignore it.

2. Symmetry: $\langle i\rangle \cdot p \doteq \langle j\rangle \cdot q$ followed from $\langle j\rangle \cdot q \doteq \langle i\rangle \cdot p \in \Phi_{n-1}$.

   According to the induction hypothesis, $\Psi \vdash \langle d_j\rangle \cdot q \doteq \langle d_i\rangle \cdot p$ holds if $d_i = itr(i)$ and $d_j = itr(d_j)$. It follows that $\Psi \vdash \langle d_i\rangle \cdot p \doteq \langle d_j\rangle \cdot q$ and therefore $\Psi \vdash tr(\langle i\rangle \cdot p \doteq \langle j\rangle \cdot q)$.

3. Reflexivity: $\langle i\rangle \cdot p \doteq \langle i\rangle \cdot p$ followed from $\langle i\rangle \cdot p \cdot r \doteq v \in \Phi_{n-1}$.

   According to the induction hypothesis, $\Psi \vdash \langle d_i\rangle \cdot p \cdot r \doteq v'$ holds where $d_i = itr(i)$ and $v' = v$ if $v \in C$, and $v' = \langle d_j\rangle \cdot q$ if $v = \langle j\rangle \cdot q$ and $d_j = itr(j)$. From $\Psi \vdash \langle d_i\rangle \cdot p \cdot r \doteq v'$ follows $\Psi \vdash \langle d_i\rangle \cdot p \doteq \langle d_i\rangle \cdot p$ and therefore $\Psi \vdash tr(\langle i\rangle \cdot p \doteq \langle i\rangle \cdot p)$.

   $\langle\rangle \doteq \langle\rangle$ follows from $\langle i\rangle \cdot p \cdot r \doteq v \in \Phi_{n-1}$ as well, but we can ignore this case because the resulting constraint is the same as that generated by the Triviality rule.

4. Substitutivity: From $\langle i\rangle \cdot p \cdot r \doteq v \in \Phi_{n-1}$ and $\langle i\rangle \cdot p \doteq \langle j\rangle \cdot q \in \Phi_{n-1}$ followed $\langle j\rangle \cdot q \cdot r \doteq v$.

   From $\langle i\rangle \cdot p \cdot r \doteq v \in \Phi_{n-1}$ follows $\Psi \vdash \langle d_i\rangle \cdot p \cdot r \doteq v'$ with $d_i = itr(i)$ and $v' = v$ if $v \in C$ and $v' = \langle d_l\rangle \cdot q$ if $v = \langle l\rangle \cdot q$ and $d_l = itr(l)$.

   From $\langle i\rangle \cdot p \doteq \langle j\rangle \cdot q \in \Phi_{n-1}$ follows $\Psi \vdash \langle d_i\rangle \cdot p \doteq \langle d_j\rangle \cdot q$ where $d_i = itr(i)$ and $d_j = itr(j)$.

   From $\Psi \vdash \langle d_i\rangle \cdot p \cdot r \doteq v'$ and $\Psi \vdash \langle d_i\rangle \cdot p \doteq \langle d_j\rangle \cdot q$ follows $\Psi \vdash \langle d_j\rangle \cdot q \cdot r \doteq v'$ and therefore $\Psi \vdash tr(\langle j\rangle \cdot q \cdot r \doteq v')$. □

Lemma 22: The algorithm terminates.

Proof:

Suppose that the algorithm does not terminate. We define Ψ as in lemma 20 and G^t as $G^t = \bigsqcup F_i^t \setminus i$. In each cycle of the algorithm, at least one feature structure F_i^{t-1} is changed because otherwise the algorithm terminates. Since the modified feature structure F_i^t must be subsumed by the original feature structure F_i^{t-1}, there are only three ways in which the feature structure can be modified, namely by changing $F_i^{t-1}(p) = \bot$ to $F_i^t(p) = c$ for some $p \in Path$ and $c \in C$ or by adding a feature $f \in L$ with $f \notin dom(F_i^{t-1}(p))$ to the domain of $F_i^t(p)$ and defining $F_i^t(p)(f) = \bot$ or by combinations of these two elementary steps. In any of these case, the size of the modified feature structure F_i^t will be at least one bigger than that of F_i^{t-1} and therefore $size(G^t) \geq size(G^{t-1}) + 1$.

Since each substructure $G^t(p)$ of G^t is defined for at most $|L|$ many features, it follows that for any N, there is a path p and a t such that $G^t(p)$ is defined and the length of p is larger than N. We will now prove that $\Psi \vdash p \doteq p$ must hold and that the constraint system Ψ is therefore cyclic (lemma 1) in contradiction to the assumption that the grammar never gives rise to cyclic constraints.

Induction hypothesis: $\Psi \vdash desc(F_i^t)[\langle\rangle \to \langle i\rangle]$ for $t < n$.

$\underline{n = 0:}$ $F_i^0 = \bot$, therefore $desc(F_i^0) = \{\langle\rangle \doteq \langle\rangle\}$. There is either a terminal node $n_i^0 = \langle i, F_i^0, \langle w, \Phi\rangle\rangle$ or a non-terminal node $n_i^0 = \langle i, F_i^0, \langle k, \Phi\rangle, D_i\rangle$ such that $\langle 0\rangle \doteq \langle 0\rangle \in \Phi$ by definition of grammar rules and lexical rules. Therefore $\langle i\rangle \doteq \langle i\rangle \in \Psi$ holds by definition of Ψ and therefore $\Psi \vdash desc(F_i^0)[\langle\rangle \to \langle i\rangle]$.

$\underline{n > 0:}$ If ϕ is in $desc(F_i^t)$, then it is either of the form $p \doteq p$ or of the form $p \doteq c$.

In the first case, it follows from $\phi \in desc(F_i^t)$ that $F_i^t \models p \doteq p$. Two cases have to be distinguished:

1. $n_i^t = \langle i, F_i^t, \ldots\rangle$ is the root node during top-down parsing. In this case, $F_i^t = F_i^{t-1}$. By the induction hypothesis, it follows immediately that $\Psi \vdash desc(F_i^t)[\langle\rangle \to \langle i\rangle]$.

2. Otherwise there is either a non-terminal node $n^t_{d_0} = \langle d_0, F^t_{d_0}, \langle k, \Phi\rangle, \langle d_1 \dots d_k\rangle\rangle$ with daughter nodes $\langle d_j, F^t_{d_j}, \dots\rangle$ for $0 < j \leq k$ or a terminal node $n^t_{d_0} = \langle d_0, F^t_{d_0}, \langle w, \Phi\rangle\rangle$ and $k = 0$ such that $i = d_l$ for some $0 \leq l \leq k$ and $F^t_i = MM(\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi)(l)$.

   If $p \doteq p \in desc(F^t_i)$, then $F^t_i \models p \doteq p$ and therefore $MM(\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi) \models \langle l\rangle \cdot p \doteq \langle l\rangle \cdot p$ (lemma 9). By lemma 14, it follows that $\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi \vdash \langle l\rangle \cdot p \cdot r \doteq v$ for some r and therefore that $\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi \vdash \langle l\rangle \cdot p \doteq \langle l\rangle \cdot p$. By the induction hypothesis, $\Psi \vdash desc(F^{t-1}_{d_j})[\langle\rangle \to \langle d_j\rangle]$ for all $0 \leq j \leq k$ and by definition of Ψ, $tr(\Phi) \subseteq \Psi$.

   By lemma 21, it follows from $\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi \vdash \langle l\rangle \cdot p \doteq \langle l\rangle \cdot p$ that $tr(\bigcup_{j=0}^k desc(F^{t-1}_j)[\langle\rangle \to \langle j\rangle] \cup \Phi) \vdash tr(\langle l\rangle \cdot p \doteq \langle l\rangle \cdot p)$. We already know that $tr(\Phi) \subseteq \Psi$ and that $\Psi \vdash desc(F^{t-1}_{d_j})[\langle\rangle \to \langle d_j\rangle]$ for $0 \leq j \leq k$. We conclude than $\Psi \vdash tr(\langle l\rangle \cdot p \doteq \langle l\rangle \cdot p)$ holds, too.

The case $p \doteq c \in desc(F^t_i)$ is analogous to the case $p \doteq p \in desc(F^t_i)$. □

3.4.2 Completeness

The algorithm generates all valid solutions.

Proof

We know by lemma 22, that the algorithm terminates after a finite number of steps T. We also know by lemma 19 that any solution $\{F_i\}$ to the feature computation problem is subsumed by all the intermediate results $\{F^0_i\}, \{F^1_i\}, \dots$ of the algorithm and therefore also by $\{F^T_i\}$.

It remains to show that the algorithm signals success after stopping, if a solution $\{F_i\}$ does exist. Suppose that this were not the case and that the algorithm signals failure.

In this case, there is either a non-terminal node $n^T_{d_0} = \langle d_0, F^T_{d_0}, \langle k, \Phi\rangle, \langle d_1 \dots d_k\rangle\rangle$ with daughter nodes $\langle d_j, F^T_{d_j}, \dots\rangle$ for $0 < j \leq k$ or a terminal node $n^T_{d_0} = \langle d_0, F^T_{d_0}, \langle w, \Phi\rangle\rangle$ and $k = 0$ such that $MM(\bigcup_{j=0}^k desc(F^T_{d_j})[\langle\rangle \to \langle j\rangle] \cup \Phi)$ does not exist. It follows (lemma 16) that $\bigcup_{j=0}^k desc(F^T_{d_j})[\langle\rangle \to \langle j\rangle] \cup \Phi$ is inconsistent.

However, $F^T_i \leq F_i$ entails $F_i \models desc(F^T_i)$ (lemma 7) and $\bigsqcup_{j=0}^k F_{d_j} \setminus j \models \Phi$ holds because $\{F^T_i\}$ is a solution. Therefore $\bigsqcup_{j=0}^k F_{d_j} \setminus j \models \bigcup_{j=0}^k desc(F^T_{d_j})[\langle\rangle \to \langle j\rangle] \cup \Phi$ holds, a contradiction to the assumption that $\bigcup_{j=0}^k desc(F^T_{d_j})[\langle\rangle \to \langle j\rangle] \cup \Phi$ is inconsistent (according to lemma 17). □

3.4.3 Soundness

All solutions generated by the algorithm are valid.

Proof

Suppose that the algorithm stops, signals success and returns $\{F_i\}$.

Then $\{F_i\}$ is a solution because otherwise there is either a non-terminal node $n^T_{d_0} = \langle d_0, F^T_{d_0}, \langle k, \Phi\rangle, \langle d_1 \ldots d_k\rangle\rangle$ with daughter nodes $\langle d_j, F^T_{d_j}, \ldots\rangle$ for $0 < j \leq k$ or a terminal node $n^T_{d_0} = \langle d_0, F^T_{d_0}, \langle w, \Phi\rangle\rangle$ and $k = 0$ such that $\bigsqcup^k_{j=0} F_{d_j} \setminus j \not\models \Phi$. In this case, there are two possibilities:

1. There is an M with $M = MM(\bigcup^k_{j=0} desc(F^T_{d_j})[\langle\rangle \rightarrow \langle j\rangle] \cup \Phi)$. M is not equal to $G = \bigsqcup^k_{j=0} F_j \setminus j$ because $M \models \Phi$ and $G \not\models \Phi$.

   Because $dom(M) = dom(G)$, there must be a label I such that $M(l) \neq M(G)$. If $l = 0$, then the algorithm would have replaced F_{d_0} with $M(0)$ in the last bottom-up pass. If $l > 0$, then the algorithm would have replaced F_{d_l} with $M(l)$ in the last top-down pass. In both cases, the algorithm would have continued for another cycle in contrast to the assumption that it has terminated.

2. $MM(\bigcup^k_{j=0} desc(F^T_j)[\langle\rangle \rightarrow \langle j\rangle] \cup \Phi)$ does not exist. Since the algorithm checks all nodes in each cycle, it would have detected the inconsistency and would have signalled failure in contrast to the assumption that it signalled success. □

3.5 Extensions

The feature logic which has been presented in this chapter has to be extended in several ways in order to serve as the theoretical basis for the feature constraints in YAP.

3.5.1 Variables

In contrast to the presented feature logic, the YAP grammar formalism uses variables to express path equality constraints. However, equations with variables are easily translated into equations without variables (see e.g. [Carpenter, 1992]).

3.5.2 Typing

Feature structures are *typed* in the YAP formalism. The type of a feature structure defines which features are appropriate for it and all appropriate features have to be present in the feature structure. This corresponds to the notion of totally well-typed feature structures in [Carpenter, 1992].

In contrast to grammar formalisms, like e.g. HPSG [Pollard and Sag, 1994], the feature structure types are unordered in the YAP formalism. All types are pairwise inconsistent and none of them is a subtype of another. They resemble data types in programming languages like Modula or C. The advantage of such a typing system is that proper typing can be checked at compile time (with one exception which will be discussed in section 3.5.4).

Feature appropriateness is formally defined as a function $ApproprF : L \times Type \rightarrow Type \bigcup \{\top\}$, where L is the set of feature labels and $Type$ is the set of types. $ApproprF(f, \tau)$ is the most general type which is appropriate for feature f of a feature structure of type τ. Because feature types are mutually disjoint, $ApproprF(f, \tau)$ is in fact the only type which is appropriate for feature f. We define $ApproprF(f, \tau) = \top$ if feature f is not appropriate for feature structures of type τ. For each category X in a YAP grammar there is a corresponding feature structure type τ_X in $Type$.

Besides the appropriateness conditions on features, there are also appropriateness conditions on atomic feature values. Feature value appropriateness is defined as a function $ApproprV : Type \rightarrow C \bigcup \{NONE\}$. $ApproprV(\tau)$ is the most general feature value which is appropriate for type τ. If some feature f is appropriate for τ, i.e. $ApproprF(f, \tau) \neq \top$, then no atomic feature value is appropriate for τ and $ApproprV(\tau) = NONE$ where $NONE$ is such that for all $c \in C$, $c \sqcup NONE$ is undefined. In other words, a feature structure with a constant value has no features because a constant/compound clash would result, otherwise.

The set of constant feature values C differs from the one we having been considering so far. Earlier, we have assumed that C is an unordered set of values. Now we assume that C is ordered such that for each feature type τ, a most general value c_τ exists which subsumes all possible values of features of type τ (see section 3.5.3 for more details).

The feature logic presented in section 3.1 will now be extended to a typed feature logic. To this end, we add type constraints to the feature logic.

Definition 24: A *typed constraint equation* is an equation of the form $p \doteq q$ or the form $p \doteq c$ or the form $p : \tau$, where p and q are feature paths and c is a constant from $C \bigcup \{NONE\}$ and $\tau \in Type \bigcup \{\top\}$.

We also need the following set of deduction rule schemes in addition to the schemes in section 3.1 in order to be able to draw the intended conclusions from the type constraints.

$$
\begin{array}{rcll}
p : \tau & \vdash & p \cdot \langle f \rangle : \tau' & [\text{if } ApproprF(f, \tau) = \tau'] \\
p : \tau & \vdash & p \doteq p & [\text{if } \tau \neq \top] \\
p : \tau & \vdash & p \doteq c & [\text{if } ApproprV(\tau) = c] \\
p \doteq c, p \doteq c' & \vdash & p \doteq c'' & [\text{if } c'' = c \sqcup c' \text{ exists}] \quad \text{(see section 3.5.3)}
\end{array}
$$

The first deduction rule scheme ensures that features have the appropriate type. The second scheme ensures that each appropriate feature is present. The third scheme ensures that atomic features have appropriate values (see section 3.5.3) and the fourth scheme deduces the conjunction of two constant feature values.

Definition 25: A set of typed constraints Φ is called *inconsistent* if and only if it entails two equations of the form

- $p : \tau$ and $p : \tau'$ where $\tau \neq \tau'$, (type clash) or
- $p \doteq c$ and $p \doteq c'$ where $c \sqcup c'$ does not exist. (constant/constant clash)

Because the grammar formalism guarantees that each feature structure is typed (with one exception which is discussed in section 3.5.4) and because feature types are disjunct, the typing constraints can be checked at compile time.

3.5.3 Disjunctive Feature Values

In the YAP formalism it is possible to assign a disjunction of constant feature values to a feature if the feature bears an enumeration type. For efficiency reasons, it is useful to retain this restricted form of disjunctive representation in the feature structures rather than to generate the disjunctive normal form of it.

Therefore we define a hierarchy over the set of possible values of each enumeration type. Each node of the hierarchy corresponds to a subset of the set of possible values and one node dominates another node in this hierarchy, written $d_1 \geq d_2$, if and only if the corresponding set of values is a subset of the set of values of the other node. The least node in the hierarchy corresponds to the set of all possible values. The greatest node corresponds to the empty set and represents an inconsistent value. An example of such a hierarchy is shown in figure 3.3.

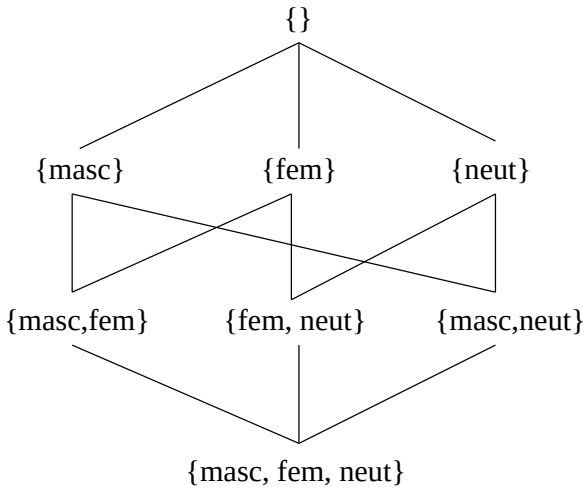

Figure 3.3: A lattice over a set of values

The *unification* of two disjunctive values d_1 and d_2 of the same type returns the least upper bound $d = d_1 \sqcup d_2$ of the two values in the hierarchy unless $d = d_1 \sqcup d_2 = \{\}$. If the least upper bound is $\{\}$, then unification fails. The way the hierarchy is constructed guarantees that the least upper bound always exists and is unique.

Despite the changes in the feature logic, the definition of feature structures (definition 6) remains unchanged. The definition of subsumption (see definition 7), however, is replaced by the following definition.

Definition 26: A feature structure F *subsumes* another feature structure F', written $F \leq F'$, iff either

- $F = c$ and $F' = c'$, where $\underline{c \leq c'}$ or
- for all $f \in dom(F)$ it is the case that $f \in dom(F')$ and $F(f) \leq F'(f)$.

The only difference to the old definition is that an atomic feature structure must subsume the other feature structure rather than being equal to it. We also modify the definition of constraint satisfaction (cmp. definition 9).

Definition 27: A feature structure F *satisfies* an equation $p \doteq v$ iff either

- v is a path $q \in Path$, and $F(p)$ and $F(q)$ are defined, and $F(p) = F(q)$ or
- v is a constant $c \in C$ and $F(p)$ is defined and $c \leq F(p)$.

A feature structure F satisfies a set of equations Φ iff Φ is consistent and F satisfies each equation $\phi \in \Phi$ of the form $p \doteq v$.

Feature trees never violate type constraints because they fail to represent type information. It is not necessary to represent type information explicitly in the feature trees because it is deducible from the type of the root node and the appropriateness specifications. We will later see how the categorial information can be represented in the feature structure by means of a special feature `Cat`.

Finally, we have to adapt the definition of tree models (cp. definition 13).

Definition 28: Feature structure $F = tm(\Phi)$, the *tree model* of a consistent set of acyclic constraint equations Φ, is defined as follows:

- $F(p) = c$ with $c \in C$ iff $\Phi \vdash p \doteq c$ holds and $\Phi \vdash p \doteq c'$ entails $c \leq c'$.
- $F(p) = \bot$ iff $\Phi \vdash p \doteq p$ and $\Phi \vdash p \cdot r \doteq v \;\Rightarrow\; r = \langle\rangle \wedge v \notin C$
- $f \in dom(F(p))$ and $F(p)(f) = F(p \cdot \langle f \rangle)$ iff $\Phi \vdash p \cdot \langle f \rangle \doteq p \cdot \langle f \rangle$.

3.5.4 Feature Structure Lists

An element of the YAP formalism which has no direct correspondence in the feature logic presented so far is the predefined feature type FS_LIST. Features of this type take a list of feature structures with category types as values. In order to integrate them into the feature logic, we will emulate feature structure lists with feature trees (see fig. 3.4). A feature structure list is then either an empty list or a non-empty list. If it is not empty, then it has two features which we name `Head` and `Tail`. The `Tail` feature has the type `FS_LIST`, as well, whereas the value of the `Head` feature could be any feature structure of a categorial type.

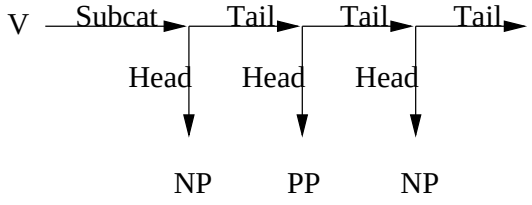

Figure 3.4: A feature structure list represented as a feature tree

There are two problems, however: The features `Head` and `Tail` are only appropriate for feature structure lists which are not empty. So how is $ApproF(Head, FS\_LIST)$ defined? We could split `FS_LIST` into two types `ELIST` and `NELIST`, so that `Head` and `Tail` are only defined for the type `NELIST`. But then the value of `ApropF(Tail,NELIST)` could be either `NELIST` or `ELIST`.

The obvious solution to this problem would be to define `FS_LIST` as a supertype of `ELIST` and `NELIST`. Another problem is the type of the `Head` feature. It could be any categorial feature type. So again, a super-type of these categorial types is needed for the `Head` feature.

It seems that we are forced to switch to a feature logic with ordered types like the one described in [Carpenter, 1992]. But there is another solution: the constraints which apply to feature structure lists can be "hardwired" into the deduction rules of the feature logic in the following way.

In order to represent whether a feature structure list is empty or not (subtype `ELIST` vs. `NELIST`), we add a new feature `Empty` which holds this information and a new feature `Data` which stores the value of the feature structure list if it is not empty. Similarly, we add a new feature `Cat` to store the category of an element of the feature structure list and a feature `Data` to store the element itself. The resulting representation of feature structure lists is slightly more complex than the one depicted in fig. 3.4.

The following inference rule schemes are added:

$$\begin{aligned}
p : FS_LIST, p \cdot \langle Empty \rangle \doteq false &\vdash p \cdot \langle Data \rangle : LIST_DATA \\
p : FS_LIST, p \cdot \langle Data \rangle \doteq p \cdot \langle Data \rangle &\vdash p \cdot \langle Empty \rangle \doteq false \\
p : CAT_FS, p \cdot \langle Cat \rangle \doteq x &\vdash p \cdot \langle Data \rangle \doteq \tau_x \\
p : CAT_FS, p \cdot \langle Data \rangle \doteq \tau &\vdash p \cdot \langle Cat \rangle \doteq x \quad [\text{where } \tau = \tau_x]
\end{aligned}$$

The first two rule schemes make sure that the `Data` feature is defined if and only if the feature structure list is not empty. The third and fourth rule ensure that the value of the `Cat` feature is `x` if and only if the value of the `Data` feature is a feature structure of the corresponding categorial feature type τ_x.

The following restrictions apply to the feature appropriateness function *ApproF*:

$$\begin{aligned}
ApproF(Empty, FS_LIST) &= BOOLEAN \\
ApproF(Head, LIST_DATA) &= CAT_FS \\
ApproF(Tail, LIST_DATA) &= FS_LIST \\
ApproF(Cat, CAT_FS) &= CAT \\
\forall_{f \in L, f \notin \{Empty, Data\}} ApproF(f, FS_LIST) &= \top \\
\forall_{f \in L} ApproF(f, BOOLEAN) &= \top \\
\forall_{f \in L, f \notin \{Head, Tail\}} ApproF(f, LIST_DATA) &= \top \\
\forall_{f \in L, f \notin \{Cat, Data\}} ApproF(f, CAT_FS) &= \top \\
\forall_{f \in L} ApproF(f, CAT) &= \top
\end{aligned}$$

Note that $ApproF(Data, FS\_LIST)$ and $ApproF(Data, CAT\_FS)$ are not defined. The above inference rule schemes deduce whether these features are appropriate and what their types are.

The following restrictions apply to the value appropriateness function *ApproV*:

$$\begin{aligned}
ApproV(FS_LIST) &= NONE \\
ApproV(BOOLEAN) &= \{true, false\}
\end{aligned}$$

$$\begin{aligned} ApproprV(LIST\_DATA) &= NONE \\ ApproprV(CAT\_FS) &= NONE \\ ApproprV(CAT) &= \{\ldots \text{categories in grammar} \ldots\} \end{aligned}$$

3.5.5 Function Values

The YAP formalism includes constraints of the form $v \doteq cat(v_1, v_2)$ where v, v_1, v_2 are variables and *cat* is a string concatenation operator. Using the variable elimination algorithm presented in section 3.5.1, it is easy to convert these constraints to constraints of the form $p \doteq cat(q, r)$. We add the following inference rule scheme to be able to draw the intended conclusions from these constraints:

$$\begin{aligned} p \doteq cat(q,r) &\quad\vdash\quad p : STRING,\ q : STRING,\ r : STRING \\ p \doteq cat(q,r),\ q \doteq c_1,\ r \doteq c_2 &\quad\vdash\quad p \doteq c \quad [\text{where } c \text{ is the concatenation of } c_1 \text{ and } c_2] \end{aligned}$$

The first rule makes sure that the arguments of `cat` are strings. The second rule deduces the result of the `cat`-function. The result is only defined if constant values for both input variables are available. It is not possible to use such a constraint to deduce the value of an argument from the result and the other argument of the operation (as in the Prolog language).

3.5.6 Parse Forests

The algorithm has to deal with ambiguities in the input parse tree. In theory, it is possible to compute the feature structures for each alternative parse independently with the presented algorithm. For efficiency reasons, however, it is necessary to work with a more compact *parse forest* representation. The next chapter presents a parsing algorithm which operates directly on parse forests.

Chapter 4

Parsing

4.1 Parsing Strategies

Parsing strategies for feature-structure based grammar formalisms with a context-free backbone can be divided into three classes. The *interleaved* strategy first applies the context-free part of a rule to generate a new node and checks the associated feature constraints immediately afterwards. The *non-interleaved* strategy first builds complete parse trees covering the whole input string based on the context-free part of the grammar and checks the feature constraints in a second step. The third strategy converts the constraints expressed in the context-free backbone into feature constraints and solves the constraint system as a whole. An advantage of the last strategy is the uniformity of the processing mechanism and the ability to process grammars without a context-free backbone.

Maxwell and Kaplan [Maxwell III and Kaplan, 1994] explore several variants of the first two strategies in their LFG parser. They conclude "... that non-interleaved pruning is always better than interleaved pruning." The reason probably is that the time spent on context-free parsing is neglectable compared to the time spent on the feature constraint evaluation. So it pays off if some feature computations can be saved by doing context-free parsing first. Maxwell and Kaplan also confirm a finding by Nagata [Nagata, 1992] "... that a medium-grained phrase-structure grammar performs better than either a coarse-grained or a fine-grained grammar." The reason might be that a coarse-grained context-free grammar is less restrictive than a medium-grained grammar and therefore saves fewer feature computations. Parsing with too fine-grained context-free grammars on the other hand is inefficient because context-free parsing is slower with the larger grammar and might even take more time than the feature computations. Also the resulting context-free parse forests are larger due to unresolved ambigities which might further slow down the parser.

Since the third strategy above is equivalent to parsing with an extremely coarse-grained grammar, the first strategy, non-interleaved parsing, seemed most promising and has been chosen for YAP.

4.2 Context-Free Parsing

Context-free parsing is the first step of the parser. A bit-vector implementation of the Cocke-Kasami-Younger algorithm (BCKY) developed by Andreas Eisele, is used for this purpose. The conversion of the grammar to Chomsky normal form which is required by the CKY algorithm is carried out automatically. Other context-free parsers which return the same packed *parse forest* format[1] can be used, as well.

4.3 Computation of Feature Structures

After context-free parsing, the parser decorates the parse forest with feature structures. These feature structures must satisfy the constraints of the grammar rules and lexicon entries which were used to build the parse forest. As long as the parse tree is unambiguous, the computation of the feature structures is simple. A standard unification algorithm will satisfy all constraints in one pass through the parse tree. Value sharing between features with an equality constraint guarantees that modifications to the value of one feature are propagated to the other feature.

4.3.1 Dealing with Parse Forests

Usually the result of context-free parsing is highly ambiguous because the context-free grammar is less restrictive than the YAP grammar from which it is derived. Millions of parse trees are not uncommon for large sentences. This huge number of analyses makes it impossible to process each analysis independently using the simple method described before. The parser instead has to operate directly on the compact representation of the parse trees – called parse forest – which is returned by the context-free parser. Parse forests can be represented as *and/or graphs*. For each *and*-node there is a corresponding grammar rule or – if it is a terminal node – a lexicon entry. Each non-terminal node corresponds to the left hand side of a grammar rule and its daughter nodes correspond to the right hand side. *Or*-nodes represent ambiguities in the parse forest. Each daughter node of an *or*-node is an *and*-node which constitutes one analysis of the *or*-node.

The computation of feature structures in parse forests is complicated by the fact that value sharing between feature structures in an *and/or* graph is not possible because one and the same node may have several mother *and*-nodes belonging to distinct analyses. Value sharing would in this case lead to cross-talk between different analyses.

It is possible to compute the feature structure of the root node in one bottom-up pass through the parse forest. In case of LFG and HPSG, the feature structure of the root node is the only feature structure needed because it contains all relevant information. The YAP parser, however, has to compute the feature structures of all nodes. To this end, the parse forest has to be traversed twice, first bottom-up and then top-down. The top-down step is necessary to update the feature structures of the non-root nodes.

[1] See section 4.3.

4.3.2 Disjunctive Feature Structures

In order to compute feature structures for the nodes in an *and/or* graph, the *or*-nodes have to be dealt with appropriately. Since an *or*-node represents a set of alternatives, its feature structure is defined as the disjunction of the feature structures of its daughter nodes. There are several possibilities to deal with such disjunctive feature structures. The simplest approach is to represent feature structures in disjunctive normal form (DNF), i.e. as a set of (non-disjunctive) feature structures, and to solve the feature constraints of each rule for all possible combinations of daughter node feature structures by means of standard unification. Since the number of combinations is equal to the product of the numbers of feature structures at the nodes, it follows that this approach is only tractable if the number of alternative feature structures is not too big.

DNF computation involves some redundancy because computations pertaining to common features of alternative feature structures are repeated. It is possible to avoid this redundancy with a more compact representation of disjunctive feature structures which factors out the common constraints. *Contexted constraints* [Maxwell III and Kaplan, 1989] are particularly effective in factorizing feature constraints. However, as Maxwell and Kaplan note, a computational overhead is associated with their technique which only pays off if the (disjunctive) feature structures contain many independent ambiguities. This type of ambiguity is frequent in the LFG formalism which Maxwell and Kaplan considered. If the propagation of syntactically irrelevant information is avoided (see the discussion in section 2.6), such independent ambiguities are far less frequent. Hence the simpler DNF approach has been chosen in the implementation YAP.

4.3.3 Feature Computation in YAP

YAP employs the same iterative method to compute the feature structures as the algorithm presented in section 3.4. After the context-free parse forest has been read, the feature structures of the nodes are initialised according to the constraints imposed by the context-free analysis and the type system. In particular, the type of a feature structure is set according to the category of the corresponding node and all features which are appropriate for this type are initialised to their most general values. Each node has a set of ,,analyses" (called *or*-nodes in *and/or*-graph terminology). Each analysis contains pointers to the daughter nodes and to the rule which licensed the analysis. The feature structures are repeatedly modified in order to satisfy the local rule constraints. Feature computation is finished when all local constraints are satisfied. This method bears some resemblance to *constraint relaxation* techniques [Montanari and Rossi, 1991].

As noted earlier (see section 3.2), the result of the unification of two feature trees is not guaranteed to satisfy all constraints satisfied by the argument feature trees because feature trees fail to represent equational constraints. Consider the following YAP grammar:

```
enum TYPE { a, b };       % definition of an enumeration type
category TOP {};          % definition of category TOP
category X { TYPE F,G; }; % definition of category X
TYPE v;                   % definition of variable 'v'
```

```
TOP {} -> 'X {F=a;};        % feature F must have value 'a'
"x": X {F=v;G=v;};          % features F and G must be equal
```

Parsing the input string "x" with the context-free part of the grammar results in a parse tree consisting of two nodes, a root node of category TOP and a daughter node of category X (see fig. 4.1). The feature structure of the X node has two features F and G which are initialised to their most general value which is the value 'a or b'. The feature structure of the X node must satisfy the constraints of the lexicon entry for 'x' and the constraints of the grammar rule. At the beginning, features F and G have the same value 'a or b' and the constraints of the lexicon entry are satisfied. In order to satisfy the constraints of the grammar rule, the value of feature F is modified to 'a'. However, this leads to a violation of the constraints in the lexicon entry which were previously satisfied. The feature structure of X is modified again, changing the value of feature G to 'a'. Thereafter, all constraints are satisfied and parsing is finished.

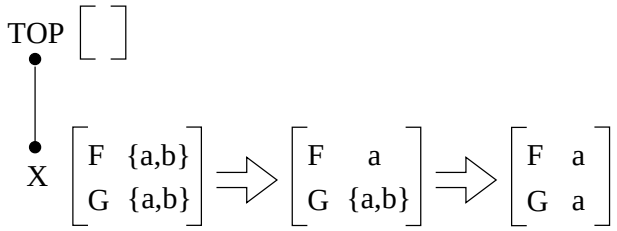

Figure 4.1: Feature computation

Computation of the feature structures in a parse forest proceeds bottom-up and top-down in turn using the functions `bottom_up_parse` and `top_down_parse` presented below. The main function is `parse`.

The Feature Computation Algorithm

The following data structures and functions are used by the algorithm:

N:	sentence length
chart[i][k]:	nodes covering the input string from position i to k
old_chart[i][k]:	nodes covering the input from i to k as computed in the last pass
n.start:	start position of node n
n.end:	end position of node n
n.link:	the set of new nodes linked to node n
n.analyses:	the set of alternative analyses of node n
n.processed:	flag indicating whether this node has been processed
n.fs:	feature structure of node n
a.number_of_daughters:	number of daughter nodes in analysis a
a.daughter[i]:	the i^{th} daughter node in analysis a
a.rule:	the grammar rule which licenses analysis a
r.assignments[i]:	set of assignments for node i

first_link(n):	returns the first element from n.link
next_link(n,n'):	returns the next element after n' in n.link
is_last_link(n,n'):	true if n' is the last element in n.link, otherwise false
var_value:	array of feature values
get_value(f,$\langle i \rangle$·p):	returns the value of feature structure f[i] at feature path p
replace(f, p, v):	replaces the value of feature structure f at feature path p with v
read_parse_forest():	reads a parse forest and initialises the feature structures

A node n is characterised by its feature structure, its start and end position and its set of possible analyses. Each analysis a is characterised by the grammar rule and the set of daughter nodes.

```
1   parse()
2
3       read_parse_forest()
4       first_cycle ← true
5       do
6           old_chart ← chart
7           chart ← new_chart()
8           successful ← bottom_up_parse()
9           if successful and (node_changed or first_cycle)
10              first_cycle ← false
11              old_chart ← chart
12              chart ← new_chart()
13              successful ← top_down_parse()
14      while node_changed and successful
15      return successful
16
17
18  bottom_up_parse()
19
20      node_changed ← false;
21      result ← false
22          % for all nodes covering the whole sentence
23      for all nodes n in old_chart[0,N] do
24          if bu_parse_node(n) = true then
25              result ← true
26      return result
27
28
29  bu_parse_node(n)
30
31          % check whether this node has already been processed
32      if n.processed then
33          if n.link = ϵ then
34              return false
35          else return true
```

```
36          % start processing of node n
37      n.processed ← true
38      result ← false
39      for all analyses a ∈ n.analyses do
40          K ← a.number_of_daughters
41          valid ← true      % initialisation
42              % call bu_parse for all the daughter nodes
43          for i ← 1 to K do
44              if bu_parse_node(a.daughter[i]) = false then
45                  valid ← false
46                  i ← K      % Skip processing of the remaining daughter nodes
47          if valid = true then      % Process all combinations of daughter nodes
48              a' ← make_copy(a)      % create a new temporary analysis
49              for i ← 1 to K do      % with the first combination of new daughter nodes
50                  a'.daughter[i] ← first_link(a.daughter[i])
51              do      % Loop over all combinations of daughter nodes
52                  if bu_parse_analysis(n, a') = true then
53                      result ← true
54                  for i ← 1 to K do      % build the next combination of daughter nodes
55                      if is_last_link(a.daughter[i], a'.daughter[i]) = true then
56                          a'.daughter[i] ← first_link(a.daughter[i])
57                      else
58                          a'.daughter[i] ← next_link(a.daughter[i], a'.daughter[i])
59                          i ← K;      % exit loop
60              while is_last_link(a.daughter[K], a'.daughter[K]) = false
61      return result
62
63
64  bu_parse_analysis(n, a)
65
66      f[0] ← n.fs      % n.fs is old and will be recomputed
67      for i ← 1 to a.number_of_daughters do
68          f[i] ← a.daughter[i].fs      % the daughter feature structures f[i] are new
69      if compute_variables(var_value, f, a.rule) = false then
70          return false
71      newf ← build_new_fs(var_value, n.fs, a.rule.assignments[0])
72      if newf ≠ n.fs then
73          node_changed ← true;
74      n' ← insert_node(n, newf)
75      add_link(n, n')
76          % add new analysis to the list of analyses of n'
77      n'.analyses ← n'.analyses ∪ {a}
78      return true
79
80
81  top_down_parse()
```

```
82
83      node_changed ← false;
84      result ← false
85          % for all nodes covering the whole sentence
86      for all nodes n in old_chart[0,N] do
87          n' ← insert_node(n, n.fs)      % root nodes are simply copied
88          if td_parse_node(n, n') = true then
89              result ← true
90      return result
91
92
93  td_parse_node(n, n')
94
95          % check whether this node has already been processed
96      if n' ∈ n.link then
97          if n'.analyses ≠ ϵ then
98              return true
99          else
100             return false
101     add_link(n, n')
102     result ← false
103     for all analyses a ∈ n.analyses do
104         if a.number_of_daughters = 0 then
105                 % terminal node
106             a' ← make_copy(a)
107             n'.analyses ← n'.analyses ∪ {a'}
108             result ← true
109         else
110                 % nonterminal node
111             if td_parse_analysis(n', a) = true then
112                 result ← true
113     return result
114
115
116 td_parse_analysis(n, a)
117
118     K ← a.number_of_daughters
119     f[0] ← n.fs
120     for i ← 1 to K do
121         f[i] ← a.daughter[i].fs
122     if compute_variables(var_value, f, a.rule) = false then
123         return false
124     a' ← make_copy(a)
125     for i ← 1 to K do
126         newf ← build_new_fs(var_value, f[i], a.rule.assignments[i])
127         if newf ≠ f[i] then
```

```
128             node_changed ← true;
129         n' ← insert_node(n, newf)
130         if td_parse_node(n, n') = false then
131             return false
132         a'.daughter[i] ← n'
133     n'.analyses ← n'.analyses ∪ {a'}
134     return true
135
136
137 compute_variables(var_value, f, r)
138
139     for i ← 1 to r.number_of_variables do
140         var_value[i] ← ⊥
141         for all equations of the form v_i = α do
142             if α is a feature path then
143                 v ← get_value(f, α)
144             else if α is of the form op(v1,v2,...,vN) then
145                 v ← compute_function(var_value, α)
146             else if α is a constant then
147                 v ← α
148             var_value[i] ← least_upper_bound(var_value[i], v)
149         if var_value[i] = ⊤ then
150             return false
151     return true
152
153
154 build_new_fs(var_value, oldf, a)
155     newf ← make_copy(oldf)
156     for all assignments of the form path:=v in a do
157         replace(newf, path, var_value[v])
158     return newf
159
160
161 insert_node(n, fs)
162
163     for all nodes n' ∈ chart[n.start, n.end] do
164         if n'.fs = fs then
165             return n'
166     n' ← make_copy(n)
167     n'.fs ← fs
168     chart[n.start, n.end] ← chart[n.start, n.end] ∪ {n'}
169     return n'
170
171
172 add_link(n, n')
173
```

174 if n' $\notin$ n.link
175 insert n' in n.link at first position

The main function `parse` calls `bottom_up_parse` and `top_down_parse` in turn until either all nodes are unchanged or the whole parse forest is found to be inconsistent. At least one complete cycle consisting of a bottom-up and a top-down pass is necessary to guarantee that all constraints are satisfied. The parser uses two charts. One chart, called `old_chart`, contains the feature structures computed in the previous step. Each feature structure corresponds to a node in the chart. The other chart, called `chart`, is incrementally filled with the feature structures computed in the current pass. The nodes in the new chart are linked to the nodes in the old chart from which they originated.

The `bottom_up_parse` function calls `bu_parse_node` with each node covering the whole input sentence and `bu_parse_node` builds the new feature structures bottom-up. `bu_parse_node` checks first whether the current node has been processed before. If this is not the case, then it computes the new feature structures of the daughter nodes for each of its analyses. Each daughter node may be linked to more than one new feature structure. Therefore it is necessary to check all possible combinations in order to compute the valid feature structures for the current node. `bu_parse_analysis` is called to actually compute a feature structure. Once a new feature structure has been build, `insert_node` checks whether a node with the same feature structure is already contained in the new chart. If this is not the case, then a new node is inserted. In either case, a new analysis with pointers to the daughter nodes is added to the node and the new node itself is linked to the original node in the old chart.

The `top_down_parse` function copies all nodes covering the whole sentence to the new chart and calls `td_parse_node` to recompute the feature structures of their subtrees. td_parse_node checks first whether the current node has been processed before. This is the case if it is linked to the old node. If a link exists, the function returns. Otherwise, the link is created in the next step and the function `td_parse_analysis` is called for all analyses of the node. td_parse_analysis recomputes the feature structures of all daughter nodes, inserts them into the new chart, adds the new analysis to the list of analyses of the mother node and calls top_down_parse to process the subtrees dominated by the daughter nodes.

Cyclic Parse Forests

The above algorithm is a simplification which works only for acyclic parse forests. The YAP formalism, however, permits rules with a cyclic context-free backbone like the following:

```
VBAR {Subcat=r;Slash=[np];} -> ‘VBAR {Subcat=[np|r];Slash=[];}
                                  NP* {}=np;
```

The context-free backbone of this grammar rule is the rule `VBAR -> VBAR`. This rule adds a cycle to each parse forest containing a VBAR node. In order to deal with cyclic parse forests, `bu_parse_node` has to be modified: If a node n has a cyclic analysis, then this cyclic analysis has to be visited again if a new node is added to $n.link$ after it was processed. Therefore the loop over all analyses of the cyclic node is repeated until no new analysis can be added anymore. Analyses which are not cyclic are removed once they have been processed. Each

cyclic analysis maintains a pointer `first_old` in order to remember which analyses of the cyclic node have been considered, so far. The pseudo code of the new function is shown below. Modified lines are marked with an asterisk (*).

```
29   bu_parse_node(n)
30
31       % check whether this node has already been processed
32     if n.processed then
33       if n.link = ε then
34         return false
35       else
36         return true
37     n.processed ← true
38     result ← false
38a*   % initialisation of variables
38b*   rec_ana_exists ← false
38c*   analysis_added ← false
38d*   for all analyses a ∈ n.analyses do
38e*     a.first_old = ε
38f*   do
39       for all analyses a ∈ n.analyses do
40         K ← a.number_of_daughters
41         valid ← true
41a*       recursive ← false
42           % process the daughter nodes
43         for i ← 1 to K do
43a*         if a.daughter[i] = n then      % recursive structure?
43b*           recursive ← true
43c*           rec_ana_exists ← true
43d*           if a.first_old = n.link      % nothing to be done
43e*             valid ← false
43f*             i ← K      % Skip processing of remaining daughter nodes
44*          else if bu_parse_node(a.daughter[i]) = false then
45             valid ← false
46             i ← K      % Skip processing of the remaining daughter nodes
47         if valid = true then      % Process all combinations of daughter nodes
48           a' ← make_copy(a)
49           for i ← 1 to K do
50             a'.daughter[i] ← first_link(a.daughter[i])
51           do
52             if bu_parse_analysis(n, a') = true then
53               result ← true
53a*             if rec_ana_exists = true then
53b*               % remember to process cyclic analyses again
53c*               analysis_added ← true
53d*           is_last_combination ← true;
54             for i ← 1 to K do      % build the next combination of daughter nodes
```

```
54a*                    a'.daughter[i] ← next_link(a.daughter[i], a'.daughter[i])
55*                     if a'.daughter[i] = ϵ or      % all links processed?
55a*                        (a.daughter[i] = n and     % cyclic analysis and
55b*                         a'.daughter[i] = a.first_old)     % all links processed?
55c*                    then
56                          a'.daughter[i] ← first_link(a.daughter[i])
57                      else
58*                         is_last_combination ← false;
59                          i ← K;     % exit loop
60*                 while is_last_combination = false
60a*            if recursive = false then
60b*                remove a from n.analyses     % only cyclic analyses are processed again
60c*    while analysis_added = true
61      return result
```

Cycles involving more than one node are possible if a cyclic set of chain rules exists, like e.g. `X -> Y, Y -> X`. Such derivations do not make much sense in NLP grammars, so they are ignored by the parser.

Processing of cyclic parse forests is guaranteed to terminate if the grammar is *offline parsable* as defined in [Shieber, 1992]. To make sure that the parser will terminate is considered the task of the grammar designer, just as it is the task of a programmer to make sure that his/her program will terminate.

4.4 Grammar Compilation

A compiler is used to transform the text representation of the grammar into a form which is easy to process for the parser. It attempts to reduce the number of computations required during parsing and reports syntax errors in the input grammar.

4.4.1 Grammar Transformation

When processing a grammar rule, lexical rule or template definition, the compiler first transforms the rule constraints into a set of constraint equations. The left side of each equation is a variable. The right side is either a variable, a feature path, a constant or a function value. The only functor in the YAP formalism is the string concatenation operator **cat**. This transformation includes the following actions:

1. For each node specification `XP {...}`, add a constraint `cat=XP` to the constraints of this node. This is also done for embedded nodes which are elements of feature structure list specifications as in `Subcat=[NP{}]`

2. Replace any constraint of the form `p=c` where `c` is not a variable with the two constraint equations `v=p` and `v=c`, where `v` is a new variable.

3. Replace any constraint of the form `p=v` where `v` is a variable with `v=p`.

4. Transform relative paths to absolute paths:

   (a) For each (potentially embedded) constraint equation of the form path $= [C_1\{CS_1\}, C_2\{CS_2\}, ..., C_n\{CS_n\}]$ where C_i is a category name and CS_i is a set of embedded constraints, add the constraints $v = p$ and $v = []$ to the same constraint set where v is a new variable and p is of the form $\underbrace{\text{tail.tail...tail}}_{n \text{ times}}$.

   (b) Replace constraints of the form $path = [C_1\{CS_1\}, C_2\{CS_2\}, ..., C_n\{CS_n\}]$ or $path = [C_1\{CS_1\}, C_2\{CS_2\}, ..., C_n\{CS_n\}|*]$ where `path` is a feature path with the constraint set $\bigcup_i CS_i[\langle\rangle \rightarrow path.\underbrace{tail.tail...tail}_{i \text{ times}}.head]$ where $CS_i[\langle\rangle \rightarrow p]$ results from prefixing `p` to all feature paths in the constraints of CS_i.

   (c) Prefix all feature paths with `0.` in constraints of the mother node

   (d) Prefix all feature paths with `i.` in constraints of the i$^{\text{th}}$ daughter node

This transformation is guaranteed to terminate because each individual transformation applies only for a finite number of times and no transformation can be applied recursively on its output.

As an example, consider the following grammar:

```
enum NUMBER {sg, pl};
enum CASE   {nom, gen, dat, acc};

struct AGR { NUMBER Number; CASE Case; };

category VP { FS_LIST Subcat; };
category  V { FS_LIST Subcat; };
category NP { AGR Agr; };

restrictor+ NP_R(NP) {Agr};

NP_R np;
FS_LIST r;

VP {Subcat=[NP{Agr.Case=nom;}]=r;} ->
        `V {Subcat=[np|r];} NP {Agr.Case=acc;}=np;
```

After the transformation, the following constraint set is obtained:

```
v1 = 0.Subcat
v1 = 1.Subcat.tail
v2 = 0.Subcat.tail
v2 = []
v3 = 0.Subcat.head.cat
v3 = NP
v4 = 0.Subcat.head.Agr.Case
```

```
v4 = nom
v5 = 1.Subcat
v6 = 1.Subcat.head.cat
v6 = NP
v7 = 1.Subcat.head
v7 = 2
v8 = 2.Agr.Case
v8 = acc
```

Now, templates are expanded by adding the constraints contained in the definition of a template to the rule constraints. Since variable names are local to a rule or template, the compiler replaces the template variables by new variables in order to avoid possible conflicts with variable names used in the rule constraints.

After template expansion the compiler adds constraints to implement feature inheritance and to define the values of automatic features. There are two automatic features, `Phon` and `HeadLex`. The `HeadLex` feature is properly defined by the feature inheritance mechanism and requires no further action.

The value of the `Phon` feature of a trace node is the empty string. The compiler therefore adds two constraint equations `p=i.Phon` and `p=""`, where `p` is a new variable and `i` is the position of the daughter node in the list of daughter nodes. In order to define the `Phon` feature of the mother node of a grammar rule, the compiler adds the constraints `p1=1.Phon`, `p2=2.Phon`, ..., `pN=N.Phon` for the `N` daughter nodes and the constraints `p0=0.Phon` and `p0=cat(p1,...,pN)` to define the **Phon** feature of the mother node. In case of lexicon entries, the compiler adds two constraints `p=0.Phon` and `p="word"` if the lexicon entry has the form `"word": X {...};`.

In our example, the following constraints are added at this point:

```
v9  = 1.Phon
v10 = 2.Phon
v11 = 0.Phon
v11 = cat(v9,v10)
```

In order to determine the constraints required by the feature inheritance rule, the compiler first checks which features of the head daughter node are not assigned a value. These are the features which never appear as a prefix of the right hand side of a constraint equation. If the compiler finds such a feature, it checks whether a feature with the same name and type is defined for the mother node. If this is the case, the compiler adds two equations `v=0.f` and `v=i.f` in order to unify the feature **f** of the mother node and the i-th daughter which is the head daughter. Similarly, the compiler checks whether a feature of the mother node is unassigned and "inherits" its value from a feature with the same name and type of the head daughter.

In our example, the following constraints are added at this point:

```
v12 = 0.HeadLex
v12 = 1.HeadLex
```

In the next step, constraints with variables of a `restrictor` type are replaced by a set of equations according to the restrictor definition. In our example the two constraints

```
v7 = 1.Subcat.head
v7 = 2
```

are replaced by the following constraints

```
v7 = 1.Subcat.head.Agr
v7 = 2.Agr
```

Similarly, the compiler replaces equations with variables of a structured feature type by a set of equations for the subfeatures. In our example, the equations for v7 are replaced by these four constraints:

```
v13 = 1.Subcat.head.Agr.Number
v13 = 2.Agr.Number
v14 = 1.Subcat.head.Agr.Case
v14 = 2.Agr.Case
```

In the next step, feature structures are *flattened* by replacing structured features with a set of new features, corresponding to the subfeatures of the structured feature. The above equations are replaced with:

```
v13 = 1.Subcat.head.AgrNumber
v13 = 2.AgrNumber
v14 = 1.Subcat.head.AgrCase
v14 = 2.AgrCase
```

where `AgrCase` and `AgrNumber` are new feature names. The flattened feature structures are easier to store and to process.

The compiler simplifies the resulting set of constraint equations and eliminates redundancies. To this end, it adds a constraint `y=x` for any pair of equations `x=rhs` and `y=rhs` with the same feature path or function value on the right hand side. Then it eliminates variable renaming constraints `y=x` and replaces all occurrences of variable **y** in other equations with **x**. Finally the compiler eliminates duplicates and replaces any pair of equations `x=c1` and `x=c2` where **c1** and **c2** are constants and **c** is the least upper bound of **c1** and **c2** by a new equation `x=c`. If the least upper bound is $\top$, the compiler reports an error.

Thereafter, the compiler uses the substitutivity rule to infer additional constraints. Consider the following example:

```
v2 = 0.Subcat.tail
v1 = 0.Subcat
v1 = 1.Subcat.tail
v2 = []
```

These constraints entail the additional constraint:

```
v2 = 1.Subcat.tail.tail
```

These inferences are necessary in order to be able to compute the minimal extensions of feature structures in one pass as described in section 4.3.3.

Finally, the compiler sorts the equations by the variables on the left side and generates *assignments*. If the constraint set of a variable contains an equation `v=c` with an non-disjunctive constant on the right side, then it generates a *fixed assignment* `p:=c` for each equation `v=p` where **p** is a feature path, and deletes `v=p`. If `v=c` is the only remaining equation with variable **v** on the left side, it is deleted as well. After generating fixed assignments, the compiler generates a *variable assignment* `p:=v` for each remaining path equation `v=p`. The corresponding path equation is not deleted in this case.

The compiler sorts the variables so that **x** follows **y** if variable **x** depends on **y**, i.e. if an equation `x=op(...,y,...)` exists. If a circular dependency is detected, the compiler reports an error. The parser will later compute the values of the variables in this order. The compiler also sorts the assignments so that an assignment `p:=v` will precede any assignment `p.q:=v'` to an embedded feature.

When the compilation of our sample rule is finished, the following data is obtained:

- context-free backbone

  ```
  VP -> V NP
  ```

- List of equations

  ```
  v0 = 1.Phon
  v1 = 2.Phon
  v2 = 1.Subcat.tail
  v2 = 0.Subcat
  v3 = 0.HeadLex
  v3 = 1.HeadLex
  v4 = 1.Subcat.head.Agr.Number
  v4 = 2.Agr.Number
  v5 = 0.Phon
  v5 = cat(v0,v1)
  ```

- assignments at mother node

  ```
  Phon := v5
  HeadLex := v3
  Subcat := v2
  Subcat.tail := []
  Subcat.head.cat := NP
  Subcat.head.Agr.Case := nom
  ```

- assignments at daughter node 1

```
HeadLex := v3
Subcat.tail := v2
Subcat.head.Agr.Number := v4
Subcat.tail.tail := []
Subcat.head.cat
Subcat.head.Agr.Case := acc
Subcat.tail.head.cat
Subcat.tail.head.Agr.Case := nom
```

- assignments at daughter node 2

```
Agr.Number := v4
Agr.Case := acc
```

This data structure is tailored to the requirements of the algorithm for the computation of minimal extensions (see section 4.3.3). The same information could also be represented with a feature graph (with the exception of constraints involving operators like the string concatenation operator which have no equivalent).

4.4.2 Generation of the Context-Free Grammar

YAP allows the grammar designer to augment the context-free backbone of the grammar by *incorporation* of features. The compiler will generate the set of refined context-free rules which are licensed by the grammar rules. The information obtained from the preceding compilation steps facilitates this task.

In order to generate the context-free rules which are consistent with a grammar rule, the compiler first assigns an arbitrary order to the incorporated features of all the nodes in the rules. Then it determines the set of possible values for each incorporated feature. If **p** is the feature path corresponding to one of the incorporated features and if there are two equations `v=p` and `v=c`, then the set of possible values is the set of non-disjunctive values subsumed by the (possibly disjunctive) constant **c**. The preceding processing steps of the compiler ensure that at most one such equation `v=c` exists. If no corresponding pair of equations is found, all values defined for the type of the feature are allowed.

In the next step, the compiler fixes the value of the first incorporated feature to the first one of its possible values. Before the value of the next feature is fixed, the compiler checks whether it is unified with a preceding feature via two equations `x=p` and `x=q` where **p** points to the current feature and **q** to the preceding feature. If this is the case, then the value of the preceding feature is the only possible value of the current feature. Otherwise the first one of its possible values is chosen. The compiler continues until the value of the last feature has been fixed. At this point, the compiler prints the obtained context-free rule. The other consistent rules are computed with backtracking.

The algorithm for the generation of the context-free grammar uses the following data structure and functions:

r.number_of_inc_features: total number of incorporated features
r.path[i]: path of the i^{th} incorporated feature
r.node[i]: number of the node to which the i^{th} incorporated feature belongs
Value[i]: current value of the i^{th} incorporated feature
init(r): compute r.number_of_inc_features, r.path, r.node
print_cfg_rule(r): print the next context-free rule

And here is the algorithm itself as pseudo code:

```
1   generate_cfg_rules(r)
2
3       init(r)
4       enumerate_cfg_rules(r, 1)
5
6
7   enumerate_cfg_rules(r, i)
8
9       if i > r.number_of_inc_features then
10          % The value of all features is fixed
10          print_cfg_rule(r)
10          return
10      % Check for unification with a preceding feature
11      p ← r.path[i]
12      if two equations v = p and v = q exist s.t. q=r.path[j] for some j<i then
14          vs ← {Value[j]}
10      % Check for a constant feature constraint
15      else if there are two equations v = p and v = c or an assignment p := c then
16          vs ← c
17      else
18          vs ← all_possible_values(r.inc_feature[i].type)
19      for all values v in vs do
20          Value[i] ← v
21          enumerate_cfg_rules(r, i+1)
```

Consider the following sample grammar:

```
enum NUMBER {sg, pl};
enum CASE   {nom, gen, dat, acc};
enum VFORM  {fin,inf,ger,pap};

struct AGR { NUMBER Number; CASE Case; };

category VBAR { VFORM VForm; FS_LIST Subcat, Slash; };
category  NP  { AGR Agr; };
category  N   { AGR Agr; };
category  DT  { AGR Agr; };
```

```
VBAR incorporates {VForm};
NP incorporates {Agr.Number, Agr.Case};
N  incorporates {Agr.Number, Agr.Case};
DT incorporates {Agr.Number, Agr.Case};

restrictor- NP_R(NP) {};

NP_R np;
AGR a;
FS_LIST r;

VBAR {Subcat=r;Slash=[np];} -> `VBAR {Subcat=[np|r];Slash=[];}  NP*{}=np;

NP {Agr=a;} -> DT {Agr=a;} `N {Agr=a;};
```

The compiler will generate the following context-free grammar for this grammar:

```
VBAR/fin -> VBAR/fin
VBAR/ger -> VBAR/ger
VBAR/inf -> VBAR/inf
VBAR/pap -> VBAR/pap

NP/sg/nom -> DT/sg/nom N/sg/nom
NP/sg/gen -> DT/sg/gen N/sg/gen
NP/sg/dat -> DT/sg/dat N/sg/dat
NP/sg/acc -> DT/sg/acc N/sg/acc
NP/pl/nom -> DT/pl/nom N/pl/nom
NP/pl/gen -> DT/pl/gen N/pl/gen
NP/pl/dat -> DT/pl/dat N/pl/dat
NP/pl/acc -> DT/pl/acc N/pl/acc
```

Because cyclic feature constraints are not allowed, the parser must check whether a grammar is free of them. Cyclic constraints within a single rule are detected offline by the YAP compiler (see section 4.4). But cycles may also arise from the interaction of several rules. Consider the following grammar:

```
"xyz": X { F=[*|l]; G=l;};
Y {} -> `X { G=[*|l]; F=l;};
```

The grammar allows to rewrite `"xyz"` first to `X` and then to `Y`. The feature structure of the X node has to satisfy the constraints of both rules which together define a cyclic list. This type of cyclicity is not detected by the compiler. A heuristic is used in YAP to detect and eliminate cycles at parse time. This heuristic limits the depth of feature structure to a fixed maximal value, e.g. 10. If a feature structure exceeds this limit, the parser stops and reports an error. The user may then have a look at the problematic feature structure and decide whether the grammar is in fact cyclic and has to be modified or whether the depth limit has to be raised.

4.5 Implementational Details

In order to make the parser efficient with respect to runtime as well as space requirements, some optimization strategies have been used which are described in this section.

4.5.1 Lexicon Compression

In order to achieve high coverage, the lexicon of a parser has to be large. With a straightforward encoding of the data, the space requirements for the compiled lexicon become very high. Lexica usually contain a lot of redundancy, however. The compiler has to reduce this redundancy in order to make the size of the lexicon tractable.

The information of the rules is stored in the form of linked lists, i.e. each element is linked to the next one by a pointer. Different lists often have many elements in common. If the common elements of two lists are located at the end of the lists, it is sufficient to store the common tail of both lists only once. The compiler uses simple heuristics in order to move elements which are unlikely to be shared with other lists towards the beginning of a list if the order of the elements is irrelevant. With this technique it was possible to compress a lexicon with 300,000 entries to some 18 MBytes, which is about 63 bytes per entry. In a simple implementation of the lexicon, about a third of these 63 bytes would we used up to store just the word form and the lemma.

4.5.2 Avoiding FS Recomputations

Sometimes it is known in advance that the recomputation of a feature structure will not lead to any changes. This is the case when the feature structures of the mother node and all daughter nodes remained unchanged during the last recomputation. The `bu_parse_analysis` and `td_parse_analysis` from section 4.3.3 have to be modified in order to avoid recomputation in these cases. The modified lines are markes with an asterisk.

```
64    bu_parse_analysis(n, a)
65
65a*    mod ← false
66      f[0] ← n.fs
66a*    if n.modified = true then
66b*      mod ← true
67      for i ← 1 to a.number_of_daughters do
68        f[i] ← a.daughter[i].fs
68a*      if a.daughter[i].modified = true then
68b*        mod ← true
68c*    m ← false
68d*    if mod = true then
69        if compute_variables(var_value, f, a.rule) = false then
70          return false
71        newf ← build_new_fs(var_value, n.fs, a.rule)
72        if newf ≠ n.fs then
73          node_changed ← true;
73a*        m ← true
```

```
73b*    else
73c*       newf ← n.fs
74      n' ← insert_node(n, newf)
74a*    if m = true then
74b*       n'.modified ← true
75      add_link(n, n')
76         % add new analysis to the list of analyses of n'
77      n'.analyses ← n'.analyses ∪ {a'}
78      return true
79
80
81   top_down_parse
82
83      node_changed ← false;
84      result ← false
85         % for all nodes covering the whole sentence
86      for all nodes n in old_chart[0,N] do
87         n' ← insert_node(n, n.fs)
87a*      n'.modified ← false
88         if td_parse_node(n, n') = true then
89            result ← true
90      return result
91   ...
115
116  td_parse_analysis(n, a)
117
117a*  mod ← false
118     K ← a.number_of_daughters
119     f[0] ← n.fs
119a*  if n.modified then
119b*     mod ← true
120     for i ← 1 to K do
121        f[i] ← a.daughter[i].fs
121a*     if a.daughter[i].modified then
121b*        mod ← true
122*    if mod = true and compute_variables(var_value, f, a.rule) = false then
123        return false
124     a' ← make_copy(a)
125     for i ← 1 to K do
125a*     m ← false
125b*     if mod = false then
125c*        newf ← f[i]
125d*     else
126           newf ← build_new_fs(var_value, f[i], a.rule)
127           if newf ≠ f[i] then
128              node_changed ← true;
128a*           m ← true
129           n' ← insert_node(n, newf)
129a*        if m = true then
129b*           n'.modified ← true
130        if td_parse_node(n, n') = false then
131           return false
132        a'.daughter[i] ← n'
```

```
133     n'.analyses ← n'.analyses ∪ {a'}
134     return true
135  ...
160
161  insert_node(n, fs)
162
163     for all nodes n' ∈ chart[n.start, n.end] do
164        if n'.fs = fs then
165           return n'
166     n' ← make_copy(n)
167     n'.fs ← fs
167a*  n'.modified ← false
168     chart[n.start, n.end] ← chart[n.start, n.end] ∪ {n'}
169     return n'
```

4.5.3 Lazy Copying

The parser must copy the feature structures before it modifies them because the old feature structure might still be needed afterwards. Copying of complete feature structures is very expensive, however. A *lazy copying* strategy [Karttunen and Kay, 1985, Emele, 1991] is therefore used to avoid unnecessary copying.

Lazy copying copies a feature structure only when one of its features is actually modified and it copies only the higher levels of the feature structure down to the modified feature. The lazy copying strategy is implemented in the function `build_new_fs` printed below.

```
path.feature:   first feature of path
path.next:      path minus its first feature
f.feature[feat]: value of feature feat in feature structure f
f.copied:       flag indicating whether the root of feature structure f has been copied
```

```
154   build_new_fs(var_value, oldf, r)
155
156*     for all assignments of the form path:=v do
157*        newf ← lazy_copy_assign(f, path, var_value[v])
158*     newf ← clear_copy_marks(newf)
159*     return newf
160   ...
176
177*  lazy_copy_assign(f, path, val)
178*     if path is empty then
179*        return val
180*     newfv ← lazy_copy_assign(f.feature[path.feature], path.next, val)
181*     if newfv ≠ f.feature[path.feature] then
182*        if f.copied = false then
183*           f ← copy_root(f)
184*        f.feature[path.feature] ← newfv
185*     return f
```

```
186*
187*
188*  clear_copy_marks(fs)
189*
190*      if fs.copied = false then
191*          return fs
192*      for all features in fs do
193*          clear_copy_marks(fs.feature[f])
194*      fs.copied = false
195*      return fs
```

The lazy copying strategy is also used in the function `least_upper_bound` which unifies its argument feature structures non-destructively.

4.5.4 Feature Structure Representatives

The presented parsing algorithm involves many feature structure equality checks. Equality of feature trees is extensional, i.e. two feature structures are equal if and only if they are structurally equal. It is possible to replace all structurally equivalent feature structures by a single representative. Comparison of feature structures then reduces to a comparison of pointers. Whenever a new feature structure is created, it is checked whether the same feature structure has been generated before. If this is the case, the old feature structure is returned, otherwise the new one is returned. The function `clear_copy_marks` from section 4.5.3 has to be modified as follows:

```
188  clear_copy_marks(fs)
189
190      if fs.copied = false then
191          return
192      for all features in fs do
193          clear_copy_marks(fs.feature[f])
194      fs.copied = false
194a* if fs' ∈ fs_table exists with fs'.feature[f] = fs.feature[f] for all features f then
194b*     return fs'
194a* else
194a*     insert(fs, fs_table)
195      return fs
```

If `fs_table` is organized as a hash table where the hash key is computed from the values of the features of a feature structure, then the insertion and lookup operations require constant time on average.

4.5.5 Chart Insertion

The insertion of new nodes into the chart is an expensive operation if the chart cells are organized as lists because each insertion has to examine all members of the list in order to

find out whether the new element is already contained in the list. There is no fixed bound to the length of these lists as in the case of context-free parsing because the number of feature structures is at worst exponential in the length of the input string. In order to make chart insertion an operation of constant runtime complexity, chart cells are organized as hash tables. Another data structure which is organized as a hash table rather than a list is `n.link`.

4.5.6 Storing the Results of Expensive Computations

Due to the redundancy of the representation of feature structures in disjunctive normal form, i.e. as a set of alternative feature structures (cp. section 4.3.2), it is possible that expensive operations like feature structure unifications are repeated. In order to avoid this, the parser stores the results of expensive computations in a hash table. Before such an operation is executed, it is checked whether the result is already available from the hash table. Two types of operations are hashed by the YAP parser: feature structure unification and string concatenation.

Chapter 5

Experimental Results

5.1 The English Grammar

All experiments reported in this chapter were carried out with versions of the English YAP grammar which is printed in appendix B. This grammar uses HPSG-style Slash and Subcat features to check constraints on argument structure and constituent movement. Most category names were taken over from the Penn Treebank database [Marcus et al., 1993]. The grammar contains 250 rules plus 50 rules for handling quotation and commas plus 45 rules for coordination. The grammar covers a broad range of linguistic phenomena, among them

- subcategorization
- long distance dependencies
- questions, imperatives, subjunctives and relative clauses
- raising and control verbs
- small clauses
- extrapositions
- coordination including some frequent cases of non-constituent coordination

The lexicon is based on the COMLEX lexical database [Grishman et al., 1994] which contains almost 40,000 base forms. It is supplemented by a hand-built lexicon which contains entries for most function words and some other words for which the Comlex entry was not appropriate for some reason. It is also possible to create additional corpus-specific lexical entries for part-of-speech tagged corpora.

5.2 Parsing the Wall-Street-Journal Corpus

The Wall Street Journal corpus, which has been manually parsed by the Penn Treebank project [Marcus et al., 1993], was reparsed with the English YAP grammar. Lexicon entries

for words which were not contained in the standard lexicon were derived from the part-of-speech tags in the Treebank corpus. The tags were not used during parsing itself, however. Quotation marks were ignored. More than 7 words per second were parsed on average on a Sun Ultra-2 workstation with 500 MByte RAM. Three times the parser stopped prematurely due to memory exhaustion. The computation of the feature structures was the most time-consuming part of parsing.

For 80 percent of the sentences, the parser generated at least one analysis, and for 54 percent of the sentences, there was at least one analysis which was compatible with the Penn Treebank analysis. The matching of the YAP analysis with the Treebank analysis will be described in section 5.4. The matching is not perfect, however. Sometimes, it fails to detect important differences between two analyses and sometimes two equivalent analyses are classified as inconsistent merely because of differences in the way syntactic phenomena are described.

Therefore 100 sentences were parsed and manually inspected to estimate how often the parser really generated an acceptable analysis. For 57 of the inspected sentences, the parser had produced a Treebank-compatible analysis, but for only 48 sentences was one of the analyses indeed correct. Interpolating these results, the portion of sentences with an acceptable analysis is probably around 45 percent in the larger corpus.

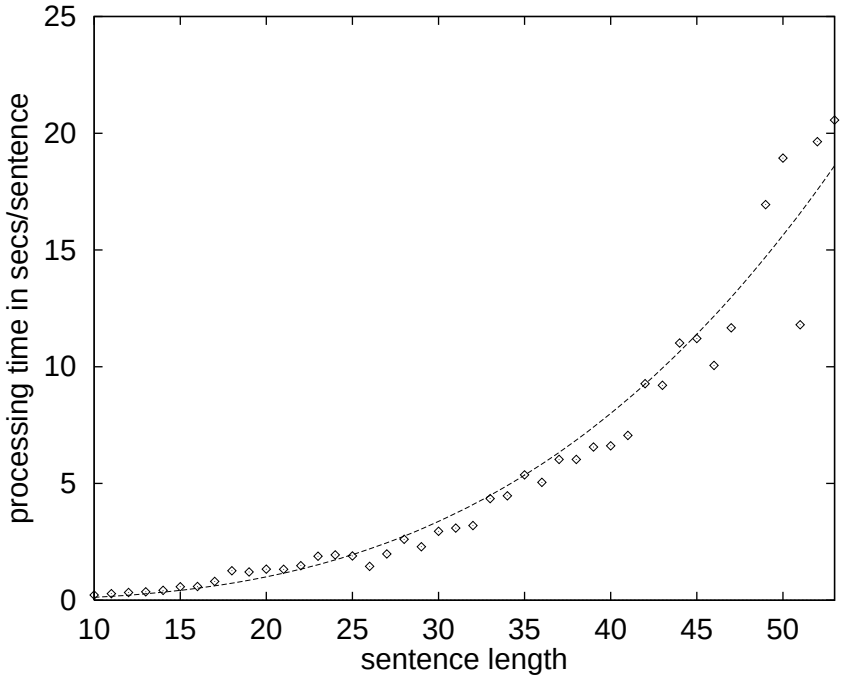

Figure 5.1: Empirical parsing complexity

Parsing of grammars with feature structures is known to be NP-complete in the worst case even if the grammar is offline parsable. This means that the worst-case runtime of the parser is exponential in the length of the input string (at least for all known algorithms). Fortunately, grammars used for parsing natural language usually do not show this worst-case runtime behaviour. Therefore we are more interested in the runtime complexity which is observed with real grammars and real data. Figure 5.1 shows how much time it took on average for the parser to parse sentences of length 10 to 55 from the Penn Treebank. The dots representing

the average parsing times are well approximated by a cubic function (the dashed line in the diagram)[1].

strategy	25 sentences	1 complex sent.
all optimizations	65.9	180
no hashing of unifications	67.4	193
no hashing of string concatenations	79.3	244
recomputing always	67.3	236

Table 5.1: Parse times for 25 randomly selected sentences and a single complex sentence

Another experiment was carried out to check the influence of some of the optimization strategies described in section 4.5 on parsing time. A randomly selected set of 25 sentences was parsed with different variants of the parser in the first part of the experiment. In the second part of the experiment, a single complex sentence with many analyses was parsed. In each variant of the parser, one optimization was switched off. Table 5.1 shows the results. Hashing of unifications showed minor effects on parsing speed. Hashing of string concatenation operations was more effective. Presumably string concatenation operations are more likely to be repeated than feature structure unifications. Avoiding unnecessary recomputation of feature structures had a larger influence on the parsing of the complex sentence than on the parsing of the simpler sentences.

The impact of the incorporation of features into the context-free grammar was examined, as well. In contrast to [Maxwell III and Kaplan, 1994], I only observed a small speedup of 3 percent for the best combination of incorporated features compared to parsing without feature incorporation. Incorporation of morpho-syntactic features like number, gender and case made the parser very slow because the parse forest generated by the context-free parser became very big, slowing down both context-free parsing and the calculation of the feature structures. Overall, a close relationship between the number of nodes in the context-free parse forest and parsing time has been observed. In those cases where the incorporation of features considerably increased the size of the context-free parse forest, it proved useful to skip the initialisation of feature structures with the values of incorporated features, allowing many nodes with different values for the incorporated features but identical category to be merged and thereby reducing the size of the parse forest. But still the grammar which did not incorporate these features showed superior performance.

5.3 Comparison With Other Parsers

The parser was compared to a state-of-the-art parser, the XLE system developed at the research laboratories of the Rank Xerox company which was available for the experiments. A corpus of 700 words, which both parsers were able to parse completely, was used in this experiment. The XLE system parsed this corpus in 110 seconds whereas the YAP parser needed 123 seconds. The tests were run on comparable machines.

[1] There is an outlier at (48, 36.7) which is not shown in the diagram.

It is very difficult to compare these parse times because the parsers are very different wrt. the grammar formalisms, the information contained in the analyses, the degree of ambiguity of the resulting parse trees and other criteria. The YAP grammar tends to generate more analyses (making parsing more difficult) but it defines smaller feature structures (making parsing easier).

In another experiment, the XLE system and YAP were compared on a small grammar for German subordinate clauses with relative clause extraposition. Due to PP attachment ambiguities, the number of attachment sites of the extraposed clause increased exponentially with the length of the sentence. In the YAP version of the grammar the extraposed sentence was attached with Slash percolation. The LFG grammar[2] attached the extraposed clause with functional uncertainty. The resulting empirical runtime complexity for input of varying[3] length was about cubic for the YAP parser and exponential for the XLE system.

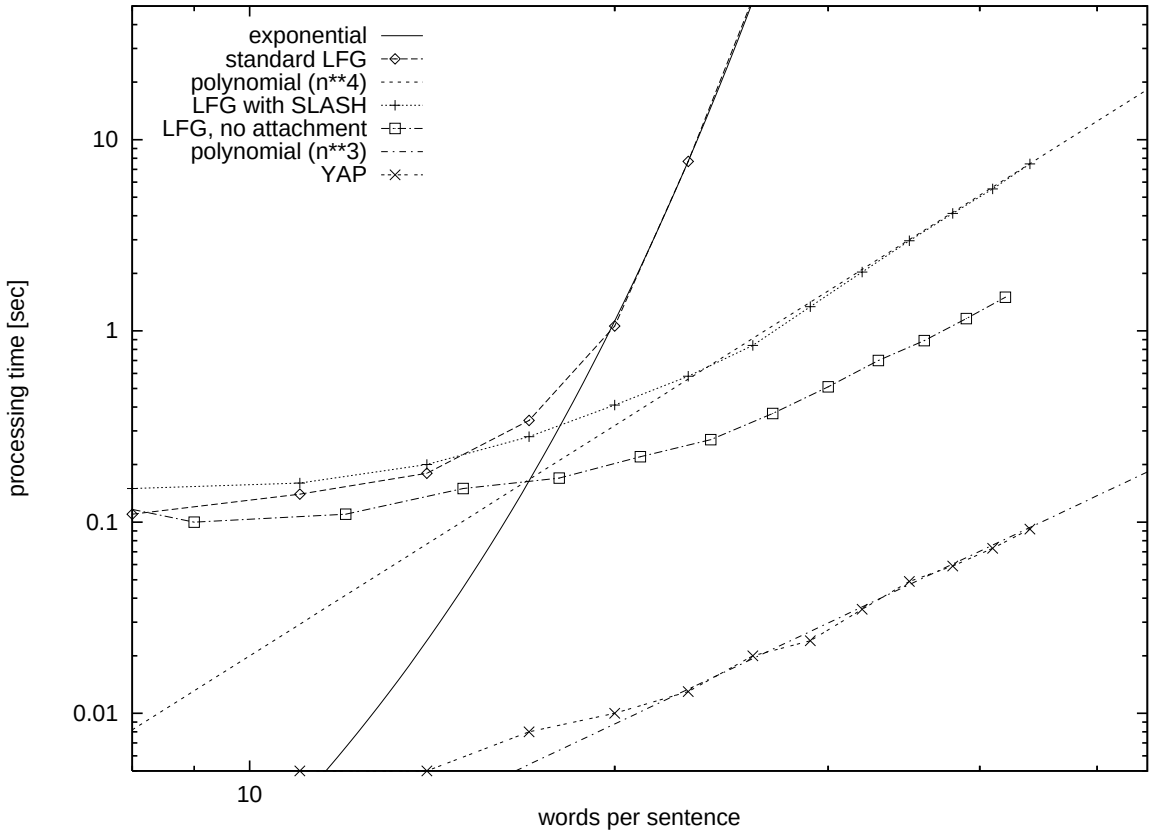

Figure 5.2: Parsing complexity of the YAP grammar and different versions of the XLE grammar

In order to find out whether the difference in parsing complexity was due to differences in the parsing algorithms or due to differences in the grammar formalisms, the XLE grammar was modified. First, functional uncertainty was replaced by Slash percolation. The resulting parsing complexity was better for large sentences but still exponential. The "Slash" grammar

[2]All versions of the LFG grammar have been written by Jonas Kuhn who is an experienced XLE user.

[3]The length of the input was increased by adding PPs in the matrix clause.

was further modified so that the Slash feature contained only agreement information and no pointer to the PRED feature. As a result of this modification, the parsing complexity dropped to about $O(n^6)$. In the last version of the LFG grammar, information about adjuncts was removed from the f-structure and the runtime complexity further dropped to about cubic complexity. But even this version of the XLE grammar was at least an order of magnitude slower than the YAP grammar for all input lengths.

This experiment confirms how useful it is to avoid the evaluation of structure-building constraints during parsing. It also indicates that the presented method for the computation of the feature structures is – at least for the grammar in this experiment – very efficient.

5.4 Tree Matching

This section describes the function `match` which is used to match the parse forest with an unambiguous skeleton parse tree (see section 5.2). It eliminates analyses with crossing brackets and performs some additional checks which are implemented in the function check_nodes.

Two analyses are said to have *crossing brackets* if the first analysis contains a constituent spanning from position i to j and the second contains a constituent spanning from k to l and either $i < k < j < l$ or $k < i < l < j$, i.e. the two intervals overlap and none is completely covered by the other. An example for two analyses with crossing brackets in list notation is the pair ((a b) c) and (a (b c)). Both of these are compatible with (a b (c)), however.

F:	a parse forest
T:	an unambiguous skeleton parse tree
F.nodes	set of all nodes in F
F.roots	the root nodes of F
T.root	the root node of T
n.processed:	flag indicating whether node n has already been processed
n.valid:	flag indicating whether node n is part of the filtered parse forest
n.bu_valid:	flag indicating whether the matching of node n was successfull
a.valid:	flag indicating analysis a is part of the filtered parse forest
a.bu_valid:	flag indicating whether the matching of analysis a was successfull
trace_node(n):	checks whether n is a trace node
terminal_node(n):	checks whether n is a terminal node

```
1   match(F,T)
2
3     for all nodes n ∈ F.nodes do
4       n.processed ← false
5       n.valid ← false
6       for all analyses a ∈ n.analyses do
7         a.valid ← false
8     % do the matching
9     for all nodes n ∈ F.roots do
10      match1(F.root, T.root, 0, T.root.number_of_daughters-1)
11    % mark analyses which did not match
```

```
12    for all nodes n ∈ F.roots do
13      filter(F.root)
14
15
16  match1( fn, tn, ts, tl)
17
18    % check whether the current node was matched before
19    if fn.processed = true then
20      return fn.bu_valid
21    % perform some additional checks like matching category names etc.
22    if check_nodes(fn, tn, ts, tl) = false then
23      return false
24    if terminal_node(fn) = true and terminal_node(tn) = true then
25      for all analyses a ∈ fn.analyses do
26        a.bu_valid ← true
27      return true
28    % skip over traces in the parse tree
29    while ts < tl and trace_node(tn.daughter[ts]) do
30      ts ← ts + 1
31    if terminal_node(fn) = true and terminal_node(tn) = false then
32      if terminal_node(tn.daughter[ts]) = true then
33        return match1(fn, tn.daughter[ts], 0, 0)
34      else
35        return match1(fn, tn.daughter[ts], 0, tn.daughter[ts].number_of_daughters-1))
36    % fn is a nonterminal node
37    fn.bu_valid ← false     % initialisation
38    for all analyses a ∈ fn.analyses do
39      a.bu_valid ← true
40      f2 ← 0
42      f1 ← 0
41      t2 ← ts
43      t1 ← ts
44      % Matching of the daughter nodes
45      while a.bu_valid and t2 ≤ tl do
46        if a.daughter[f2].end < tn.daughter[t2].end then
47          f2 ← f2 + 1
48        else if a.daughter[f2].end > tn.daughter[t2].end then
49          t2 ← t2 + 1
50        else     % end positions are identical
51          % Does one parse forest node match one or more tree nodes?
52          if a.daughter[f1].start = a.daughter[f2].start then
53            if match1(a.daughter[f1], tn, t1, t2) = false then
54              a.bu_valid ← false
55          % Does one tree node match several parse forest nodes?
56          else if tn.daughter[t1].start = tn.daughter[t2].start then
57            if match2(fn, a, f1, f2, tn.daughter[t2]) = false then
```

```
58                  a.bu_valid ← false
59              else     % crossing brackets
60                 a.bu_valid ← false
61              f2 ← f2 + 1
62              t2 ← t2 + 1
63              f1 ← f2
64              t1 ← t2
65        % end of while loop
66        if a.bu_valid = true then
67           fn.bu_valid ← true
68     return fn.bu_valid
69
70
71  match2( fn, a, tn, fs, fl)
72
73     % ignore trace nodes
74     while fs < fl and trace_node(a.daughter[fs]) do
75        fs ← fs + 1
76     if terminal_node(tn) then
77        if match1(a.daughter[fs], tn, 0, 0) = true then
78           return true
79        else
80           return false
81     % tn is a nonterminal node
82     valid ← true     % initialisation
83     f1 ← fs
84     f2 ← fs
85     t1 ← 0
86     t2 ← 0
87     % Matching of the daughter nodes
88     while valid = true and f2 ≤ fl do
89        if a.daughter[f2].end < tn.daughter[t2].end then
90           f2 ← f2 + 1
91        else if a.daughter[f2].end > tn.daughter[t2].end then
92           t2 ← t2 + 1
93        else     % both end positions are identical
94           % Does one parse forest node match one or more tree nodes?
95           if a.daughter[f1].start = a.daughter[f2].start then
96              if match1(a.daughter[f2], tn, t1, t2) = false then
97                 valid ← false
98           % Does one tree node match several parse forest nodes?
99           else if tn.daughter[t1].start = tn.daughter[t2].start then
100                if match2(fn, a, f1, f2, tn.daughter[t2]) = false then
101                   valid ← false
102          else     % crossing brackets
103             valid ← false
```

```
104       f2 ← f2 + 1
105       t2 ← t2 + 1
106       f1 ← f2
107       t1 ← t2
108     % end of while loop
109     return valid
110
111
112  filter(fn)
113
114     if fn.bu_valid = true then
115        fn.valid ← true
116     for all analyses a ∈ fn.analyses do
117        if a.bu_valid then
118           a.valid ← true
119           for i ← 1 to a.number_of_daughters do
120              filter(a.daughter[i])
```

After initialising data structures, the function `match` processes all top nodes of the parse forest (i.e. each node covering the whole input string) by calling the function `match1`. Afterwards, the function `filter` is called for all top nodes to mark those nodes of the parse forest which are not part of a compatible analysis.

Function `match1` takes four arguments, a parse forest node, a node from the treebank analysis and the number of the first daughter node and the last daughter node of the treebank node to be matched with the parse forest node. The function checks first whether the current parse forest node has already been processed. If so, it returns the result[4]. Otherwise, it invokes the function `check_nodes` to perform some additional compatibility tests. If both the parse forest node and the treebank node are terminal nodes, then all analyses of the parse forest node are marked as valid. Otherwise, the function skips over trace nodes in the parse tree and determines the next non-empty daughter node of the treebank node. If the parse forest node is a terminal node, then `match1` is called recursively to match the parse forest node with the next parse tree daughter node. Otherwise, the parse forest node is a non-terminal node and the function has to check all possible analyses of the parse forest node.

For each analysis, it determines whether the first non-empty daughter of the parse forest node has the same span as one or more daughter nodes of the treebank node. If this is the case, then `match1` is called recursively to match this parse forest daughter with the treebank daughters. Otherwise, the function checks whether the next daughter node of the treebank has the same span as a subset of the daughter nodes of the parse forest node. If this is the case, then the function `match2` is called to match the subset with the next treebank daughter. If neither of the two cases applies, then crossing brackets have been detected and the current analysis is marked as incompatible. Otherwise, `match1` continues and matches the remaining nodes.

The function `match2` takes five arguments: a parse forest node, an analysis of this node, a treebank node and the number of the first daughter node and the last daughter node of the

[4]Because the parse tree is unambiguous, it is not possible that the same parse forest node is matched with two different parts of the parse tree.

parse forest node to be matched with the treebank node. The function first determines the next non-empty daughter node of the parse forest node. If the treebank node is a terminal node, then `match1` is called to match the treebank node with the next non-empty parse forest node. Otherwise, the function checks whether the next parse forest node has the same span as a subset of the treebank daughters. If this is the case, then `match1` is called to match the next parse forest daughter with the treebank nodes. Otherwise, `match2` checks whether the first non-empty treebank daughter has the same span as a subset of the parse forest daughters. If this is the case, then `match2` is called recursively to match the treebank daughter with the parse forest daughters. If none of the two cases applies, then crossing brackets exist and the function returns false.

The crossing brackets criterion is only a crude test of the compatibility of two syntactic analyses. It fails to check the compatibility of the labels, it does not differentiate between arguments and adjuncts, and it can not detect the PP-attachment incompatibility of the two analyses (V NP PP) and (V (NP PP)) where the PP attaches to the verb in the first analysis and to the noun in the second. The function `check_nodes` performs these and other checks in order to improve the matching of YAP analyses and Penn Treebank analyses.

Chapter 6

Parse Forest Disambiguation

A major problem in syntactic analysis is ambiguity. It occurs that sentences have thousands of parse trees. Most applications, however, are only interested in the most plausible analyses and cannot deal with thousands of different analyses. Hence it is necessary to score the individual parse trees and to extract the best ones efficiently.

Major sources of syntactic ambiguity in English are prepositional phrases, noun compounds, coordinations and combinations thereof as in the sentence ``The agency studied swings in stock and stock-index prices'', which has seven different analyses depending on where the prepositional phrase attaches, what the conjunction `and` coordinates, and what the noun compound headed by `prices` looks like. Resolution of these types of ambiguity requires semantic information and/or world knowledge. One approach to disambiguation is therefore to eliminate a reading if a contradiction between the semantic content of this reading and background knowledge is derivable. This disambiguation method is used e.g. in the Verb*mobil* project, a German research project aiming at the development of a system for automatic translation of appointment scheduling dialogs. It is difficult to extend this technique to broader domains, however, because a huge amount of hand-coded background knowledge is required and the search space for the derivation of contradictions is so big.

Many disambiguation problems require no in-depth semantic analysis, however. Consider the sentences ``The account comprises trade in goods'' and ``Investors pour money into funds''. By examining a large text corpus, we find that `trade in goods` and `pour ...something... into ...something...` are frequent constructions. Based on this frequency information, it is possible to decide that nominal attachment of the prepositional phrase is probably correct in the first sentence and verbal attachment in the second. Such frequency-based or statistical methods have the following advantages:

- Statistical methods require little or no handcoded information. The relevant information is automatically extracted from training corpora.
- Processing is fast compared to the inferential methods and also compared to the time required for parsing. This is important for processing large corpora.
- Analyses are ranked rather than divided into valid and invalid analyses. It is often the case that several readings of a sentence are acceptable, but that some of them are more plausible than the others.

- Some statistical methods are trained on plain text. These methods require no manually parsed training corpora (treebanks) which are expensive to produce and of limited size compared to the amount of unparsed material available.

6.1 Probabilistic Grammars

A probabilistic grammar is a grammar which assigns probabilities to parse trees, and a parser for a probabilistic grammar disambiguates a sentence by selecting the parse tree with the highest probability. Three types of probabilistic grammars will be discussed here: probabilistic context-free grammars, head-lexicalized probabilistic context-free grammars and feature-based probabilistic context-free grammars. Other types of probabilistic grammars have been presented e.g. in [Black et al., 1992], [Magerman, 1994], [Briscoe and Waegner, 1992], [Eisner, 1996], [Collins, 1996].

6.1.1 Probabilistic Context-Free Grammars

A probabilistic context-free grammar is a context-free grammar which assigns a probability $P(r)$ to each grammar rule `r`. The probabilities of all rules with the same left hand side must sum to 1.

The probability of a parse tree is defined as follows:

$$P(T) \quad = \quad \prod_{rule\ r} P(r)^{F(r,T)}$$

$F(r,T)$ is the number of times, rule `r` was used to generate `T`.

The values of the parameters are learned automatically. When an unambiguously parsed training corpus, also called a *treebank*, is available, it is possible to estimate the probabilities directly from the observed rule frequencies. Otherwise, the *Inside-Outside algorithm* [Baker, 1982], an instance of the more general *Expectation Maximization* (EM) algorithm [Baum and Sell, 1968] can be applied to learn the parameter settings from unparsed corpora.

The availability of efficient training algorithms makes PCFGs very attractive, but they fail to disambiguate some frequent syntactic ambiguities like coordination and PP-attachment ambiguities. Consider the context-free grammar in table 6.1. Any PP-attachment ambiguity will be resolved in the same way in any probabilistic version of this grammar because the decision depends only on the ratio $P(VP \rightarrow VP\ PP) : P(NP \rightarrow NP\ PP)$. Therefore the most probable parse tree for one of the two sentences ``The account comprises trade in goods'' (nominal attachment) and ``Investors pour money into funds'' (verbal attachment) is always incorrect. Furthermore both analyses of the noun phrase ``the sale of the company to the competitor'' always have the same probability because the same rules are applied in two different orders.

6.1.2 Head-Lexicalized Probabilistic Context-Free Grammars

Head-Lexicalized probabilistic context-free grammars (LPCFG) extend the PCFG approach by incorporating information about the lexical head of constituents into the probabilistic

S	→	NP	VP
VP	→	VP	PP
VP	→	V	NP
PP	→	P	NP
NP	→	NP	PP
NP	→	DT	N
DT	→	the	
...			

Table 6.1: A context-free grammar

model [Charniak, 1997, Carroll and Rooth, 1998]. An LPCFG rule looks like a PCFG rule, but one of the daughters is marked as the head. The rule probabilities $P_{rule}(C \rightarrow \alpha)$ (or rather $P_{rule}(C \rightarrow \alpha|C)$ because the probability of a rule is only non-zero if the constituent which is expanded has the appropriate category) are replaced by lexicalised rule probabilities $P_{rule}(C \rightarrow \alpha|C, h)$ where h is the lexical head of the mother constituent C. In other words, in an LPCFG, the probability of a rule depends on the lexical head. Assume that our grammar has two rules which expand `VP`s, namely `VP` → `V NP` and `VP` → `V`. Then the first rule should be more likely if the lexical head is `buy` whereas the second rule should be more likely if the lexical had is the intransitive verb `sleep`.

Once we have applied a lexicalised grammar rule, we need to know the lexical heads of the non-head daughters in the rule before we can further expand the daughter nodes. Therefore LPCFGs comprise another type of probabilities, the `lexical choice` probabilities $P_{choice}(h_d|C_d, C_m, h_m)$, which tell us how likely it is that a given word h_d is the lexical head of a constituent with category C_d when the mother node has category C_m and the lexical head h_m. So, $P_{choice}(book|NP, VP, buy)$ is the probability that the NP argument of the verb `buy` is headed by `book`.

Finally, an LPCFG comprises a probability distribution $P_{start}(h)$ which determines how likely the word h is to be the lexical head of the root node of a parse tree. The probability of a parse tree T is defined as the product:

$$\begin{aligned} P(T) \quad = \quad & P_{start}(head(root(T))) \ * \\ & \prod_{nonterm\ n\ in\ T} P_{rule}(r \mid cat(n), head(n)) \ * \\ & \prod_{nonroot\ n\ in\ T} P_{choice}(head(n) \mid cat(n), cat(parent(n)), head(parent(n))) \end{aligned}$$

where the function `root(T)` returns the root node of a parse tree `T` and `cat(n)` returns the category and `head(n)` the lexical head of a node `n`.

A head-lexicalized version of the context-free grammar in table 6.1 is able to assign the highest probability to the correct parse trees for both sentences, ‘‘`The account comprises trade in goods`’’ and ‘‘`Investors pour money into funds`’’ if the probabilities $P_{choice}(in \mid NP, PP, trade)$ and $P_{choice}(into \mid VP, PP, pour)$ are large compared to the probabilities $P_{choice}(in \mid VP, PP, comprise)$ and $P_{choice}(into \mid NP, PP, money)$.

Because head-lexicalized context-free grammars can be mapped to probabilistic context-free grammars[1], it is possible to use the Inside-Outside algorithm to estimate the parameters. The huge number of parameters, however, leads to serious data sparseness problems. No matter how big the training corpus is, some possible lexical choice events will not be observed in the corpus and their probability will therefore be estimated as 0, which is not acceptable because the probability of any parse tree containing such an event will be zero. The problem can be solved with smoothing techniques which distribute a small fraction of the probability mass of the observed events over unobserved events, either uniformly (see e.g. [Church and Gale, 1991]) or by interpolation [Jelinek and Mercer, 1980] or according to a backoff scheme [Katz, 1987] or based on the similarity of words e.g. in a taxonomy [Abney and Light, 1998] or according to some similarity metric [Dagan and Pereira, 1994].

The data sparseness problem can also be tackled by reducing the number of parameters. In the *latent semantic class* (LSC) model of Mats Rooth [Rooth, 1994], the probability of a word pair $\langle w, w' \rangle$ is defined as $\sum_{\tau \in \mathcal{T}} p(\tau) p_1(w|\tau) p_2(w'|\tau)$, where $\mathcal{T}$ is the set of semantic classes, $p(\tau)$ is the apriori probability of class τ, $p_1(w|\tau)$ is the probability of the first word of the pair given semantic class τ, and $p_2(w'|\tau)$ is the probability of the second word of the pair given τ. The number of semantic classes in the model is defined by the user. The number of parameters of the LSC model grows linearly rather than quadratically with the number of words if the number of classes is fixed. The LSC parameters are automatically learned from a training corpus using a variant of the EM algorithm. Latent semantic classes can be thought of as a set of prototypical semantic relationships. The latent semantic class model could also serve as one of the distributions in an interpolated model or a backoff model.

6.1.3 Probabilistic Constraint-Based Grammars

Probabilistic versions of constraint-based grammars have been examined as well (see e.g. [Eisele, 1994, Brew, 1995, Abney, 1996, Riezler, 1999]). Andreas Eisele proposed a probabilistic extension of CUF, a constraint-logic programming language which is used to implement constraint-based grammar formalism like HPSG. Eisele's method assigns probabilities to clauses of a constraint-logic program and the probability of an analysis is defined as the probability of its proof tree rather than the probability of the parse tree as in PCFGs. The proof tree probability is the product of the probabilities of all clauses in the proof tree. Eisele suggests using the EM algorithm to estimate the probabilities of the clauses from unparsed training text. As Eisele points out, the probability model may assign non-zero probabilitites to proof trees which fail due to violations of feature constraints. The probabilities are therefore renormalized in order to get a probability distribution in which the probabilities of all parse trees add up to 1. This does not completely solve the problem, however, because the obtained parameters may systematically converge to non-optimal results if the distribution of unification failures is not random.

We will illustrate the problem with a YAP grammar because it appears there in the same form. Consider the following grammar:

[1] In order to turn an LPCFG with a set of categories $\mathcal{C}$ and start symbol $S \in \mathcal{C}$ and a set of lexical forms $\mathcal{W}$ into a PCFG, we replace each rule $C_m \rightarrow C_{d_1} \ldots C'_{d_h} \ldots C_{d_n}$ by a set of rules $\{C^w_m \rightarrow C^{C_m,w}_{d_1} \ldots C^w_{d_h} \ldots C^{C_m,w}_{d_n} | w \in \mathcal{W}\}$ with probabilities equal to the rule choice parameters of the LPCFG. We further add a set of rules $\{C^{C',w} \rightarrow C^{w'} | C, C' \in \mathcal{C} \wedge w, w' \in \mathcal{W}\}$ and a set of rules $\{S \rightarrow S^w | w \in \mathcal{W}\}$ with probabilities equal to the lexical choice parameters of the LPCFG.

```
(R1) NP {} -> DT {Number=n;} ‘N {Number=n;};
(R2) "this":  DT {Number=sg;};
(R3) "these": DT {Number=pl;};
(R4) "man":   N {Number=sg;};
(R5) "men":   N {Number=pl;};
```

Assume that the training corpus consists of the two unambiguous phrases `this man` and `these men`. The corresponding parse trees are shown in figure 6.1.

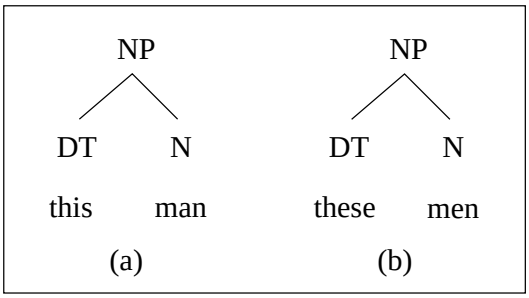

Figure 6.1: Parse trees for the phrases `this man` and `these men`

Training the grammar on these two phrases will return the following probabilities:

P(R1) = 1

P(R2) = P(R3) = P(R4) = P(R5) = 0.5

The sum of the probabilities of all parse trees generated by this grammar (these are the two trees shown in fig 6.1) is 0.5. In order to renormalize the probabilities such that the sum of probabilities of all parse trees equals 1, we multiply the tree probabilities with 2. The resulting tree probabilities are now identical to the relative frequencies of the sentences in the training corpus. This is the desired result because we want the probability model to reflect the distribution in the training corpus.

Now assume that the training corpus contains `this man` twice and `these men` once. In this case, we get the following probability estimates:

P(R1) = 1

P(R2) = P(R4) = 2/3

P(R3) = P(R5) = 1/3

The renormalization factor is 9/5 and the probability of tree (a) is 4/5 and the probability of tree (b) is 1/5. These probabilities are quite different from the relative frequencies in the training corpus which are 2/3 and 1/3. The likelihood of the training data for this model is 0.128. If we choose the following parameters instead

P(R1) = 1

P(R2) = P(R4) = 0.58

P(R3) = P(R5) = 0.42

and the normalisation constant 1.95, then we obtain a likelihood of about 0.148. This proves that the above training procedure is not optimal since it fails to find the parameters which

maximise the probability of the training data. The reason is that the application of rules 2, 3 and 4, 5 is not statistically independent.

Steven Abney [Abney, 1996] presented a method for the disambiguation of constraint-based parses which is based on *log-linear models*. A log-linear model defines a set of features $\{f\}$ with weights λ_f and the probability of a parse tree is defined as:

$$P(T) = \frac{1}{Z}\, e^{\sum_f \lambda_f F(f,T)}$$

where $F(f,T)$ is the frequency of feature f in T and Z is a normalisation constant which is defined as follows:

$$Z = \sum_T e^{\sum_f \lambda_f F(f,T)}$$

PCFGs are a special case of log-linear models where Z is 1 and the features are the grammar rules and the weight of a feature is the logarithm of the probability of the grammar rule. In contrast to PCFGs, however, log-linear models normally do not satisfy the constraint that the weights form a family of probability distributions and they cannot be interpreted as models of a generative process.

Abney also presented an algorithm for the training of log-linear models on treebank bank data which guarantees that the feature weights maximise the likelihood of the training data even when the features are not statistically independent. Abney's algorithm was extended by Stefan Riezler [Riezler, 1999, Riezler, 1998] for unsupervised training on unparsed data. Recently, Riezler and Johnson [Johnson et al., 1999] successfully applied Abney's method in a small-scale experiment in which LFG analyses were disambiguated.

The computation of the normalisation constant Z is a problem in Abney's and Riezler's algorithm. This computation is only exact if the set of parses is finite. Otherwise, Z has to be estimated and there is no guarantee that the "probabilities" of all parses sum to 1. This problem is not relevant for syntactic disambiguation, however, because a change in the constant Z has no influence on the ranking of the parses of a sentence.

6.2 Hybrid Disambiguation Methods

The probabilistic methods presented so far compute scores for complete parse trees and disambiguate the parse forest in one step. It is also possible to disambiguate a parse forest step by step, starting with the most reliable disambiguation decision and removing all analyses which are incompatible with this decision, then removing the analyses which are incompatible with the second-best decision, and so on. This method has the advantage that arbitrary probabilistic and symbolic methods can be used to derive the individual decisions because there is no need to combine scores for decisions numerically in order to get overall scores for whole parse trees. Therefore it is possible to combine the disambiguation methods which have been developed for PP-attachment disambiguation [Hindle and Rooth, 1993, Brill and Resnik, 1994], noun compound disambiguation [Lauer, 1995] and subject-object disambiguation in German [de Lima, 1997].

6.3 A Disambiguation Method for YAP

Because of the theoretical problems (see section 6.1.3) as well as practical problems with probabilistic feature-based grammars (parsing of a large corpus like the British National Corpus (BNC) which contains 100 million words with YAP and the presented English grammar would take more than half a year), a hybrid strategy was chosen to disambiguate the output of the YAP parser with the English grammar.

6.3.1 The Basic Idea

The basic idea of this disambiguation method is to look for word pairs in the parse forest which stand in a head-argument or modifier-head relation. These word pairs are ranked with respect to the strength of the association between the words in the given relation. Starting with the strongest association, all analyses not containing the respective word pair in the given relation are deleted. Then the second-best association in the remaining analyses of the sentence is selected to prune the parse forest, and so on until only one analysis is left or the set of word pairs is empty. If several word pairs have the same association score, all analyses containing either one of them are retained. The association score used by the parser is the *log-likelihood ratio* (see [Dunning, 1993, Daille, 1995]) which is defined as follows:

$$
\begin{aligned}
L(C,w,r,C',w') &= 2(AlogA + BlogB + ClogC + DlogD \\
&\quad -(A+B)log(A+B) - (A+C)log(A+C) \\
&\quad -(B+D)log(B+D) - (C+D)log(C+D) \\
&\quad +(A+B+C+D)log(A+B+C+D)) \\
A &= F(C,w,r,C',w') \\
B &= \left[\sum_{w'} F(C,w,r,C',w')\right] - A \\
C &= \left[\sum_{w} F(C,w,r,C',w')\right] - A \\
D &= \left[\sum_{w,w'} F(C,w,r,C',w')\right] - A
\end{aligned}
$$

Compared to the *mutual information* statistics, the log-likelihood ratio assigns more adequate scores to rare events [Dunning, 1993].

6.3.2 Computation of the Scores

The word pair frequencies have been obtained by parsing 90 million words from the BNC with Carroll and Rooth's head-lexicalized context-free parser for English (see section 6.1.1). The result was a table of estimated lexical choice frequencies $F_{\langle C,w,C',w'\rangle}$ where C is the category of the mother node of a rule, C' is the category of a non-head constituent, w is the lexical

V arg N:	VFC1	→	... NC1 ...
V arg PART:	VFC1	→	... PART_C ...
V arg P: P mod V:	VFC1	→	... PC1 ...
P arg N:	P_C	→	... NC1 ...
N arg V:	NC1	→	... VTOC1 ...
N arg P: P mod N:	NC1	→	... PC1 ...
ADJ arg V:	ADJC1	→	... VTOC1 ...
ADJ arg P:	ADJC1	→	... PC1 ...
ADJ mod N:	NPL=	→	... ADJMOD ...
	NSG=	→	... ADJMOD ...
	NPL_	→	... ADJC1 ...
	NSG_	→	... ADJC1 ...
	NPL_	→	... CD_ ...
	NSG_	→	... CD_ ...
ADV mod P:	PREP_	→	... ADV= ...
ADV mod V:	VFC1	→	... ADV= ...
	VF=	→	... ADV= ...
	S_C	→	... ADV= ...
ADV mod ADJ:	ADJ=	→	... ADV= ...
	CD_	→	... ADV= ...
ADV mod ADV:	ADV_	→	... ADV_ ...
N mod N:	NPL_	→	... NPL_ ...
	NPL_	→	... NSG_ ...
	NPL_	→	... PN_ ...
	NSG_	→	... NPL_ ...
	NSG_	→	... NSG_ ...
	NSG_	→	... PN_ ...
	PN_	→	... NPL_ ...
	PN_	→	... NSG_ ...
	PN_	→	... PN_ ...

Table 6.2: Mapping of lexical choice frequencies from the head-lexicalized grammar to lexical association frequencies

head of the mother node and w' is the lexical head of the daughter node and f is the frequency of occurrence.

Frequencies for a set of lexical associations were derived from this frequency table. The left column in table 6.3.2 shows the different types of word relations for which log-likelihood scores have been computed. The right column shows the lexical choice frequencies which have been added up to obtain frequencies for the respective relation. `V arg N` is the relation between a verb and its nominal argument, `ADV mod V` is the relation between a verbal category and its adverbial modifier.

Example:
$F_{\langle ADV,purely,MOD,ADJ,aesthetic\rangle} = F_{\langle ADJ=,aesthetic,ADV=,purely\rangle} + F_{\langle CD\_,aesthetic,ADV=,purely\rangle}$

Log-likelihood ratios have been computed for the word pairs of each relation type. Only word pairs with a frequency of at least 1 and a log-likelihood ratio of at least 2 were used for disambiguation.

The LPCFG makes no distinction between arguments and adjuncts. Therefore the scores for the `V arg P` relation and the `P mod V` relation have to be computed from the same lexical choice frequencies. To this end, the lexical choice frequencies are split into two sets. The first set contains of verb-preposition pairs where the verb subcategorizes for a prepositional phrase headed by the preposition. The other set consists of the remaining pairs. The first set is used to compute scores for the `V arg P` relation and the second to compute scores for the `P mod V` relation. Scores for the `N arg P` relation and the `P mod N` relation are computed in the same way.

In order to get more significant scores for numbers and names, the counts of all ordinal numbers, all cardinal numbers and all proper names were merged.

So far, we have only considered lexical association scores. Although this kind of information seems to be most important, the parser also needs information about the probabilities of grammar rules and lexical entries. This is necessary to be able to penalize rare readings of words and rare syntactic constructions like e.g. extrapositions. To this end, the frequencies of all word-tag-pairs have been counted in the tagged version of the BNC [Leech et al., 1994] and mapped to the lexical entries. Furthermore, about 1 million words from the Wall Street Journal corpus have been parsed and matched to the treebank analysis (see section 5.4) in order to extract rule frequencies from the filtered parse forests. The score of a grammar rule $L(r)$ or a lexical entry $L(C, w)$ was then defined as the logarithm of its frequency multiplied by 10. This heuristic score worked quite well.

6.3.3 Disambiguation

For efficiency reasons, disambiguation is carried out with a version of the Viterbi algorithm rather than with the stepwise disambiguation procedure described above. The score of a non-terminal node in a parse forest is computed by adding up the scores of the daughter nodes and the score of the grammar rule. If a node has more than one analysis, then the highest score is selected. The score of a terminal node is defined as the sum of the argument scores $L(C, w, arg, C', w')$ plus the score of the lexical entry $L(C, w)$ plus the modifier score $L(C, w, mode, C'', w'')$ in case the maximal constituent headed by the current node modifies some other node. Here, $C/C'/C''$ is either V, N, P, ADJ, ADV or PART depending on whether

the node/its argument/the modified constituent is verbal or nominal, etc. (table 6.3 shows the mapping in detail); $w/w'/w''$ is the lexical head of the current node/the argument/the modified constituent, respectively.

V	←	S, VP, VBAR, V, _V
N	←	NP, NBAR, N
ADJ	←	ADJP, ADJ, _ADJ
ADV	←	ADVP, ADV
P	←	PP, P
PART	←	RP

Table 6.3: Mapping of the categories of the English YAP grammar to the categories of the association table.

The score of a terminal node of category `P` with lexical head `in` which subcategorizes for an `NP` argument with lexical head `goods` and modifies an `NP` with lexical head `trade` is $L(P, in, arg, N, goods) + L(P, in, mod, N, trade) + L(P, in)$.

The English YAP grammar (see appendix B) generates all information required to compute the scores. The lexical head of a constituent is stored in the `HeadLex` feature. Arguments are accessible at terminal nodes via the `Subcat` feature and – in case of prepositions – the `Arg` feature. Modified constituents are accessible via the `Mod_Elem` feature.

In case of coordination, the lexical head is explicitly defined as the concatenation of the lexical heads of the coordinated constituents. We define the score $L(C, W, r, C', W')$, where $r \in \{arg, mod\}$ and W and W' are lists of lexical heads, as $\sum_{w \in W} \sum_{w' \in W'} L(C, w, r, C', w')$. Summing scores produced better results than averaging. Why this is the case can be explained with an example. Consider the sentence `She met the prime ministers of France and Italy`. Assume that $L(P, of, arg, N, France) = 100$ and $L(P, of, arg, N, Italy) = 50$ and that these two are the only relevant scores. If the scores are added in case of coordination, we get a score of 150 for the analysis in which `France` and `Italy` are coordinated and a score of 100 for the analysis in which `the prime ministers of France` and `Italy` are coordinated. In case of averaging, the respective scores are 75 and 100, so that in this case the analysis containing only one of the lexical associations is preferred.

Experiments have shown that the difference between the results of the presented disambiguation method which is based on the Viterbi algorithm and the method which disambiguates step by step by removing all analyses without the respective lexical association is minimal. This justifies the application of the more efficient Viterbi algorithm.

6.3.4 Disambiguation Results

The applicability of the presented disambiguation method was checked in a small-scale experiment in which 100 sentences from the BNC have been parsed, scored and manually disambiguated. The sentences and the disambiguation results are shown in appendix D.

The correct analysis had the highest score for 19 sentences, which had 235 analyses on average, and the unique highest score for 7 sentences with 7.1 analyses on average.

25 percent of the analyses had a higher score on average than the correct analysis and 35 percent had a score at least as good. We will now analyse some sentences where the highest scoring analysis was not the correct one.

6.3.5 Error Analysis

Rockwell said the agreement calls for it to supply 200 additional so-called shipsets for the planes.

In this sentence, the noun-attachment score $L(P, for, mod, N, call)$ (2551) is higher than the verb-attachment score $L(V, call, arg, P, for)$ (2537), thus preferring analyses where `calls` is a noun which is modified by a prepositional phrase headed by `for`.

"Mr. Carlucci, 59 years old, served as defense secretary in the Reagan administration."

In this sentence, the score $L(P, in, arg, N, \langle name \rangle)$ (7663) is much higher than $L(P, in, arg, N, administration)$ (2.4). Therefore the parser prefers an analysis in which `serve` is a transitive verb, `administration` is its argument and ``` ``in the Reagan'' ``` is a PP adjunct. The problem arises because person names and place names are not distinguished.

"In January, he accepted the position of vice chairman of Carlyle Group, a merchant banking concern."

In this sentence, the score $L(P, of, arg, N, concern)$ (870) is higher than the score $L(P, of, arg, N, Group)$ (857). Therefore concern is picked as the head of the NP ``` ``Carlyle Group, a merchant banking concern'' ``` rather than `Group`. Another problem arises because the grammar allows to analyse `In January` as an extraposed adjunct of the NP `concern`. This analysis is preferred because the score $L(P, in, mod, N, concern)$ (92) is higher than that of $L(P, in, mod, V, accept)$ (0).

"Thomas E. Meador, 42 years old, was named president and chief operating officer of Balcor Co., a Skokie, Ill., subsidiary of this New York investment banking firm."

In this sentence, there is a spurious ambiguity concerning the attachment point of the comma after `old`. The comma could either attach to the preceding adjectical phrase or the noun phrase. Both analyses are semantically equivalent and have the same score.

The complex noun phrase ``` ``president and chief operating officer of Balcor Co., a Skokie, Ill., subsidiary of this New York investment banking firm'' ``` is highly ambiguous due to the large number of prepositional phrases and possible coordinations and appositions contained in this phrase. The parser correctly coordinates `president` and `chief operating officer`. Unfortunately, it prefers to attach both "`of`"-PPs to this coordinated NP because the score $L(P, of, mod, N, president)$ (3286) is higher than that of $L(P, of, mod, N, subsidiary)$ (261). There is no restriction in the grammar which would disallow this.

"Balcor, which has interests in real estate, said the position is newly created."

Here, the score $L(V, say, arg, N, position)$ (89) is higher than $L(V, say, arg, V, create)$ (0). Hence an obscure analysis in which ``` ``has interests in real estate'' ``` coordinates with

‘‘`said the position`’’ wins over the correct one. Furthermore, ‘‘`real estate`’’ is analysed as an adjective modifying a noun rather than as a compound, because `real estate` is tagged with `AJ0 NN1` in the BNC corpus.

“In addition to his previous real-estate investment and asset-management duties, Mr. Meador takes responsibility for development and property management.”

The highest-scoring analysis of this sentence has a similar inverted word order as the sentence ‘‘`Behind the silly posturing lies a real dispute.`’’. The subject is the noun `management`, ‘‘`responsibility for development`’’ is coordinated with `property` and ‘‘`addition to his previous real-estate investment`’’ is coordinated with ‘‘`asset-management duties, Mr. Meador`’’ where `Meador`, which is mapped to $\langle name \rangle$, is the head of the second NP. The extraposed PP is a modifier of the object rather than an argument of the verb. The high total score of this analysis compared to other analyses results mainly from the scores $L(P, in, arg, N, \langle name \rangle)$ (7663), $L(P, in, mod, N, development)$ (458), $L(P, in, mod, N, property)$ (106).

“Those duties had been held by Van Pell, 44, who resigned as an executive vice president.”

In this sentence, `Van` is mapped to $\langle name \rangle$ whereas `Pell` is not. Since the score $L(P, by, arg, N, \langle name \rangle)$ (3963) is fairly high, an obscure analysis where ‘‘`duties had been`’’ is analysed as a reduced relative clause and `held` is active voice and ‘‘`Pell, 44, who resigned as an executive vice president.`’’ is the object, wins over the other analyses.

“Before the loan-loss addition, it said, it had operating profit of $10 million for the quarter.”

For this sentence, the correct analysis also has the highest score and there is just one unresolved ambiguity pertaining to whether `quarter` is singular or plural. The tag statistics are not explicit enough to distinguish this.

“The move followed a round of similar increases by other lenders against Arizona real estate loans, reflecting a continuing decline in that market.”

In this sentence, the score $L(P, in, mod, N, increase)$ (18212) is much higher than that of $L(P, in, mod, N, decline)$ (3279). Hence the prepositional phrase ‘‘`in that market`’’ is erroneously attached to `increase`. Due to a tagging error in the BNC corpus – `real` in `real estate` is tagged as an adjective rather than as a noun – the score $L(ADJ, real, mod, N, estate)$ (334) is higher than $L(N, real, mod, N, estate)$ (0).

“Arbitragers weren’t the only big losers in the collapse of UAL Corp. stock.”

In this sentence, the score $L(prep, in, mod, V, be)$ (36694) was very high and the prepositional phrase ‘‘`in the collapse of UAL Corp. stock`’’ was attached to `be` rather than `loser`.

“Mr. Johnson succeeds Harry W. Sherman, who resigned to pursue other interests, in both positions.”

Because of the large score $L(P, in, mod, N, interest)$ (11580), the prepositional phrase ‘‘`in both positions`’’ is attached to the noun `interests` rather than the verb `succeeds`.

“Manville is a building and forest products concern.”

The parser preferred here to coordinate the nouns `building` and `forest` rather than `building` and `forest products`.

"US Facilities Corp. said Robert J. Percival agreed to step down as vice chairman of the insurance holding company."

The parser here preferred an awkward analysis in which ``` ``J. Percival agreed to step down'' ``` is a reduced relative clause attached to `Robert` which is the object of `say`. The reason is that the score $L(V, say, N, \langle name \rangle)$ (48771) was high and that the head of `Robert` is $\langle name \rangle$ whereas the head of `Percival` is `Percival`. If the latter is changed to $\langle name \rangle$ as well, the score of the correct analysis increases, because of the score $L(N, \langle name \rangle, mod, N, \langle name \rangle)$ (61552).

Another problem remains, however: The score $L(P, as, mod, V, say)$ (17) is higher than that of $L(P, as, mod, V, step)$ (0). The reason is that the statistics does not distinguish between the verb `step` and the verb `step down`.

"There was a difference of opinion as to the future direction of the company, a spokeswoman said."

The score of $L(V, say, arg, N, difference)$ (121) is higher than that of $L(V, say, arg, V, be)$ (0). The high value of the former score might be the result of misanalysing sentential arguments during training in such a way that the true subject of the subordinated sentence becomes the object of the main verb.

"Mr. Percival declined to comment."

According to the grammar, the phrase `to comment` is either an argument of the verb `decline`, or an adjunct as in the (non-sense) sentence `Mr. Percival declined in order to comment`. Both analyses get the same score because the score of the verb-argument relation is always 0 in case of verbal arguments (we haven't computed verb-arg-verb scores because they rarely exceed the thresholds). By setting $L(V, v, arg, V, v')$ to some small positive constant for all v and v', the argument reading would be generally favoured and the result would be usually correct.

"In a statement, US Facilities said Mr. Percival's employment contract calls for him to act as a consultant to the company for two years."

Because of the large score $L(V, be, arg, P, for)$ (95122), the parser prefers analyses where `'s` is a clitic form of `is`.

"Mr. Percival will be succeeded on an interim basis by George Kadonada, US Facilities chairman and president."

The high score of $L(P, by, arg, N, < name >)$ (3963) and the fact that `Kadonda` is not mapped to $\langle name \rangle$ results in a preference for analyses where `by George` is a PP-adjunct of the verb `succeeded` which is active voice. ``` ``Percival will be'' ``` is in this case analysed as a reduced relative clause and ``` ``Kadonada, US Facilities chairman and president'' ``` is the object.

"The buy-back represents about 3 % of the company's shares, based on the 3.7 million shares outstanding as of Sept. 30."

Since
the score $L(P, of, arg, N, company)$ (1235) is higher than the score $L(P, of, arg, N, share)$

(511), the parser prefers an analysis which contains the PP ``of the company's'' (analogous to ``The legitimate son of his father's''. The rest of the sentence ``shares, based on the 3.7 million shares outstanding as of Sept. 30.'' is analysed as a reduced relative clause.

"In national over-the-counter trading yesterday, US Facilities closed at $ 3.625, unchanged."

Here, the highest score is $L(P, in, arg, N, US)$ (296). Therefore analyses where US is the head of the noun phrase ``national over-the-counter trading yesterday, US'' (analogous to ``the president of the United States, Bill Clinton'') are preferred by the parser. The score $L(P, in, arg, N, over - the - counter)$ (6.4) is also higher than $L(P, in, arg, N, trading)$ (0), so that analyses containing the PP ``In national over-the-counter'' also score better than the correct analysis.

6.3.6 Conclusions

By disambiguating the output of a feature-based grammar with statistics from a probabilistic context-free parser, it is possible to combine the strengths of both approaches, namely the descriptive power of the feature-based formalism and the trainability of the probabilistic context-free grammar. However, this approach also has some drawbacks.

- There is no theoretical basis for how to combine the log-likelihood ratios. Although the presented heuristic works quite well, it is not optimal.
- The lexical choice frequencies which are used to compute the scores fail to capture all relevant relations. In particular, it would be useful to have information about the preposition as well as the head of the nominal phrase in case of prepositional phrase arguments (cp. [Hindle and Rooth, 1993]). Sabine Schulte im Walde [im Walde, 1998] shows how this information can be extracted from Viterbi parses of the BNC corpus.
- Complex arguments tend to get lower scores than simple arguments. Consider e.g. the sentence ``He was waiting for her to come''. The score of $\langle V, wait, arg, P, for \rangle$ is much higher than the summed scores of $\langle V, wait, arg, V, come \rangle$ and $\langle V, come, subj, N, she \rangle$.
- The lexical choice scores are sometimes incorrect. Among the 70 highest-scoring nominal arguments of "say" are the words problem and there. Probably, these words were the first words of sentential arguments. If such an argument fails to be parsed completely, the parser will attach as many potential arguments to the preceding verb as possible and cover the rest of the sentence with robustness rules.
- Names of persons, companies, organisations, countries, cities etc. are currently not distinguished which sometimes leads to strange results.

The foregoing error analysis suggests the following improvements to the presented disambiguation method:

- Traces should get a negative score in order to penalise complex analyses with movement operations.
- Sentential arguments and VP arguments of verbs should generally get a small positive score to achieve preference over adjuncts.
- The score of verbs should be computed at the preterminal level where the underlying argument structure was mapped to a more surface-oriented structures, e.g. in case of a verb which subcategorizes for a sentence of the form ``for ...NP... to ...VP...'', the Subcat list contains a VP and a subject NP at the terminal level, but a PP, a VP and a subject NP at the preterminal level.
- The classification of proper names has to be improved:
  - At least, person names have to be distinguished from place names. Further distinctions are probably also useful.
  - All proper names have to be classified accordingly.
- Either the grammar or the disambiguation model should somehow capture the fact that two prepositional phrases which are headed by the same preposition are unlikely to attach to the same site.
- The lexical scores of some critical verbs like `say` should be manually checked to eliminate scores like $L(V, say, arg, N, difference)$. Auxiliary verbs should be completely eliminated from the statistics because they cause too many errors.
- Verb forms with and without particles should be distinguished in the statistics.
- Prepositional phrases headed by `as`, `by` and `for` in adjunct position should get low scores if they compete with verbal arguments.
- The grammar has to be improved to eliminate more invalid analyses.
  - The NP grammar has to be improved, e.g. slash propagation has to be better restricted and a feature **Person** should be introduced to be able to restrict certain constructions.
  - Reduced relative clauses have to be better restricted.

Chapter 7

Summary

A feature-based grammar formalism, a parser for this formalism, an English grammar and a disambiguation method have been presented. The grammar formalism combines elements from other grammar formalisms and programming languages. YAP grammars have a context-free backbone like LFGs. Features are typed as in HPSGs. Subcat lists and the Slash percolation mechanism are also familiar from HPSG theory. The type system resembles the type system of programming languages. In contrast e.g. to HPSGs, feature typing only serves to enhance the compiler's ability to detect grammar errors and to make the representation of feature structures more efficient. No information about a specific constituent is stored in feature types[1]. The design of the grammar formalism emphasises efficiency considerations. Templates, default inheritance, restrictor types and automatic features increase the usability of the formalism. The detrimental effects of the propagation of syntactically irrelevant information on parsing efficiency are discussed and an integration of parsing and semantic construction is argued against. Experiments with an LFG parser confirmed that the elimination of irrelevant feature information can speed up parsing dramatically.

A compiler transforms YAP grammars into an efficient internal representation and detects errors in the grammar specification. A subset of the features and the associated constraints is compiled into the context-free backbone grammar. Experiments by Maxwell and Kaplan [Maxwell III and Kaplan, 1996] suggested that this could speed up parsing. This was not confirmed in the experiments with YAP, but feature incorporation turned out to be a useful tool for the development of purely context-free grammars.

The parser consists of a standard context-free parser which generates a parse forest, and a constraint-evaluation component which computes the feature structures. Feature structures are represented as trees rather than graphs which simplifies their storage, access and manipulation. A novel iterative algorithm was presented which computes the feature structures by stepwise approximation. The correctness of the algorithm was formally proved. Several techniques to improve the efficiency of the parser have been presented. In an experiment on data from the Wall Street Journal, the parser was able to process more than 7 tokens per second on average. The runtime of the parser grew about cubically with sentence length on this data.

[1] with a minor exception discussed in section 2.5.2.

In a comparison experiment with a state-of-the-art parser for broad-coverage grammars, the presented parser was almost as fast as the state-of-the-art parser. The result of another experiment with two smaller but more comparable grammars indicates that the presented parser might actually be much faster than the state-of-the-art parser if the parsers are compared on similar grammars.

An English grammar was developed for the YAP formalism. It covers all major syntactic phenomena like subcategorisation, wh-extraction, extrapositions, long-distance dependencies, control and raising verbs, constituent coordination and some frequent forms of ellipsis and non-constituent coordination. The grammar incorporates the COMLEX dictionary and is able to parse about 80 precent of the sentences in the Wall Street Journal corpus. About 50 percent of the sentences received an analysis which was completely correct.

Finally, a disambiguation method based on word association scores was presented. The scores were obtained from frequency information in a 90 million words subcorpus of the British National Corpus which had been parsed with the head-lexicalized context-free parser of Carroll and Rooth [Carroll and Rooth, 1998]. The best analysis of a sentence is extracted with the Viterbi algorithm by summing the association scores. Preliminary results of this disambiguation method are very encouraging.

Bibliography

[Abney, 1996] Abney, S. (1996). Stochastic attribute-value grammars. unpublished manuscript, electronically available at http://xxx.lanl.gov/abs/cmp-lg/9610003.

[Abney and Light, 1998] Abney, S. and Light, M. (1998). Hiding a semantic class hierarchy in a markov model. available from Marc Light's homepage at http://www.ims.uni-stuttgart.de/ light.

[Baker, 1982] Baker, J. (1982). Trainable grammars for speech recognition. In Klatt, D. and Wolf, J., editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550.

[Barwise and Etchemendy, 1988] Barwise, J. and Etchemendy, J. (1988). *The Liar: An Essay on Truth and Circularity*. Oxford University Press.

[Baum and Sell, 1968] Baum, L. E. and Sell, G. R. (1968). Growth transformations on functions on manifolds. *Pacific Journal of Mathematics*, 27.

[Black et al., 1992] Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, page 6. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9405007.

[Brew, 1995] Brew, C. (1995). Stochastic hpsg. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, Dublin.

[Brill and Resnik, 1994] Brill, E. and Resnik, P. (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9410026.

[Briscoe and Waegner, 1992] Briscoe, T. and Waegner, N. (1992). Robust stochastic parsing using the inside-outside algorithm. In *Proceedings of AAAI92 Workshop on Probabilistically-Based Natural Language Processing Techniques*, pages 39–53, San Jose, Ca. an extended version is electronically available at http://xxx.lanl.gov/abs/cmp-lg/9412006.

[Carpenter, 1992] Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.

[Carroll and Rooth, 1998] Carroll, G. and Rooth, M. (1998). Valence induction with a head-lexicalized PCFG. In *Proceedings of Third Conference on Empirical Methods in Natural Language Processing*, Granada, Spain. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9805001.

[Charniak, 1997] Charniak, E. (1997). Statistical parsing with a context-free grammar and word Statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*, Menlo Parc.

[Church and Gale, 1991] Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54.

[Collins, 1996] Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL, University of California*, Santa Cruz, Cal. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9605012.

[Dagan and Pereira, 1994] Dagan, I. and Pereira, F. C. (1994). Similarity-based estimation of word cooccurrence probabilities. In *Proceedings of the 32nd Annual Meeting of the ACL, New Mexico State University*, Las Cruces, New Mexico. Also available electronically at http://xxx.lanl.gov/abs/cmp-lg/9405001.

[Daille, 1995] Daille, B. (1995). Combined approach for terminology extraction: lexical statistics and linguistic filtering. Technical Report 5, UCREL, Lancaster University.

[de Lima, 1997] de Lima, E. (1997). Assigning grammatical relations with a backoff model. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9706001.

[Dörre, 1997] Dörre, J. (1997). Efficient construction of underspecified semantics under massive ambiguity. In *Proceedings of the 35th Annual Meeting of the ACL*, Madrid, Spain.

[Dunning, 1993] Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74.

[Earley, 1970] Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

[Eisele, 1994] Eisele, A. (1994). Towards probabilistic extensions of constraint-based grammars. In Dörre, J., editor, *Computational Aspects of constraint-based linguistic Description II*, pages 3–21. Institute for Computational Linguistics (IMS-CL), Stuttgart. DYANA-2 Deliverable R1.2.B.

[Eisner, 1996] Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 340–345, Copenhagen, Denmark. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9706003.

[Emele, 1991] Emele, M. (1991). Unification with lazy non-redundant copying. In *Proceedings of the 29th Annual Meeting of the ACL, University of California*, pages 323–330, Berkeley, California.

[Grishman et al., 1994] Grishman, R., Macleod, C., and Meyers, A. (1994). Comlex syntax: Building a computational lexicon. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan.

[Groenendijk, 1993] Groenendijk, M. (1993). Xmfed user guide. Technical report, Commision of the European Community.

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

[im Walde, 1998] im Walde, S. S. (1998). Automatic semantic classification of verbs according to their alternation behaviour. Master's thesis, Institute for Computational Linguistics (IMS-CL), University of Stuttgart, Stuttgart,Germany.

[Jelinek and Mercer, 1980] Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam.

[Johnson et al., 1999] Johnson, M., Geman, S., Canon, S., Chi, C., and Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, MD.

[Kaplan and Bresnan, 1982] Kaplan, R. M. and Bresnan, J. (1982). Lexical-Functional Grammar : A formal system for grammatical representation. In Bresnan, J., editor, *The mental representation of grammatical relations*, chapter 4, pages 173–281. The MIT Press.

[Karttunen and Kay, 1985] Karttunen, L. and Kay, M. (1985). Structure sharing with binary trees. In *Proceedings of the 23rd Annual Meeting of the ACL, University of Chicago*, pages 133–136, Chicago, Ill.

[Kasper and Krieger, 1996] Kasper, W. and Krieger, H.-U. (1996). Modularizing codescriptive grammars for efficient parsing. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 628–633, Copenhagen, Denmark.

[Katz, 1987] Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on ASSP*, 34(3):400–401.

[Lauer, 1995] Lauer, M. (1995). Corpus statistics meet the noun compound: Some empirical results. In *Proceedings of the 33rd Annual Meeting of the ACL, Massachusetts Institute of Technology*, pages 47–54, Cambridge, Mass. electronically available at http://xxx.lanl.gov/abs/cmp-lg/9504033.

[Leech et al., 1994] Leech, G., Garside, R., and Bryant, M. (1994). Claws4: The tagging of the british national corpus. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 622–628, Kyoto, Japan.

[Magerman, 1994] Magerman, D. M. (1994). *Natural Language Processing as Statistical Pattern Recognition*. PhD thesis, Stanford University.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[Maxwell III and Kaplan, 1989] Maxwell III, J. T. and Kaplan, R. M. (1989). An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Tec hnologies*, pages 18–27, Pittsburgh. Carnegie Mellon University.

[Maxwell III and Kaplan, 1994] Maxwell III, J. T. and Kaplan, R. M. (1994). The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

[Maxwell III and Kaplan, 1996] Maxwell III, J. T. and Kaplan, R. M. (1996). Unification-based parsers that automatically take advantage of context freeness. Draft.

[Montanari and Rossi, 1991] Montanari, U. and Rossi, F. (1991). Constraint relaxation may be perfect. *Artificial Intelligence*, 48(2):143–170.

[Nagata, 1992] Nagata, M. (1992). An empirical study of rulegranularity and unification interleaving toward an efficient unification-based parsing system. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 177–183, Nantes, France.

[Pollard and Sag, 1994] Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, London.

[Riezler, 1998] Riezler, S. (1998). Statistical inference and probabilistic modeling for constraint-based NLP. In *Proceedings of KONVENS 98*, Bonn.

[Riezler, 1999] Riezler, S. (1999). *Probabilistic Constraint Logic Programming*. PhD thesis, Seminar für Sprachwissenschaft, Universität Tübingen. AIMS Report, 5(1), IMS, Universität Stuttgart.

[Rooth, 1994] Rooth, M. (1994). Two-dimensional clusters in grammatical relations. unpublished manuscript.

[Schiehlen, 1996] Schiehlen, M. (1996). Semantic construction from parse forests. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark.

[Shieber, 1992] Shieber, S. M. (1992). *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Language*. The MIT Press, Cambridge, Ma.

[Simpkins, 1994] Simpkins, N. K. (1994). *ALEP-2 User Guide*. CEU, Luxembourg. This dokument is online available at http://www.anite-systems.lu/alep/doc/index.html.

[Wahlster, 1993] Wahlster, W. (1993). Verbmobil-Translation of Face-to-Face Dialogs. Technical report, German Research Centre for Artificial Intelligence (DFKI). In Proceedings of MT Summit IV,Kobe,Japan.

[Younger, 1967] Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.

Appendix A

BNF Syntax of the YAP Grammar Formalism

```
/*****************************************************/
/*          Grammar Definition                       */
/*****************************************************/

Grammar -> Grammar Declaration | Grammar Rule | Grammar Entry | ''


/*****************************************************/
/*          Declarations                             */
/*****************************************************/

Declaration -> 'auto' Name_List ';'  |
               'category' Name '{' Feature_Decls '}' ';'  |
               'struct'   Name '{' Feature_Decls '}' ';'  |
               'enum' Name '{' Name_List '}' ';'  |
               'restrictor+' Name '(' Cat ')' '{' Feature_List '}' ';' |
               'restrictor-' Name '(' Cat ')' '{' Feature_List '}' ';' |
               'restrictor+' Name '(' Cat ')' '{' '}' ';'  |
               'restrictor-' Name '(' Cat ')' '{' '}' ';'  |
               FType Var_List ';'  |
               Name ':' Cat_List Fe_Struc ';'  |
               Cat 'incorporates' '{' Path_List '}' ';'  |

Feature_Decls -> Feature_Decls Feature_Decl | ''
Feature_Decl  -> FType Feature_List ';'


/*****************************************************/
/*          Definition of Grammar Rules              */
```

```
/*****************************************************/

Rule         -> MNode '->' Node_List ';'
MNode        -> Cat_List Fe_Struc
Node_List    -> Node_List QRNode | ''
QRNode       -> '`' RNode | RNode
RNode        -> Cat_List Fe_Struc | Cat_List '*' Fe_Struc
Fe_Struc     -> '{' Feature_Eqns '}' | Fe_Struc '=' Var
Feature_Eqns -> Feature_Eqns Feature_Eqn ';' | ''
Feature_Eqn  -> Path '=' RHS | Feature_Eqn '=' RHS
RHS          -> String | 'cat' '(' Var_List ')'  |
                '(' Value_List ')' | Value  |
                '[' ']' | '[' Car_List ']' | '[' Car_List '|' Var ']' |
                '[' Car_List '|' '*' ']' | Var

/*****************************************************/
/*          Definition of Lexicon Entries            */
/*****************************************************/

Entry    -> String  ':' Cat_List Fe_Struc ';'  |
            Special ':' Cat_List Fe_Struc ';'

Special  -> '<cardinal>' | '<ordinal>' | '<propername>' | '<default>'


/*****************************************************/
/*          Some Further Definitions                 */
/*****************************************************/

Feature_List  -> Feature_List ',' Feature | Feature
Value_List    -> Value_List ',' Value | Value
Name_List     -> Name_List ',' Name | Name
Var_List      -> Var_List ',' Var | Var
Cat_List      -> Cat  | Template_List
Template_List -> Template | Template_List '&' Template
Path_List     -> Path_List ',' Path | Path
Path          -> Path '.' Feature | Feature
Car_List      -> Car_List ',' Car | Car
Car           -> Cat_List Fe_Struc | '*' | Var

Var      -> Name
Feature  -> Name
Value    -> Name
Cat      -> Name
Template -> Name

Name     -> ([A-Za-z0-9'_$#/]|(\.))+
```

```
String   -> "([^"]|(\.))*"
```

Characters with a special meaning like '$*$', '[', '{' etc. have to be quoted in names with a preceding backslash '\'. Also, the double quote '"' has to be quoted within strings.

Include commands and comments can be inserted at any position. Comments start with a percent sign '%' and extend to the end of the line.

Equations with identical left hand sides like {`f=a;f=b;`} may be abbreviated to {`f=a=b;`}.

Appendix B

The English YAP Grammar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                   %
%  File:    declarations.yap                        %
%  Purpose: Declarations for my English YAP grammar %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart  %
%                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


auto Phon, HeadLex;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Feature Types               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


enum PERSON  {1st,2nd,3rd};
enum NUMBER  {sg,pl};
enum CASE    {nom,gen,acc};
enum DEGREE  {pos,comp,sup,as,too};
enum VFORM   {fin,inf,bse,prp,pap,pas};
enum SFORM   {decl,quest,imp,inv,subj,gerund};
enum CFORM   {quest,rel,arg,argw,argb,subj,adj,coord1,coord2,coord3,coord4,coord5};
enum NFORM   {noun,pronoun,propername,gerund};
enum WHFORM  {quest,rel,expl,-};
enum ORDER   {post,pre};
enum COMMAS  {lr,left,right,-};
enum MOD     {-,verb,noun,adj,adv,prep,sbar,clitic};
enum NPLEVEL {0,1,2};
enum BOOLEAN {+,-};
enum POSITION {left,right};
enum INFL    {1s,3s,13s,2s_pl,bse,past,prp,pap,pas};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Category Definitions          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
category TOP {
  SFORM   SForm;
};


category SM {   %%% sentence punctuation
  SFORM SForm;
};


category SBAR {
  CFORM   CForm;
  STRING  Comp;
  COMMAS  Commas;
  BOOLEAN Coord;
  FS_LIST Slash;
};


category C {
  CFORM   CForm;
};


category S {
  SFORM   SForm;
  BOOLEAN Coord;
  COMMAS  Commas;
  FS_LIST Slash;
};


category VP {
  VFORM   VForm;
  BOOLEAN Aux;
  BOOLEAN Coord;
  COMMAS  Commas;
  FS_LIST Subcat;
  FS_LIST Slash;
};


category VBAR {
  VFORM   VForm;
  FS_LIST Subcat;
  FS_LIST Slash;
};


category V {
  VFORM   VForm;
  BOOLEAN Aux;
  BOOLEAN Coord;
  FS_LIST Subcat;
};
```

```
category _V1 {
  INFL    Infl;
  FS_LIST Subcat;
};


category _V {
  INFL    Infl;
  FS_LIST Subcat;
  STRING  ComlexFrame;
};


category NP {
  NFORM   NForm;
  WHFORM  WhForm;
  CASE    Case;
  NUMBER  Number;
  PERSON  Person;
  DEGREE  Degree;
  BOOLEAN Elliptical;
  BOOLEAN Adjunctive;
  BOOLEAN Coord;
  NPLEVEL NPLevel;
  COMMAS  Commas;
  FS_LIST Slash;
};


category NBAR {
  NFORM   NForm;
  WHFORM  WhForm;
  CASE    Case;
  NUMBER  Number;
  DEGREE  Degree;
  BOOLEAN Elliptical;
  BOOLEAN Adjunctive;
  BOOLEAN Coord;
  FS_LIST Subcat;
  FS_LIST Slash;
};


category N {
  NFORM   NForm;
  NUMBER  Number;
  BOOLEAN Adjunctive;
  BOOLEAN Coord;
  BOOLEAN Compound;
  FS_LIST Subcat;
  FS_LIST Mod_Elem;
};


category DTP {
  WHFORM WhForm;
```

```
  NUMBER Number;
};


category DT {
  WHFORM WhForm;
  NUMBER Number;
};


category PDT {
  NUMBER Number;
};

category ADJP {
  WHFORM  WhForm;
  BOOLEAN Pred;
  BOOLEAN Numerical;
  COMMAS  Commas;
  BOOLEAN Coord;
  DEGREE  Degree;
  FS_LIST Mod_Elem;
};

category ADJ {
  BOOLEAN Pred;
  ORDER   Order;
  BOOLEAN Numerical;
  DEGREE  Degree;
  BOOLEAN Coord;
  FS_LIST Subcat;
  FS_LIST Mod_Elem;
};

category _ADJ {
  BOOLEAN Pred;
  BOOLEAN Nominal;
  DEGREE  Degree;
  FS_LIST Subcat;
  FS_LIST Mod_Elem;
  STRING  ComlexFrame;
};

category CURR {  %%% currency unit
  FS_LIST Subcat;
};

category M {     %%% measure unit
};

%%% genitive marker
category GM {
};


category DEGP {
  DEGREE  Degree;
```

```
};


category DEG {
  DEGREE  Degree;
};


category COMP {
  DEGREE Degree;
};


category ADVP {
  WHFORM  WhForm;
  BOOLEAN Not;
  DEGREE  Degree;
  COMMAS  Commas;
  BOOLEAN Coord;
  MOD     Mod;
  FS_LIST Mod_Elem;
};


category ADV {
  WHFORM  WhForm;
  BOOLEAN Not;
  DEGREE  Degree;
  BOOLEAN Coord;
  MOD     Mod;
  FS_LIST Mod_Elem;
};


category PP {
  WHFORM  WhForm;
  ORDER   Order;
  COMMAS  Commas;
  BOOLEAN Coord;
  FS_LIST Arg;
  FS_LIST Slash;
  MOD     Mod;
  FS_LIST Mod_Elem;
};


category P {
  ORDER   Order;
  BOOLEAN Coord;
  FS_LIST Arg;
  MOD     Mod;
  FS_LIST Mod_Elem;
};


category RP {
};
```

```
category CM {   %%% comma punctuation
};

category Q {    %%% quotation marks
  POSITION Pos;
};

category FOR {   %%% He was waiting for her to come
};

category THEP {
  COMMAS  Commas;
  FS_LIST Arg;
};

category THE {   %%% The more the cat eats the fatter it gets .
};

category MWLE {  %%% element of a multi word lexeme
};


#include "incorporation.yap"


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Restrictor Definitions        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


restrictor- NP_R(NP) {};
restrictor- NP2_R(NP) {Phon,Commas};
restrictor+ REL_R(NP) {HeadLex,Number,Person};
restrictor- REL2_R(NP) {Slash};
restrictor- REL3_R(NP) {Phon,Case,Commas};
restrictor- PP_R(PP) {};
restrictor- PP2_R(PP) {Phon,Commas};
restrictor- ADJP_R(ADJP) {};
restrictor- ADJP2_R(ADJP) {Phon,Commas};
restrictor- ADVP_R(ADVP) {};
restrictor- ADVP2_R(ADVP) {Phon,Commas};
restrictor- V_R(V) {};
restrictor- V2_R(V) {Phon};
restrictor- VP_R(VP) {};
restrictor- VP2_R(VP){Phon,Commas};
restrictor- S_R(S) {};
restrictor- S2_R(S) {Phon,Commas};
restrictor- SBAR_R(SBAR) {};
restrictor- SBAR2_R(SBAR) {Phon,Commas};
restrictor- RP_P(RP) {Phon};
restrictor- NBAR_R(NBAR) {};
restrictor- VBAR_R(VBAR) {};
restrictor- N_R(N) {};
restrictor- ADJ_R(ADJ) {};
restrictor- ADV_R(ADV) {};
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        Variable Definitions         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


NP_R    np_p,np_p2,ag,np,pat,ben;
NP2_R   np2_p;
REL_R   rnp_p;
REL2_R  rnp2_p;
REL3_R  rnp3_p;
PP_R    pp_p,pp1,pp2;
PP2_R   pp2_p;
ADJP_R  adjp_p,adjp;
ADJP2_R adjp2_p;
ADVP_R  advp_p,advp;
ADVP2_R advp2_p;
V_R     v_p;
V2_R    v2_p;
VP_R    vp_p,vp;
VP2_R   vp2_p;
S_R     s_p,s;
S2_R    s2_p;
SBAR_R  sbar_p,sbar,sbar2;
SBAR2_R sbar2_p;
RP_P    rp_p,rp;
NBAR_R  nbar_p;
VBAR_R  vbar_p;
N_R     n_p;
ADJ_R   adj_p;
ADV_R   adv_p;

FS_LIST r,r2;
NUMBER  n;
PERSON  p;
SFORM   sf;
WHFORM  wf;
BOOLEAN b;
CFORM   cf;
STRING  h,h1,h2;
VFORM   vf;
CASE    c;
DEGREE  deg;
ORDER   o;
NFORM   nf;
MOD     mod;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                    %
%  File:    templates.yap                            %
%  Purpose: Templates for my English YAP grammar     %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart   %
%                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              N                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_              : N  {NForm=(noun,propername);Coord=-;Adjunctive=-;Compound=-;};
N_sg            : N_ {Number=sg;};
N_pl            : N_ {Number=pl;};

NN_             : N_   {NForm=noun;};
NN_sg           : N_sg {NForm=noun;};
NN_pl           : N_pl {NForm=noun;};

PN_             : N_  {NForm=propername;Subcat=[];};
PN_sg           : PN_ {Number=sg;};
PN_pl           : PN_ {Number=pl;};

AN_             : N    {NForm=noun;Coord=-;Subcat=[];Compound=-;Adjunctive=+;};
AN_sg           : AN_  {Number=sg;};
AN_pl           : AN_  {Number=pl;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              ADJ                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ADJ_            : ADJ  {Numerical=-;Order=pre;Coord=-;};

ADJ_pos         : ADJ_ {Degree=pos;};
ADJ_cmp         : ADJ_ {Degree=comp;};
ADJ_sup         : ADJ_ {Degree=sup;};

ADJ_post        : ADJ  {Numerical=-;Order=post;Pred=+;Degree=pos;Coord=-;Subcat=[];};

ADJ_card        : ADJ  {Numerical=+;HeadLex="<cardinal>";Coord=-;
                        Degree=pos;Subcat=[];Mod_Elem=[*];};
ADJ_ord         : ADJ  {Numerical=+;HeadLex="<ordinal>";Order=pre;Coord=-;
                        Degree=pos;Subcat=[];Mod_Elem=[*];};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            _ADJ                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

_ADJ_pos        : _ADJ {Degree=pos;};
_ADJ_cmp        : _ADJ {Degree=comp;};
_ADJ_sup        : _ADJ {Degree=sup;};

_ADJ_attr       : _ADJ {Pred=-;};
_ADJ_pred       : _ADJ {Pred=+;Nominal=-;};

_ADJ_pos_pred   : _ADJ_pos & _ADJ_pred {};
_ADJ_cmp_pred   : _ADJ_cmp & _ADJ_pred {};
_ADJ_sup_pred   : _ADJ_sup & _ADJ_pred {};
```

```
_ADJ_pos_attr   : _ADJ_pos & _ADJ_attr {};
_ADJ_cmp_attr   : _ADJ_cmp & _ADJ_attr {};
_ADJ_sup_attr   : _ADJ_sup & _ADJ_attr {};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              ADJP                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NADJP:  ADJP {Commas=-;Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              ADV                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ADV_            : ADV {Coord=-;Not=-;};
ADV_pos         : ADV_ {Degree=pos;Mod=(-,verb,adj,adv);};
ADV_cmp         : ADV_ {Degree=comp;Mod=(-,verb);};
ADV_sup         : ADV_ {Degree=sup;Mod=(-,verb);};
ADV_verb        : ADV_ {Degree=pos;Mod=verb;};
ADV_verb_prep   : ADV_ {Degree=pos;Mod=(verb,prep);};
ADV_noun        : ADV_ {Degree=pos;Mod=noun;};
ADV_noun_prep   : ADV_ {Degree=pos;Mod=(noun,prep);};
ADV_prep        : ADV_ {Degree=pos;Mod=prep;};
ADV_adj         : ADV_ {Degree=pos;Mod=adj;};
ADV_adv         : ADV_ {Degree=pos;Mod=adv;};
ADV_adj_adv     : ADV_ {Degree=pos;Mod=(adj,adv);};
ADV_nps         : ADV_ {Degree=pos;Mod=(noun,prep,sbar);};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              ADVP                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NADVP   : ADVP {Commas=-;Coord=-;};

WHADVP  : ADVP {WhForm=quest;Not=-;Degree=pos;Commas=-; Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               V                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V_              : V {Coord=-;Aux=-;};
V_fin           : V_{VForm=fin;};
V_bse           : V_{VForm=bse;};
V_pas           : V_{VForm=pas;};
V_prp           : V_{VForm=prp;};
V_pap           : V_{VForm=pap;};

AUX             : V   {Coord=-;Aux=+;};
AUX_fin         : AUX {VForm=fin;};

BE              : AUX {HeadLex="be";};
HAVE            : AUX {HeadLex="have";};
DO              : AUX {HeadLex="do";};
```

```
WILL            : AUX {HeadLex="will";};
CAN             : AUX {HeadLex="can";};
SHALL           : AUX {HeadLex="shall";};
GET             : AUX {HeadLex="get";};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             DT                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DT_sg           : DT {WhForm=-;Number=sg;};
DT_pl           : DT {WhForm=-;Number=pl;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              P                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P_      : P {Order=pre;Arg=[*];Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              C                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

C_      : C {CForm=adj;};

NP_curr : NP {NForm=(pronoun,noun);WhForm=-;Case=acc;Slash=[];
              Adjunctive=-;Coord=-;Commas=-;NPLevel=0;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            NBAR                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NBAR_: NBAR {Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             NP                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NP_     : NP     {WhForm=(-,quest,rel);};  % not an expletive pronoun
NP_nom  : NP_    {Case=nom;};
NP_acc  : NP_    {Case=acc;};
NP_sg   : NP_nom {Number=sg;};
NP_pl   : NP_nom {Number=pl;};
NP_3    : NP_nom {Person=3rd;};
NP_3s   : NP_3 & NP_sg {};
NP_12s  : NP_sg  {Person=(1st,2nd);};

NPe_nom : NP      {Case=nom;};
NPe_sg  : NPe_nom {Number=sg;};
NPe_pl  : NPe_nom {Number=pl;};
```

```
NPe_3   : NPe_nom {Person=3rd;};
NPe_3s  : NPe_3 & NPe_sg {};
NPe_n3s : NPe_sg  {Person=(1st,2nd);};
NPe_acc : NP      {Case=acc;};

NNP     : NP  {Case=(nom,acc);Commas=-;Coord=-;};
DNP     : NNP {Elliptical=-;Adjunctive=-;NPLevel=0;Degree=pos;Slash=[];};
ENP     : NNP {Case=acc;Number=sg;Person=3rd;};


%%% NP-PRO %%%%%%%%%%%%%%%%%%%%%%%%%

RPRO    : NP   {NForm=pronoun;Elliptical=-;Adjunctive=-;Degree=pos;
                Coord=-;Commas=-;};
PRO     : RPRO {Slash=[];};

PRO_    : PRO  {WhForm=-;NPLevel=1;};
PRO_sg  : PRO_ {Number=sg;};
PRO_pl  : PRO_ {Number=pl;};
PRO_2   : PRO_ {Case=(nom,acc);Person=2nd;};
PRO_3s  : PRO_sg {Case=(nom,acc);Person=3rd;};
PRO_3p  : PRO_pl {Case=(nom,acc);Person=3rd;};

PROO_   : PRO  {WhForm=-;NPLevel=0;};
PROO_3s : PROO_ {Case=(nom,acc);Number=sg;Person=3rd;};
PROO_3p : PROO_ {Case=(nom,acc);Number=pl;Person=3rd;};

PPRO_   : PRO  {WhForm=-;NPLevel=2;};
PPRO_sg : PPRO_ {Number=sg;};
PPRO_pl : PPRO_ {Number=pl;};
PPRO_1s : PPRO_sg {Case=(nom,acc);Person=1st;};
PPRO_1p : PPRO_pl {Case=(nom,acc);Person=1st;};
PPRO_2  : PPRO_   {Case=(nom,acc);Person=2nd;};
PPRO_2s : PPRO_sg {Case=(nom,acc);Person=2nd;};
PPRO_2p : PPRO_pl {Case=(nom,acc);Person=2nd;};
PPRO_3s : PPRO_sg {Case=(nom,acc);Person=3rd;};
PPRO_3p : PPRO_pl {Case=(nom,acc);Person=3rd;};

EXPL    : PRO  {WhForm=expl;Case=nom;Person=3rd;NPLevel=2;};
NP_it   : EXPL {HeadLex="it";Number=sg;};
NP_there: EXPL {HeadLex="there";};

WHNP    : PRO  {WhForm=quest;Person=3rd;NPLevel=0;};

RELNP   : RPRO {WhForm=rel;Person=3rd;Number=n;HeadLex=h1;
                Slash=[NP{HeadLex=h1;WhForm=-;Person=3rd;Number=n;NPLevel=0;
                          Slash=[];Commas=-;}];};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             PP                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

PP_     : PP {Coord=-;Commas=-;};

PP_arg  : PP {Arg=[NP{}];Mod_Elem=[];};
PP_arg0 : PP {Arg=[];Mod_Elem=[];};
```

```
PP_by   : PP {HeadLex="by";Arg=[NP{}];Mod_Elem=[];};
PP_to   : PP {HeadLex="to";Arg=[NP{}];Mod_Elem=[];};
PP_as   : PP {HeadLex="as";Arg=[NP{}];Mod_Elem=[];};
PP_for  : PP {HeadLex="for";Arg=[NP{}];Mod_Elem=[];};


%%% PP-PRO %%%%%%%%%%%%%%%%%%%%%%%%%

WHPP    : PP_ {WhForm=quest;Order=pre;Mod=(-,verb);Arg=[NP{}];Slash=[];};

RELPP   : PP_ {WhForm=rel;Order=pre;Arg=[NP{}];
               Slash=[NP{WhForm=-;Person=3rd;Slash=[];Commas=-;NPLevel=0;}];};

PP0     : PP_ {WhForm=-;Order=pre;Arg=[NP{}];Slash=[];Mod=(-,noun,verb,prep,adj);};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             RP                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RP_to   : RP {HeadLex="to";};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             VP                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

VP_bse          : VP  {VForm=bse;};
VP_inf          : VP  {VForm=inf;};
VP_prp          : VP  {VForm=prp;};
VP_pas          : VP  {VForm=pas;};
VP_pap          : VP  {VForm=pap;};
VP_prp_inf      : VP  {VForm=(prp,inf);};
VP_prp_pas_inf  : VP  {VForm=(prp,pas,inf);};

VP_             : VP {Commas=-;Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             SBAR                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SBAR_arg        : SBAR {CForm=arg;};
SBAR_argb       : SBAR {CForm=argb;};
SBAR_argw       : SBAR {CForm=argw;};
SBAR_argwb      : SBAR {CForm=(argw,argb);};
SBAR_argx       : SBAR {CForm=(arg,argw,argb);};

SBAR_that       : SBAR_arg {Comp="that";};
SBAR_subj       : SBAR     {CForm=subj;};
SBAR_for        : SBAR_arg {Comp="for";};
SBAR_how        : SBAR_argw {Comp="how";};

SBARQ:  SBAR {CForm=quest;Slash=[];Commas=-;Coord=-;};
SARG:   SBAR {CForm=(arg,argb,argw);Slash=[];Commas=-;Coord=-;};
SARGW:  SBAR {CForm=argw;Slash=[];Commas=-;Coord=-;};
SARGB:  SBAR {CForm=argb;Slash=[];Commas=-;Coord=-;};
```

```
SADJ:   SBAR {CForm=adj;Slash=[];Commas=-;Coord=-;};
SREL:   SBAR {CForm=rel;Commas=-;Coord=-;};
SLDQ:   SBAR {CForm=argw;Commas=-;Coord=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               S                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S_      : S  {Coord=-;Commas=-;};
S_di    : S_ {SForm=(decl,imp);};
S_decl  : S_ {SForm=decl;};
S_imp   : S_ {SForm=imp;};
S_quest : S_ {SForm=quest;};
S_subj  : S_ {SForm=subj;};
S_ger   : S_ {SForm=gerund;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               _V                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

_V_1s   : _V {Infl=1s;};
_V_3s   : _V {Infl=3s;};
_V_13s  : _V {Infl=13s;};
_V_2s_pl: _V {Infl=2s_pl;};
_V_bse  : _V {Infl=bse;};
_V_past : _V {Infl=past;};
_V_prp  : _V {Infl=prp;};
_V_pap  : _V {Infl=pap;};
_V_pas  : _V {Infl=pas;};
_V_npas : _V {Infl=(1s,3s,13s,2s_pl,bse,past,prp,pap);};

_V1_1s  : _V1 {Infl=1s;};
_V1_3s  : _V1 {Infl=3s;};
_V1_13s : _V1 {Infl=13s;};
_V1_2s_pl: _V1 {Infl=2s_pl;};
_V1_bse : _V1 {Infl=bse;};
_V1_past: _V1 {Infl=past;};
_V1_prp : _V1 {Infl=prp;};
_V1_pap : _V1 {Infl=pap;};
_V1_pas : _V1 {Infl=pas;};




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                      %
%  File:    grammar.yap                                              %
%  Purpose: English grammar for the YAP parser                       %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart                   %
%                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


#include "declarations.yap"
#include "templates.yap"
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           TOP                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% declarative word order; type could be either decl, quest or imp
TOP {SForm=sf;} -> `S {SForm=decl;Slash=[];Commas=-;} SM {SForm=(decl,quest,imp)=sf;};

        %%% Wh-question
TOP {SForm=quest;} -> `SBAR {CForm=quest;Slash=[];Commas=-;} SM {SForm=quest;};

        %%% yes-no question
TOP {SForm=quest;} -> `S {SForm=quest;Slash=[];Commas=-;} SM {SForm=quest;};

        %%% imperative sentence
TOP {SForm=imp;} -> `S {SForm=imp;Slash=[];Commas=-;} SM {SForm=imp;};
        %%% for the parsing of manuals: Press the button.
TOP {SForm=imp;} -> `S {SForm=imp;Slash=[];Commas=-;} SM {SForm=decl;};


        %%% But ...
TOP {SForm=sf;}    -> C {CForm=(coord1,coord2);}
                      `S {SForm=sf;Slash=[];Commas=(-,left);}
                       SM {SForm=sf=(decl,quest,imp);};
        %%% repair rule for cases where sentence marker and sentence type don't match
TOP {SForm=sf;}    -> C {CForm=(coord1,coord2);}
                      `S {SForm=sf=(quest,imp);Slash=[];Commas=(-,left);}
                       SM {SForm=decl;};
TOP {SForm=quest;} -> C {CForm=(coord1,coord2);}
                    `SBAR {CForm=quest;Slash=[];Commas=(-,left);}
                       SM {SForm=quest;};
        %%% This gap eventually closes , but slowly .
TOP {SForm=sf;}    ->`S {SForm=sf;Slash=[];Commas=right;}=s_p
                       C {CForm=(coord1,coord2);}
                    ADVP {Mod=verb;Mod_Elem=[s_p];}
                       SM {SForm=sf=(decl,imp);};  %%% unused


        %%% Sentences with Elliptical VPs
        %%% He will VP*.
TOP {SForm=sf;} -> `S {SForm=sf;Slash=[VP{}];Commas=-;}
                   SM {SForm=(decl,quest,imp)=sf;};
        %%% But he will VP*.
TOP {SForm=sf;} ->  C {CForm=(coord1,coord2);}
                   `S {SForm=sf;Slash=[VP{}];Commas=-;}
                   SM {SForm=(decl,quest,imp)=sf;};

        %%% Remember the old days ?
TOP {SForm=quest;} -> NP* {WhForm=-;Person=2nd;Slash=[];Slash=[];Adjunctive=-;}=np_p
                     `VP_bse {Slash=[];Subcat=[np_p];Commas=-;}
                      SM {SForm=quest;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SBAR(Q)                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %%% Whom did you meet ?
SBARQ {Comp=h1;} -> NP {Phon=h1;WhForm=quest;Slash=[];Adjunctive=-;Commas=-;}=np_p
                    'S {SForm=quest;Slash=[np_p];Coord=*;Commas=-;};
SBARQ {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;}=pp_p
                    'S {SForm=quest;Slash=[pp_p];Coord=*;Commas=-;};
SBARQ {Comp=h1;} -> ADJP {Phon=h1;WhForm=quest;Pred=+;Commas=-;}=adjp_p
                    'S   {SForm=quest;Slash=[adjp_p];Coord=*;Commas=-;};
SBARQ {Comp=h1;} -> ADVP {Phon=h1;WhForm=quest;Not=-;Commas=-;}=advp_p
                    'S   {SForm=quest;Slash=[advp_p];Coord=*;Commas=-;};
        %%% adjunct Wh-clauses
SBARQ {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;Mod=verb;Mod_Elem=[s_p];}
                    'S {SForm=quest;Slash=[];Coord=*;Commas=-;}=s_p;
SBARQ {Comp=h1;} -> ADVP {Phon=h1;WhForm=quest;Not=-;Commas=-;Mod=verb;Mod_Elem=[s_p];}
                    'S   {SForm=quest;Slash=[];Coord=*;Commas=-;}=s_p;


        %%% If ..., who ...?
SBARQ {} -> SBAR {CForm=adj;Slash=[];Commas=right;} 'SBAR {Coord=*;Commas=-;};

        %%% So what did you do ?
SBARQ {} -> ADVP {WhForm=-;Not=-;Commas=(-,right);Mod=verb;Mod_Elem=[sbar_p];}
           'SBAR {Coord=*;Commas=-;}=sbar_p;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SBAR                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% indirect questions (WH-phrase is a moved argument)
SARGW {Comp=h1;} -> NP {Phon=h1;WhForm=quest;Slash=[];Adjunctive=-;Commas=-;NPLevel=0;}=np_p
                    'S {SForm=decl;Slash=[np_p];Coord=*;Commas=-;};
SARGW {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;}=pp_p
                    'S {SForm=decl;Slash=[pp_p];Coord=*;Commas=-;};
SARGW {Comp=h1;} -> ADJP {Phon=h1;WhForm=quest;Pred=+;Commas=-;}=adjp_p
                    'S   {SForm=decl;Slash=[adjp_p];Coord=*;Commas=-;};

        %%% indirect questions  (WH-phrase is an adjunct)
SARGW {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;Mod=verb;Mod_Elem=[s_p];}
                    'S {SForm=decl;Slash=[];Coord=*;Commas=-;}=s_p;
SARGW {Comp=h1;} -> ADVP {Phon=h1;WhForm=quest;Not=-;Commas=-;Mod=verb;Mod_Elem=[s_p];}
                    'S   {SForm=decl;Slash=[];Coord=*;Commas=-;}=s_p;

        %%% infinitival indirect questions
        %%% He was uncertain whether to go
SARGB {Comp=h1;} -> C {Phon=h1;CForm=argb;}
                   'VP {VForm=inf;Slash=[];Coord=*;Commas=-;};
        %%% He asked what to do
SARGW {Comp=h1;} -> NP {Phon=h1;WhForm=quest;Slash=[];Adjunctive=-;Commas=-;NPLevel=0;}=np_p
                    'VP {VForm=inf;Slash=[np_p];Coord=*;Commas=-;};
SARGW {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;}=pp_p
                    'VP {VForm=inf;Slash=[pp_p];Coord=*;Commas=-;};
SARGW {Comp=h1;} -> ADJP {Phon=h1;WhForm=quest;Pred=+;Commas=-;}=adjp_p
                    'VP  {VForm=inf;Slash=[adjp_p];Coord=*;Commas=-;};
SARGW {Comp=h1;} -> PP {Phon=h1;WhForm=quest;Slash=[];Commas=-;Mod=verb;Mod_Elem=[vp_p];}
                    'VP {VForm=inf;Slash=[];Coord=*;Commas=-;}=vp_p;
SARGW {Comp=h1;} -> ADVP {Phon=h1;WhForm=quest;Not=-;Commas=-;Mod=verb;Mod_Elem=[vp_p];}
                    'VP  {VForm=inf;Slash=[];Coord=*;Commas=-;}=vp_p;
```

```
        %%% indirect question, long distance dependency
SLDQ {Comp=h1;Slash=[np_p]=[np2_p];} ->
        NP* {WhForm=quest;Commas=-;}=np2_p
        'S  {SForm=decl;Slash=[NP{Phon=h1;}=np_p];Coord=*;Commas=-;};
SLDQ {Comp=h1;Slash=[pp_p]=[pp2_p];} ->
        PP* {WhForm=quest;Commas=-;}=pp2_p
        'S  {SForm=decl;Slash=[PP{Phon=h1;}=pp_p];Coord=*;Commas=-;};
SLDQ {Comp=h1;Slash=[adjp_p]=[adjp2_p];} ->
        ADJP* {WhForm=quest;Pred=+;Commas=-;}=adjp2_p
         'S   {SForm=decl;Slash=[ADJP{Phon=h1;}=adjp_p];Coord=*;Commas=-;};

       %%% adjunct Wh-clauses
SLDQ {Comp=h1;Slash=[PP{Phon=h1;}=pp2_p];} ->
        PP* {WhForm=quest;Commas=-;Mod=verb;Mod_Elem=[s_p];}=pp2_p
         'S {SForm=decl;Slash=[];Coord=*;Commas=-;}=s_p;
SLDQ {Comp=h1;Slash=[ADVP{Phon=h1;}=advp2_p];} ->
        ADVP* {WhForm=quest;Not=-;Mod=verb;Commas=-;Mod_Elem=[s_p];}=advp2_p
         'S   {SForm=decl;Slash=[];Coord=*;Commas=-;}=s_p;



        %%% adjunct and 'that' clauses
SBAR {Comp=h1;CForm=(adj,arg,argb)=cf;Slash=[];Commas=-;} ->
               C {Phon=h1;CForm=cf;}  'S {SForm=decl;Slash=[];Coord=*;Commas=-;};
        %%% Elliptical sentences
SBAR {Comp=h1;CForm=(adj,arg,argb)=cf;Slash=[];Commas=-;} ->
               C {Phon=h1;CForm=cf;} 'S {SForm=decl;Slash=[VP{}];Coord=*;Commas=-;};
        %%% subjunctive clauses
SBAR {Comp=h1;CForm=subj;Commas=-;} ->
               C {Phon=h1;CForm=subj;}  'S {SForm=subj;Slash=[];Coord=*;Commas=-;};
        %%% while making many more disgruntled
        %%% an option that , if exercised , would increase the value
SBAR {Comp=h1;CForm=adj;Commas=-;} ->
                 C {Phon=h1;CForm=adj;}
               'VP {VForm=(prp,pas);Subcat=[NP{}];Slash=[];Coord=*;Commas=-;};
        %%% even though ...
SBAR {} -> ADVP {Mod=sbar;Mod_Elem=[sbar_p];} 'SBAR {}=sbar_p;

        %%% relative clause - np argument
SREL {Comp=h1;Slash=[rnp_p];} ->
        NP {Phon=h1;WhForm=rel;Slash=[rnp_p];Adjunctive=-;Commas=-;NPLevel=0;}=rnp2_p
        'S {SForm=decl;Slash=[rnp2_p];Coord=*;Commas=-;};
        %%% relative clause - pp argument
SREL {Comp=h1;Slash=[rnp_p];} ->
        PP {Phon=h1;WhForm=rel;Slash=[rnp_p];Commas=-;}=pp_p
        'S {SForm=decl;Slash=[pp_p];Coord=*;Commas=-;};
        %%% relative clause - pp adjunct
SREL {Comp=h1;Slash=[rnp_p];} ->
        PP {Phon=h1;WhForm=rel;Slash=[rnp_p];Commas=-;Mod=verb;Mod_Elem=[s_p];}
        'S {SForm=decl;Slash=[];Coord=*;Commas=-;}=s_p;

        %%% rel. clause without relative pronoun
        %%% (NP (NP the movie) (SBAR WHNP-1* (S I saw yesterday NP-1*))
SREL {Comp="";Slash=[rnp_p];} ->
        RELNP* {Case=acc;Slash=[rnp_p];}=rnp2_p
```

```
         'S {SForm=decl;Slash=[NP{Slash=[];}=rnp2_p];Slash=[rnp_p];Coord=*;Commas=-;};

        %%% 'that' relative clause
SREL {Comp=h1;Slash=[rnp_p];} -> C {Phon=h1="that";CForm=rel;}
                                 'S {SForm=decl;Slash=[rnp_p];Coord=*;Commas=-;};

SBAR {CForm=arg;Comp="for";Commas=-;Coord=-;} ->
                         FOR {}
                         NP {WhForm=-;Case=acc;Slash=[];Adjunctive=-;Commas=-;}=np_p
                        'VP {VForm=inf;Subcat=[np_p];Coord=*;Commas=-;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           S                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% adjuncts, no inversion
S_decl {} -> SBAR {CForm=adj;Slash=[];Commas=(-,right);}
             'S {Coord=*;Commas=-;Slash=[];};
S_di {} -> ADVP {WhForm=-;Not=-;Commas=(-,right);Mod=verb;Mod_Elem=[s_p];}
             'S {Coord=-;Commas=-;Slash=[];}=s_p;
S_decl {} ->   PP {WhForm=-;Commas=(-,right,lr);Slash=[];Mod=verb;Mod_Elem=[s_p];}
             'S {Coord=-;Commas=-;Slash=[];}=s_p;
S_decl {} -> NP {WhForm=-;Adjunctive=+;Case=acc;Slash=[];Commas=(-,right,lr);NPLevel=0;}
             'S {Coord=-;Commas=-;Slash=[];};
S_decl {} -> VP {VForm=(inf,prp,pas);Subcat=[NP{}];Slash=[];Commas=(-,right);}
             'S {Coord=-;Commas=-;Slash=[];Slash=[];};
        %%% had it existed then , Cray Computer would have incurred loss
S_decl {} ->    S {SForm=quest;Slash=[];Coord=*;Commas=right;}
             'S {Coord=*;Commas=-;Slash=[];Slash=[];};

        %%% extrapositions
S_decl {Slash=r;} -> PP {WhForm=-;Commas=(-,right);Slash=[];}=pp_p
             'S {SForm=(decl,inv);Coord=*;Commas=-;Slash=[pp_p|r];};
S_decl {Slash=r;} -> NP {WhForm=-;Slash=[];Commas=right;Case=acc;}=np_p
             'S {Coord=*;Commas=-;Slash=[np_p|r];};
S_decl {Slash=r;} -> S {SForm=(imp,decl);Slash=[];Commas=right;}=s_p
             'S {SForm=(decl,inv);Coord=*;Commas=-;Slash=[s_p|r];};
S_decl {Slash=r;} -> ADVP {WhForm=-;Not=-;Commas=-;}=advp_p
             'S {Coord=*;Commas=-;Slash=[advp_p|r];};

        %% inverted word order
        %%% Never have I seen such a place
S_decl {SForm=decl;Slash=[];} -> ADVP {WhForm=-;Not=-;Commas=-;Mod=verb;Mod_Elem=[s_p];}
                                 'S {SForm=quest;Slash=[];Coord=*;Commas=-;}=s_p;
S_decl {SForm=decl;Slash=[];} -> ADVP {WhForm=-;Not=-;Commas=-;}=advp_p
                                 'S {SForm=quest;Slash=[advp_p];Coord=*;Commas=-;};
S_decl {SForm=decl;Slash=[];} -> PP {WhForm=-;Mod=verb;Slash=[];Commas=-;Mod_Elem=[s_p];}
                                 'S {SForm=quest;Slash=[];Coord=*;Commas=-;}=s_p;
S_decl {SForm=decl;Slash=[];} -> PP {WhForm=-;Mod=-;Slash=[];Commas=-;}=pp_p
                                 'S {SForm=quest;Slash=[pp_p];Coord=*;Commas=-;};

        %%% basic S-rule
S_decl {} -> NP {WhForm=(-,expl);Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                  'VP {VForm=fin;Subcat=[np_p];Coord=*;Commas=-;};
S_decl {} -> SBAR {CForm=(arg,argw,argb,subj);Slash=[];Commas=-;}=sbar_p
```

```
                  'VP {VForm=fin;Subcat=[sbar_p];Coord=*;Commas=-;};
S_decl {} -> VP  {VForm=(prp,inf);Subcat=[*];Slash=[];Commas=-;}=vp_p
                  'VP {VForm=fin;Subcat=[vp_p];Coord=*;Commas=-;};
        %%% They worried about (S him (VP drinking))
S_ger {}  -> NP {WhForm=(-,expl);Case=acc;Slash=[];Adjunctive=-;Commas=-;}=np_p
                 'VP {VForm=prp;Subcat=[np_p];Coord=*;Commas=-;};

        %%% imperative
S_imp {Slash=[];} -> NP* {WhForm=-;Person=2nd;Slash=[];Slash=[];Adjunctive=-;}=np_p
                 'VP {Subcat=[np_p];VForm=bse;Coord=*;Commas=-;};
S_imp {Slash=[NP{Number=n;HeadLex=h1;}];} ->
                NP* {Number=n;HeadLex=h1;WhForm=-;Person=2nd;Slash=[];}=np_p
                'VP {Subcat=[np_p];VForm=bse;Slash=[];Coord=*;Commas=-;};
        %%% Imperalist , go home!
S_imp {Slash=[];} -> NP {WhForm=-;Slash=[];Commas=right;Adjunctive=-;NPLevel=0;}=np_p
                  'S {Slash=[np_p];Coord=*;Commas=-;};
        %%% Go home, imperalists!
S_imp {Slash=[];} -> 'S {Slash=[np_p];Coord=*;Commas=-;}
                  NP {WhForm=-;Slash=[];Commas=left;Adjunctive=-;NPLevel=0;}=np_p;


        %%% relative clause with nominative rel. pronoun
        %%% The NP to which the relative clause adjoins is put on the Subcat list
S_decl {Slash=[np_p];} -> NP* {Case=nom;Commas=-;Slash=[];}=np2_p
        'VP {Subcat=[np_p];Subcat=[np2_p];VForm=fin;Slash=[];Coord=*;Commas=-;};

        %%% subjunctive clauses
S_subj {Slash=[];} -> NP {WhForm=(-,expl);Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                 'VP {VForm=bse;Subcat=[np_p];Coord=*;Commas=-;};


        %%% the ... the ... clauses
ADJP {Degree=pos;Pred=+;} -> 'ADJP {WhForm=-;Degree=comp;Coord=*;Commas=-;}
                             THEP {Arg=[S{}];};

S_decl {Slash=[];Commas=-;} -> THEP {} 'THEP {Arg=[S{}];};
S_decl {Slash=[];Commas=-;} -> 'THEP {Arg=[S{}];} THEP {Arg=[ADJP{}];};
S_decl {Slash=[];Commas=-;} -> 'THEP {Arg=[S{}];} THEP {Arg=[NP{}];};


        %%% clausal adjuncts
S_decl {} -> 'S {SForm=decl;Coord=*;Commas=-;Slash=[];}
        S {SForm=decl;Slash=[];Commas=left;};


        %%% (" Nothing has been agreed to " ,) said Donald Dion
S {SForm=inv;Slash=[S{}]=r;Coord=-;Commas=(-,right);} ->
        'VP {VForm=fin;Subcat=[np_p];Coord=*;Commas=-;Slash=r;}
         NP {WhForm=-;Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p;
        %%% Behind the silly posturing lies a real dispute .
S {SForm=inv;Slash=[PP{}]=r;Coord=-;Commas=-;} ->
        'VP {VForm=fin;Subcat=[np_p];Coord=*;Commas=-;Slash=r;}
         NP {WhForm=-;Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           THEP                    %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


THEP {Arg=[s_p];Commas=-;} -> `THE {}
               NP{WhForm=-;Degree=comp;Slash=[];Adjunctive=-;Commas=-;}=np_p
               S {SForm=decl;Slash=[np_p];Commas=-;}=s_p;
THEP {Arg=[s_p];Commas=-;} -> `THE {}
               ADVP {WhForm=-;Degree=comp;Not=-;Commas=-;}=advp_p
                  S {SForm=decl;Slash=[advp_p];Commas=-;}=s_p;
THEP {Arg=[s_p];Commas=-;} -> `THE {}
               ADJP {WhForm=-;Degree=comp;Commas=-;}=adjp_p
                  S {SForm=decl;Slash=[adjp_p];Commas=-;}=s_p;
THEP {Arg=[s_p];Commas=-;} -> `THE {}
               ADVP {WhForm=-;Degree=comp;Not=-;Commas=-;Mod=verb;Mod_Elem=[s_p];}
                  S {SForm=decl;Slash=[];Commas=-;}=s_p;
        %%% the longer the rhetoric hangs in the air , the more the divisions .
THEP {Arg=[np_p];Commas=-;} -> `THE {}
               ADJP {WhForm=-;Degree=comp;Commas=-;Pred=+;Mod_Elem=[np_p];}
                 NP {WhForm=-;Slash=[];Adjunctive=-;Commas=-;}=np_p;

THEP {Arg=[np_p];Commas=-;}   -> `THE {}
                 NP {WhForm=-;Degree=comp;Slash=[];Adjunctive=-;Commas=-;}=np_p;
THEP {Arg=[adjp_p];Commas=-;} -> `THE {}
               ADJP {WhForm=-;Degree=comp;Pred=+;Commas=-;Mod_Elem=[NP{}];}=adjp_p;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SQ                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% (SQ Did Casey (VP throw the ball))
        %%% (SBARQ (WHNP-1 What) (SQ did Casey (VP throw NP-1*)))
S {SForm=quest;Slash=r;} -> V {Aux=+;VForm=fin;}=v_p
         `S {SForm=decl;Slash=[v_p|r];Coord=*;Commas=-;};

        %%% (SBARQ (WHNP-1 Who) (SQ NP-1* (VP threw the ball)))
S {SForm=quest;Slash=[np2_p]=[np_p];} ->
        NP* {Case=nom;WhForm=quest;Commas=-;Slash=[];}=np2_p
        `VP {VForm=fin;Subcat=[np_p];Slash=[];Coord=*;Commas=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         be-SQ                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% inverted word order with "be"
        %%% Problem: Movement out of an empty VP
        %%% How big (SQ is John)
S_quest {Slash=[];} ->
                  `V {VForm=fin;HeadLex="be";Subcat=[adjp_p,np_p];}
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                ADJP {WhForm=-;Commas=-;}=adjp_p;
S_quest {Slash=[];} ->
                  `V {VForm=fin;HeadLex="be";Subcat=[sbar_p,np_p];}
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                SBAR {CForm=quest;}=sbar_p;
```

```
S_quest {Slash=[];} ->
                  'V {VForm=fin;HeadLex="be";Subcat=[np2_p,np_p];}
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                  NP {WhForm=-;Case=acc;Slash=[];Adjunctive=-;Commas=-;}=np2_p;
        %%% WHADJ
S_quest {Slash=[adjp2_p];} ->
                  'V {VForm=fin;HeadLex="be";Subcat=[adjp_p,np_p];}
                ADJP* {WhForm=quest;Commas=-;}=adjp2_p=adjp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p;
S_quest {Slash=[adjp_p]=[adjp2_p];} ->
                  'V {VForm=fin;HeadLex="be";Subcat=[adjp_p,np_p];}=v_p
                ADJP* {WhForm=quest;Commas=-;}=adjp2_p=adjp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                 ADVP {WhForm=-;Not=-;Commas=-;Mod=verb;Mod_Elem=[v_p];};
S_quest {Slash=[adjp2_p];}=s_p ->
                  'V {VForm=fin;HeadLex="be";Subcat=[adjp_p,np_p];}
                ADJP* {WhForm=quest;Commas=-;}=adjp2_p=adjp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                  PP {Slash=[];Mod=verb;Mod_Elem=[s_p];};


       %%% WHPP
S_quest {Slash=[pp2_p];} ->
                  'V {VForm=fin;HeadLex="be";Subcat=[pp_p,np_p];}
                 PP* {WhForm=(-,quest);Commas=-;Mod=-;}=pp2_p=pp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p;
S_quest {Slash=[pp2_p];} ->
                  'V {VForm=fin;HeadLex="be";Subcat=[pp_p,np_p];}=v_p
                PP* {WhForm=(-,quest);Commas=-;Mod=-;}=pp2_p=pp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                 ADVP {WhForm=-;Not=-;Commas=-;Mod=verb;Mod_Elem=[v_p];};
S_quest {Slash=[pp2_p];}=s_p ->
                  'V {VForm=fin;HeadLex="be";Subcat=[pp_p,np_p];}
                 PP* {WhForm=(-,quest);Commas=-;Mod=-;}=pp2_p=pp_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p
                  PP {Slash=[];Mod=verb;Mod_Elem=[s_p];};

        %%% inverted word order with "do"
S_quest {Slash=[advp2_p];} ->
                  'V {VForm=fin;HeadLex="do";Subcat=[advp_p,np_p];}
                ADVP* {Not=-;Commas=-;Mod=-;Mod_Elem=[];}=advp_p=advp2_p
                  NP {Case=nom;Slash=[];Adjunctive=-;Commas=-;}=np_p;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           VP                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% VBAR to VP raising rule
VP_ {Aux=-;} -> 'VBAR {Subcat=[*];};

        %%% auxiliary verbs have VP complements
VP_ {Subcat=r2;Slash=r;HeadLex=h1;} ->
         'V {Aux=+;Subcat=[vp_p|r2];Coord=*;HeadLex=*;}
         VP {Subcat=r2;Slash=r;Commas=-;HeadLex=h1;}=vp_p;
        %%% Elliptical VPs
VP {Subcat=r2;Slash=[vp2_p];HeadLex=h1;Commas=-;Coord=-;} ->
```

```
         'V {Aux=+;Subcat=[vp_p|r2];Coord=*;HeadLex=*;}
         VP* {Subcat=[NP{}]=r2;Slash=[];HeadLex=h1;}=vp_p=vp2_p;
        %%% country funds are or soon will be listed in New York or London .
VP {Slash=r;Coord=-;} ->
        'VP {Aux=+;Slash=[vp_p|r];Commas=-;Coord=+;}
         VP {Slash=r;Commas=-;}=vp_p;

        %%% auxiliary trace: Will Peter (V* hire) ?
VP_ {Subcat=r2;Slash=[v2_p];HeadLex=h1;} ->
        'V* {Aux=+;VForm=fin;Subcat=[vp_p|r2];Coord=*;HeadLex=*;}=v2_p
         VP {VForm=(bse,prp,pap,pas);Subcat=r2;Slash=[];Commas=-;HeadLex=h1;}=vp_p;
VP_ {Subcat=r2;Slash=[v2_p|r];HeadLex=h1;} ->
        'V* {Aux=+;VForm=fin;Subcat=[vp_p|r2];Coord=*;HeadLex=*;}=v2_p
         VP {VForm=(bse,prp,pap,pas);Subcat=r2;Slash=[*]=r;Commas=-;HeadLex=h1;}=vp_p;

        %%% preceding VP adjuncts
VP_ {} -> ADVP {WhForm=-;Not=-;Commas=(-,lr);Mod=verb;Mod_Elem=[vp_p];}
           'VP {Coord=*;Commas=-;}=vp_p;
VP_ {} ->   NP {WhForm=-;Adjunctive=+;Case=acc;Slash=[];Commas=(-,lr);NPLevel=0;}
           'VP {VForm=(fin,bse,prp,pap,pas);Coord=*;Commas=-;};
VP_ {} ->   PP {WhForm=-;Slash=[];Mod=verb;Commas=(-,right,lr);Mod_Elem=[vp_p];}
           'VP {VForm=(fin,bse,prp,pap,pas);Coord=*;Commas=-;}=vp_p;
VP_ {} ->   VP {VForm=pas;Subcat=[NP{}]=r;Slash=[];Commas=lr;}
           'VP {VForm=(fin,bse,prp,pap,pas);Subcat=r;Coord=*;Commas=-;};
VP_ {} -> SBAR {CForm=adj;Slash=[];Commas=lr;}
           'VP {VForm=(fin,bse,prp,pap,pas);Coord=*;Commas=-;};
VP_ {} ->   NP {NForm=pronoun;Number=pl;Person=3rd;HeadLex="both";}
           'VP {VForm=(fin,bse,prp,pap,pas);Coord=*;Commas=-;Subcat=[NP{Coord=+;}];};
        %%% negation: always in front of a non finite VP
VP_ {} -> ADVP {Not=+;Commas=-;Mod=verb;Mod_Elem=[vp_p];}
           'VP {VForm=(inf,bse,prp,pap,pas);Coord=*;Commas=-;}=vp_p;


        %%% ...is currently waiving management fees , which boosts its yield.
VP_ {Aux=-;} -> 'VP {Coord=*;Commas=-;}
        SBAR {CForm=rel;Comp="which";Slash=[NP{Number=sg;Person=3rd;}];Commas=left;};
VP_ {Aux=-;} -> 'VP {Coord=*;Commas=-;}
        NP {WhForm=-;NForm=(pronoun,noun);Case=acc;Slash=[];Adjunctive=-;Commas=left;};
VP_ {Aux=-;} -> 'VP {Subcat=[np_p];Coord=*;Commas=-;}
        ADJP {WhForm=-;Pred=+;Numerical=-;Commas=(-,left);Mod_Elem=[np_p];};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           VBAR                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        %%% Verb arguments
VBAR {Subcat=r;} ->
        'VBAR {Subcat=[np_p|r];}
        NP {Case=acc;WhForm=-;Slash=[];Adjunctive=-;Commas=-;}=np_p;
VBAR {Subcat=r;} -> 'VBAR {Subcat=[pp_p|r];}
                      PP {WhForm=-;Slash=[];Mod=-;Commas=-;}=pp_p;
VBAR {Subcat=r;} -> 'VBAR {Subcat=[adjp_p|r];}
                     ADJP {Pred=+;Commas=-;}=adjp_p;
VBAR {Subcat=r;} -> 'VBAR {Subcat=[advp_p|r];}
```

```
                    ADVP {WhForm=-;Not=-;Commas=-;Mod=-;}=advp_p;
VBAR {Subcat=r;} -> `VBAR {Subcat=[rp_p|r];}
                      RP {}=rp_p;
VBAR {Subcat=r;} -> `VBAR {Subcat=[vp_p|r];}
                      VP  {VForm=(bse,inf,prp,pas);Subcat=[*];Slash=[];Commas=-;}=vp_p;
VBAR {Subcat=r;Slash=r2;} -> `VBAR {Subcat=[vp_p|r];Slash=[];}
                      VP  {VForm=inf;Subcat=[*];Slash=[*]=r2;Commas=-;}=vp_p;
VBAR {Subcat=r;} -> `VBAR {Subcat=[s_p|r];}
                       S {SForm=(decl,imp);Slash=[];Commas=(-,left,lr);}=s_p;
VBAR {Subcat=r;} -> `VBAR {Subcat=[sbar_p|r];}
                    SBAR {CForm=(arg,argw,argb,subj);Slash=[];Commas=(-,left,lr);}=sbar_p;
        %%% The race was easy for her to win *
VBAR {Subcat=r;} -> `VBAR {Subcat=[sbar_p|r];}
                SBAR_for {Slash=[NP_{}];Commas=-;}=sbar_p;
        %%% The race was easy to win *
VBAR {Subcat=r;} -> `VBAR {Subcat=[vp_p|r];}
                  VP_inf {Subcat=[NP{NForm=noun;}];Slash=[NP_{}];Commas=-;}=vp_p;
        %%% stranded prepositions
VBAR {Subcat=r;Slash=[*]=r2;} -> `VBAR {Subcat=[pp_p|r];Slash=[];}
                       PP {Slash=r2;Mod=-;Commas=-;Mod_Elem=[];}=pp_p;
        %%% of which he appointed the president PP*
VBAR {Subcat=r;Slash=[*]=r2;} -> `VBAR {Subcat=[np_p|r];Slash=[];}
        NP {Case=acc;WhForm=-;NForm=(noun,pronoun,propername);
    Slash=r2;Adjunctive=-;Commas=-;}=np_p;

        %%% "be" arguments

        %%% VP adjuncts between arguments
VBAR {} -> `VBAR {Subcat=[*,*|*];}=vbar_p
             PP {WhForm=-;Slash=[];Mod=verb;Commas=(-,left,lr);Mod_Elem=[vbar_p];};
VBAR {} -> `VBAR {Subcat=[*,*|*];} NP {WhForm=-;Case=acc;Slash=[];Adjunctive=+;Commas=-;};
VBAR {} -> `VBAR {Subcat=[*,*|*];}=vbar_p
            ADVP {WhForm=-;Not=-;Commas=(-,left,lr);Mod=verb;Mod_Elem=[vbar_p];};

        %%% following VP adjuncts
VBAR {} -> `VBAR {Subcat=[*];}=vbar_p
             PP {WhForm=-;Slash=[];Mod=verb;Commas=(-,left,lr);Mod_Elem=[vbar_p];};
VBAR {} -> `VBAR {Subcat=[*];} NP {WhForm=-;Case=acc;Slash=[];Adjunctive=+;Commas=-;};
VBAR {} -> `VBAR {Subcat=[*];}=vbar_p
            ADVP {WhForm=-;Not=-;Commas=(-,left,lr);Mod=verb;Mod_Elem=[vbar_p];};
VBAR {} -> `VBAR {Subcat=[*];} SBAR {CForm=adj;Slash=[];Commas=(-,left,lr);};
        %%% I bought the cheap one to save money
        %%% I bought the cheap one saving money
        %%% I bought the cheap one convinced by Peter's recommendation
VBAR {} -> `VBAR {VForm=(fin,inf,bse,prp,pap);Subcat=[NP_{}]=r;}
         VP {VForm=(inf,prp,pas);Subcat=r;Slash=[];Commas=(-,lr,left);};
VBAR {} -> `VBAR {VForm=pas;Subcat=[*];}
         VP {VForm=(inf,prp);Subcat=[NP{}];Slash=[];Commas=(-,lr,left);};

        %%% Trace verb arguments
VBAR {Slash=[np2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[np_p|r];}
          NP* {Case=acc;NForm=(pronoun,propername,noun);
       Slash=[];Adjunctive=-;Commas=-;}=np_p=np2_p;
VBAR {Slash=[pp2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[pp_p|r];}
          PP* {Slash=[];Mod=-;Commas=-;}=pp_p=pp2_p;
```

```
VBAR {Slash=[sbar2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[sbar_p|r];}
        SBAR* {CForm=(arg,argw,argb);Slash=[];Commas=-;}=sbar_p=sbar2_p;
VBAR {Slash=[s2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[s_p|r];}
           S* {SForm=(decl,imp);Slash=[];}=s_p=s2_p;
VBAR {Slash=[adjp2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[adjp_p|r];}
        ADJP* {Pred=+;Commas=-;}=adjp_p=adjp2_p;
VBAR {Slash=[advp2_p];Subcat=r;} ->
        `VBAR {Slash=[];Subcat=[advp_p|r];}
        ADVP* {Not=-;Commas=-;Mod=-;Mod_Elem=[];}=advp_p=advp2_p;

        %%% long distance dependencies
VBAR {Subcat=r2;Slash=[*]=r;}-> `VBAR {Subcat=[sbar_p|r2];Slash=[];}
                                SBAR {CForm=(arg,argw,argb);Slash=r;Commas=-;}=sbar_p;
VBAR {Subcat=r2;Slash=[*]=r;} -> `VBAR {Subcat=[s_p|r2];Slash=[];}
                                   S {SForm=decl;Slash=r;Commas=-;}=s_p;


        %%% V to VBAR raising rule
VBAR {Slash=[];} -> `V {Aux=-;};


        %%% negation clitics
V {} ->   `V {Aux=+;}=v_p
        ADVP {Not=+;Commas=-;Mod=clitic;Mod_Elem=[v_p];};
V {} ->   `V {HeadLex="have";Aux=-;}=v_p
        ADVP {Not=+;Commas=-;Mod_Elem=[v_p];};
V {} ->   `V {HeadLex="be";Aux=-;}=v_p
        ADVP {Not=+;Commas=-;Mod_Elem=[v_p];};




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           NP                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% PPs are always adjoined
NNP {WhForm=wf;NForm=(noun,pronoun,propername);NPLevel=1;} ->
        `NP {WhForm=-;Coord=*;Commas=-;NPLevel=(0,1);}=np_p
         PP {WhForm=wf;Slash=[];Mod=noun;Commas=(-,left,lr);Mod_Elem=[np_p];};
NNP {WhForm=quest;NForm=(noun,pronoun,propername);NPLevel=2;} ->
        `NP {WhForm=quest;Coord=*;Commas=-;NPLevel=0;}=np_p
         PP {WhForm=-;Slash=[];Mod=noun;Commas=-;Mod_Elem=[np_p];};
        %%% after a 5 % increase the last year
NNP {NForm=noun;NPLevel=1;} ->
        `NP {WhForm=-;Coord=*;Commas=-;NPLevel=(0,1);Elliptical=-;}
         NP {WhForm=-;Adjunctive=+;Case=acc;Slash=[];Commas=-;NPLevel=0;};

        %%% of which he appointed the president
NNP {WhForm=-;NForm=noun;Slash=[pp2_p];NPLevel=2;} ->
        `NP {Coord=*;Commas=-;NPLevel=0;Slash=[];}=np_p
         PP* {WhForm=(quest,rel);Mod=noun;Commas=-;Mod_Elem=[np_p];}=pp2_p;
        %%% which he appointed the president of
NNP {WhForm=-;NForm=noun;Slash=[*]=r;NPLevel=2;} ->
```

```
        'NP {Coord=*;Commas=-;NPLevel=0;Slash=[];}=np_p
         PP {Mod=noun;Slash=r;Mod_Elem=[np_p];};

        %%% Apposition
        %%% Bill Clinton , the president of the US ,
NNP {NPLevel=1;NForm=(noun,propername);Case=c;} ->
                'NP {WhForm=-;Case=c;Elliptical=-;Coord=*;Commas=-;NPLevel=(0,1);}
                 NP {WhForm=-;Case=c;NForm=noun;WhForm=-;Commas=(lr,left);
                    Slash=[];Adjunctive=-;};
        %%% Bill Clinton himself, the question itself (reflexive pronoun)
NNP {NPLevel=1;NForm=(pronoun,noun,propername);Case=c;} ->
                'NP {WhForm=-;Number=n;Person=p;Case=c;Elliptical=-;
     Coord=*;Commas=-;NPLevel=(0,1);}
                 PPRO_3s {HeadLex="<refpro>";Number=n;Person=p;};
        %%% Pierre Vinken, 65 years old,
NNP {NForm=(noun,propername);NPLevel=1;} ->
                'NP {WhForm=-;Elliptical=-;Coord=*;Commas=-;NPLevel=(0,1);}=np_p
                ADJP{WhForm=-;Pred=+;Numerical=-;Commas=(lr,left);Mod_Elem=[np_p];};
        %%% the president designate, December 1985, Act 2, Louis XIV
NNP {NForm=(noun,propername);} ->
                'NP {Elliptical=-;Coord=*;Commas=-;NPLevel=0;}=np_p
                ADJ {Order=post;Pred=+;Coord=-;Mod_Elem=[np_p];};
        %%% nothing else
NNP {NForm=pronoun;} ->
                'NP {Elliptical=-;Coord=*;Commas=-;NPLevel=0;}=np_p
                ADJ {Order=post;Pred=+;Coord=-;Mod_Elem=[np_p];};
        %%% nothing radical
NNP {NForm=pronoun;NPLevel=2;} ->
                'NP {WhForm=-;Degree=pos;Elliptical=-;Coord=-;NPLevel=0;}=np_p
                ADJP {Pred=+;Coord=-;Numerical=-;Mod_Elem=[np_p];};
        %%% the British premier minister John Major
NNP {NForm=noun;Case=c;NPLevel=2;} ->
                'NP {WhForm=-;Case=c;Elliptical=-;Coord=*;Commas=-;NPLevel=(0,1);}
                 NP {WhForm=-;NForm=propername;Case=c;Elliptical=-;Commas=(-,lr);
                    Slash=[];Adjunctive=-;NPLevel=0;};
        %%% Westborough , Mass. ,
NNP {NPLevel=1;NForm=propername;Case=c;} ->
                'NP {WhForm=-;Case=c;Elliptical=-;Coord=*;Commas=-;NPLevel=(0,1);}
                 NP {WhForm=-;Case=c;NForm=propername;WhForm=-;Commas=(lr,left);
                    Slash=[];Adjunctive=-;Elliptical=-;};

        %%% genitive NPs
NP {Case=gen;NForm=(noun,propername);Commas=-;Coord=-;NPLevel=2;} ->
        'NP {WhForm=-;Case=(nom,acc);Coord=*;Commas=-;NPLevel=(0,1);}
         GM {};

        %%% relative clauses
NNP {NForm=(noun,pronoun,propername);NPLevel=2;} ->
  'NP {Coord=*;Commas=-;NPLevel=(0,1);}=np_p
         SBAR {CForm=rel;Slash=[np_p];Commas=(-,left,lr);};
        % reduced relative cl.: remarks concerning the case
NNP {NForm=(noun,propername);NPLevel=2;} -> 'NP {Coord=*;Commas=-;NPLevel=(0,1);}=rnp3_p
           VP {VForm=(prp,pas);Subcat=[rnp3_p];Slash=[];Commas=(-,left,lr);};
        %%% next element to be removed
NNP {NForm=noun;NPLevel=2;} -> 'NP {Coord=*;Commas=-;NPLevel=(0,1);}=rnp3_p
           VP {VForm=inf;Subcat=[rnp3_p];Slash=[];Commas=(-,left,lr);};
        %%% an odd thing to put on the list
```

```
NNP {NForm=noun;NPLevel=2;} -> ‘NP {Coord=*;Commas=-;NPLevel=(0,1);}=rnp3_p
          VP {VForm=inf;Slash=[rnp3_p];Subcat=[NP{}];Commas=-;};

        %%% basic NP rule
        %%% the pictures, the pictures of whom
NNP {Person=3rd;Slash=[];NPLevel=0;} ->
        DTP {WhForm=-;Number=n;} ‘NBAR {Number=n;Subcat=[];Coord=*;};
        %%% pictures, how many pictures, pictures of whom
NNP {Person=3rd;Slash=[];NPLevel=0;} -> %%% DTP* {WhForm=-;Number=n;}
                                ‘NBAR {Elliptical=-;Subcat=[];Coord=-;};
        %%% at Harper’s
NNP {Person=3rd;NPLevel=2;} ->
           NP {NForm=(noun,propername);WhForm=-;Case=gen;Slash=[];
               Adjunctive=-;Commas=-;}
        ‘NBAR*{WhForm=-;NForm=noun;Elliptical=+;Slash=[];Subcat=[];HeadLex="@location@";
                Number=sg;Degree=pos;};

        %%% which picture
NNP {Person=3rd;WhForm=quest;Slash=[];NPLevel=0;} ->
        DTP {WhForm=quest;Number=n;} ‘NBAR {WhForm=-;Number=n;Subcat=[];Coord=*;};

        %%% almost the whole country
NNP {NForm=(pronoun,noun);WhForm=wf;} ->
        ADVP {WhForm=wf;Commas=-;Mod=noun;Mod_Elem=[np_p];}
        ‘NP  {WhForm=-;NPLevel=0;Coord=*;Commas=-;}=np_p;


        %%% derived nouns
        %%% They discussed his writing novels
DNP {NForm=gerund;Person=3rd;Number=sg;WhForm=-;} ->
         NP {Case=gen;}=np_p
        ‘VP {VForm=prp;Subcat=[np_p];Coord=*;Commas=-;};

        %%% a million dollars
NNP {Person=3rd;Slash=[];NPLevel=0;} ->
        DTP {WhForm=-;Number=n;} M {} ‘NBAR {Number=n;Subcat=[];Coord=*;};
        %%% a million
DNP {Person=3rd;Slash=[];NPLevel=0;Number=pl;} ->
        DTP {WhForm=-;Number=sg;} ‘M {};
        %%% a further 48,000 rooms
NNP {Person=3rd;Slash=[];NPLevel=0;} ->
          DTP {WhForm=-;Number=sg;HeadLex="a";}
         ADJP {WhForm=-;Pred=-;Degree=pos;Commas=-;Mod_Elem=[nbar_p];}
        ‘NBAR {Subcat=[];Degree=pos;Number=pl;Coord=*;}=nbar_p;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           NBAR                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% adjectival adjuncts
NBAR_{} -> ADJP {WhForm=-;Pred=-;Degree=pos;Commas=-;Mod_Elem=[nbar_p];}
        ‘NBAR {Subcat=[];Coord=*;}=nbar_p;
NBAR {Degree=deg;} ->
 ADJP {WhForm=-;Pred=-;Degree=(comp,sup,as)=deg;Commas=-;Mod_Elem=[nbar_p];}
        ‘NBAR {Subcat=[];Degree=pos;Coord=*;}=nbar_p;
```

```
        %%% its LaSalle I nuclear plant
NBAR_{} -> NP {WhForm=-;NForm=propername;Case=acc;Commas=-;Slash=[];NPLevel=(0,1);}
        `NBAR {NForm=noun;Subcat=[];Coord=*;};
        %%% how many pictures
NBAR_{WhForm=quest;} -> ADJP {WhForm=quest;Pred=-;Commas=-;Mod_Elem=[nbar_p];}
                    `NBAR {WhForm=-;Subcat=[];Coord=*;}=nbar_p;

        %%% Noun arguments
        %%% belief (SBAR that the world is flat)
NBAR_{Subcat=r;} -> `NBAR {Subcat=[sbar_p|r];}  SBAR {CForm=arg;}=sbar_p;
NBAR_{Subcat=r;} -> `NBAR {Subcat=[s_p|r];}  S {SForm=decl;Commas=-;Slash=[];}=s_p;
NBAR_{Subcat=r;} -> `NBAR {Subcat=[vp_p|r];}
                       VP {Slash=[];VForm=inf;Subcat=[*];Commas=-;}=vp_p;
NBAR_{Subcat=r;} -> `NBAR {Subcat=[pp_p|r];}  PP {Slash=[];Mod=-;Mod_Elem=[];}=pp_p;
        %%% three times the expected number
NBAR_{Subcat=r;} ->  `NBAR {Subcat=[np_p|r];}
                         NP {NForm=(noun,pronoun,propername);WhForm=-;Slash=[];
                             Adjunctive=-;Commas=-;}=np_p;

        %%% the largest
NBAR_{HeadLex=h1;WhForm=wf;Case=(nom,acc);Degree=deg;} ->
          ADJP {HeadLex=h1;WhForm=wf;Pred=-;Commas=-;
Degree=deg;Numerical=-;Mod_Elem=[nbar_p];}
        `NBAR* {NForm=noun;HeadLex="@entity@";WhForm=-;Elliptical=+;Slash=[];Subcat=[];
                Adjunctive=-;Coord=*;}=nbar_p;
        %%% numerals: There were 500
NBAR_{NForm=noun;WhForm=-;Case=(nom,acc);Degree=pos;Elliptical=-;
      Number=pl;Adjunctive=-;Slash=[];Subcat=[];} ->
        `ADJP {Numerical=+;Pred=+;Coord=*;Commas=-;Mod_Elem=[NP{}];};

        %%% currency numerals: $ 5 billion
NBAR_{NForm=noun;WhForm=-;Case=(nom,acc);Degree=pos;Elliptical=-;Number=pl;
      Adjunctive=-;Slash=[];Subcat=[];} ->
        `CURR {HeadLex=h1;} ADJP {Pred=-;Numerical=+;Mod_Elem=[NBAR{HeadLex=h1;}];};
        %%% the almost $ 5 billion
NBAR_{NForm=noun;WhForm=-;Case=(nom,acc);Degree=pos;Elliptical=-;
      Adjunctive=-;Slash=[];Subcat=[];} ->
         ADVP {WhForm=-;Commas=-;Mod=noun;Mod_Elem=[];}
        `CURR {HeadLex=h1;} ADJP {Pred=-;Numerical=+;Mod_Elem=[NBAR{HeadLex=h1;}];};

        %%% N to NBAR raising rule
NBAR_{Elliptical=-;Slash=[];WhForm=-;Case=(nom,acc);Degree=pos;} ->
        `N {Coord=*;Mod_Elem=[];};

        %%% compounds
N {Compound=+;} ->
         N {NForm=propername;Compound=-;Mod_Elem=[];}
        `N {NForm=propername;HeadLex=*;Coord=-;Compound=*;};
N {Compound=+;} ->
         N {NForm=noun;Subcat=[];Mod_Elem=[n_p];}
        `N {NForm=noun;Coord=-;Compound=*;}=n_p;




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           DTP                    %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


DTP {} -> ‘DT {};
DTP {} -> PDT {Number=n;} ‘DT {Number=n;};

DTP {Number=*;} -> ‘NP {Case=gen;Commas=-;Number=*;};
DTP {Number=pl;} -> PDT {HeadLex="all";Number=pl;}
                    ‘NP {Number=*;Case=gen;NForm=pronoun;Commas=-;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            ADJP                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% arguments
ADJ {Order=*;Pred=+;Subcat=r;} ->
        ‘ADJ {Order=pre;Subcat=[pp_p|r];}
         PP{WhForm=-;Slash=[];Mod=-;Mod_Elem=[];}=pp_p;
ADJ {Order=*;Pred=+;Subcat=[];} ->
        ‘ADJ {Order=pre;Subcat=[np_p];}
         NP{WhForm=-;Slash=[];Adjunctive=+;Commas=-;}=np_p;
ADJ {Order=*;Pred=+;Subcat=[];} ->
        ‘ADJ {Order=pre;Subcat=[vp_p];}
         VP {VForm=inf;Slash=[];Subcat=[*];}=vp_p;
ADJ {Order=*;Pred=+;Subcat=[];} ->
        ‘ADJ {Order=pre;Subcat=[sbar_p];}
         SBAR {CForm=(arg,argw,argb,subj);Slash=[];Commas=-;}=sbar_p;
        %%% They were certain he would win
ADJ {Pred=+;Subcat=[];} ->
        ‘ADJ {Order=pre;Subcat=[s_p];}
         S {SForm=decl;Slash=[];Commas=-;}=s_p;
        %%% (ADJ (N New) (ADJP York-based))
ADJ {Subcat=[];} ->
         N {NForm=propername;Mod_Elem=[];}=n_p
        ‘ADJ {Order=pre;Subcat=[n_p];};
ADJ {Subcat=[];} ->
         N {NForm=propername;}=n_p CM {}
        ‘ADJ {Order=pre;Subcat=[n_p];};

        %%% The race was easy to win .
ADJ {Pred=+;Subcat=[];Mod_Elem=[rnp3_p];} ->
        ‘ADJ {Pred=+;Order=pre;Mod_Elem=[vp_p];}
         VP {VForm=inf;Slash=[rnp3_p];Subcat=[*];}=vp_p;
        %%% The race was easy for her to win .
ADJ {Pred=+;Subcat=[];Mod_Elem=[rnp3_p];} ->
        ‘ADJ {Pred=+;Order=pre;Mod_Elem=[sbar_p];}
         SBAR {CForm=arg;Slash=[rnp3_p];Commas=-;}=sbar_p;

        %%% (how) much bigger, 65 years old, three more apples
NADJP {WhForm=wf;} ->
          NP {WhForm=wf;NForm=(noun,pronoun);Case=acc;Adjunctive=-;
      Commas=-;NPLevel=0;Slash=[];}=np_p
        ‘ADJ {Order=pre;Subcat=[np_p];};

        %%% adverbial adjunct
```

```
NADJP {WhForm=wf;} -> ADVP {WhForm=(quest,-)=wf;Commas=-;Mod=adj;Mod_Elem=[adjp_p];}
                      'ADJP {WhForm=-;Coord=*;Commas=-;}=adjp_p;
        %%% It was too early to make concessions
NADJP {Degree=pos;} ->
        'ADJP {Degree=too;Pred=+;Coord=*;}
         VP {VForm=inf;Slash=[];Subcat=[*];};


        %%% ADJ to ADJP raising rule
NADJP {WhForm=-;} -> 'ADJ {Order=pre;Subcat=[];Coord=-;};

        %%% more important, as important, very important
NADJP {WhForm=-;Degree=deg;} ->
        DEGP {Degree=deg;} 'ADJ {Order=pre;Subcat=[];Degree=pos;Coord=-;};

        %%% 5 billion
NADJP {WhForm=-;} -> 'ADJ {Order=pre;Numerical=+;} M {};
        %%% 10 3/4 (Penn Treebank)
NADJP {WhForm=-;} -> 'ADJ {Order=pre;Numerical=+;} ADJ {Order=pre;Numerical=+;} M {};

        %%% between 1000 and 1500 people
ADJP {Commas=-;Coord=+;} ->         C {HeadLex=h1;CForm=coord4;}
                                 ADJP {WhForm=wf;Degree=deg;Coord=-;}
                                    C {HeadLex=h1;CForm=coord5;}
                                'ADJP {WhForm=wf;Degree=deg;Coord=-;};
        %%% 55 years old and former chairman of ...
ADJP {Pred=+;Commas=-;Coord=+;} ->
           NP {WhForm=-;Degree=pos;Case=acc;Adjunctive=-;Coord=-;Slash=[];
               NPLevel=(0,1);}
            C {CForm=coord1;}
        'ADJP {Pred=+;WhForm=-;Degree=pos;Coord=-;};
ADJP {Pred=+;Commas=-;Coord=+;} ->
        'ADJP {WhForm=-;Degree=pos;Coord=-;Coord=*;Commas=*;}
            C {CForm=coord1;}
           NP {WhForm=-;Degree=pos;Case=acc;Adjunctive=-;Coord=-;Slash=[];
               NPLevel=(0,1);Commas=-;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           M                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M {HeadLex=h1;Phon=h1;} -> 'M {HeadLex=*;} M {};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           PP                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% basic PP rule, preposition
PP_ {WhForm=wf;Slash=r;} -> 'P {Order=pre;Arg=[np_p];Coord=*;}
        NP {Commas=-;Case=acc;WhForm=wf;Slash=r;Adjunctive=-;}=np_p;
        %%% He thought about whether he wanted to go
PP_ {WhForm=-;Slash=[];} -> 'P {Mod=-;Order=pre;Arg=[sbar_p];Coord=*;}
                            SBAR_argwb {Commas=-;Slash=[];}=sbar_p;
        %%% They worried about him drinking
```

```
        %%% with interest rates rising
PP_ {WhForm=-;Slash=[];} -> ‘P {Mod=(-,verb);Order=pre;Arg=[s_p];Coord=*;}
                            S {SForm=gerund;Commas=-;Slash=[];}=s_p;
        %%% I prevented her (PP from (VP leaving the room))
PP_ {WhForm=-;Slash=[];} -> ‘P {Mod=(noun,-,verb);Order=pre;Arg=[vp_p];Coord=*;}
                            VP {VForm=prp;Commas=-;Slash=[];}=vp_p;
        %%% from a year earlier, from outside the room
PP_ {WhForm=wf;Slash=[];} -> ‘P {Order=pre;Arg=[pp_p];Coord=*;}
                            PP {WhForm=wf;Slash=[];Commas=-;Mod_Elem=[];}=pp_p;
        %%% for long, for as long as six days
PP_ {WhForm=-;Slash=[];} ->
        ‘P   {HeadLex="for";Order=pre;Arg=[adjp_p];Coord=-;}
        ADJP {HeadLex="long";Degree=(pos,as);WhForm=-;Pred=+;
      Commas=-;Mod_Elem=[NP{}];}=adjp_p;

        %%% stranded preposition
PP_ {WhForm=-;Slash=[np2_p];} -> ‘P {Order=pre;Arg=[np_p];Coord=*;}
        ENP*{Case=acc;Adjunctive=-;Elliptical=-;Slash=[];Commas=-;}=np_p=np2_p;
        %%% Where is he going to
PP_ {WhForm=-;Slash=[PP{WhForm=quest;HeadLex="where";}];} ->
         ‘P {Order=pre;Arg=[np_p];Coord=*;}
        ENP* {NForm=pronoun;HeadLex="where";WhForm=quest;}=np_p;
        %%% The place where he is going to
PP_ {WhForm=-;Slash=[PP_{WhForm=rel;HeadLex="where";Slash=[NP{HeadLex=h1;}];}];} ->
        ‘P {Order=pre;Arg=[np_p];Coord=*;}
        ENP*{NForm=pronoun;HeadLex=h1;WhForm=rel;}=np_p;

        %%% Postposition
PP_ {WhForm=wf;Slash=r;} ->
        NP {NForm=(noun,propername,pronoun);Commas=-;Case=acc;WhForm=wf;Slash=r;
            NPLevel=0;Adjunctive=-;}=np_p
        ‘P {Order=post;Arg=[np_p];Coord=*;};

        %%% Adverbial modifiers
PP_ {} -> ADVP {WhForm=-;Not=-;Degree=(pos,comp,sup);
Commas=-;Coord=-;Mod=prep;Mod_Elem=[pp_p];}
          ‘PP {}=pp_p;

        %%% from New York to Los Angeles
PP_ {HeadLex="from_to";} ->
        ‘PP {HeadLex="from";Slash=[];}
         PP {HeadLex="to";Mod_Elem=[];Slash=[];};
%%% from New York to Los Angeles to San Francisco
PP_ {HeadLex="from_to";} ->
        ‘PP {HeadLex="from_to";Slash=[];}
         PP {HeadLex="to";Mod_Elem=[];Slash=[];};
%%% to Los Angeles from New York
PP_ {HeadLex="from_to";} ->
         PP {HeadLex="to";Mod_Elem=[];Slash=[];}
        ‘PP {HeadLex="from";Slash=[];};




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           ADVP                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %%% ADV TO ADVP raising rule
NADVP {WhForm=-;} -> `ADV {Coord=*;};
NADVP {WhForm=wf;} ->
        ADVP {WhForm=wf;Commas=-;Mod=adv;Mod_Elem=[adv_p];}
        `ADV {Coord=*;}=adv_p;

        %%% as soon
NADVP {WhForm=-;Degree=deg;} -> DEGP {Degree=deg;} `ADV {Degree=pos;Coord=*;};

        %% no matter what they pay


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Comparisons                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% bigger than John
NADJP {Degree=pos;} -> `ADJP {Degree=comp;Coord=*;Commas=-;Pred=+;}
                COMP {Degree=comp;} NP {WhForm=-;Adjunctive=-;Slash=[];Commas=-;};
NADJP {WhForm=quest;Degree=pos;} -> `ADJP {WhForm=-;Degree=comp;Coord=*;Commas=-;Pred=+;}
                COMP {Degree=comp;} NP {WhForm=quest;Adjunctive=-;Slash=[];Commas=-;};
        %%% bigger than John is
NADJP {Degree=pos;} ->
        `ADJP {Degree=deg;HeadLex=h1;Coord=*;Commas=-;Pred=+;}
         COMP {Degree=(comp,as)=deg;}
        S {SForm=decl;Slash=[ADJP{Degree=pos;HeadLex=h1;Pred=+;WhForm=-;}];Commas=-;};
        %%% bigger than in the past
NADJP {Degree=pos;} ->
        `ADJP {Degree=deg;Coord=*;Commas=-;Pred=+;}
        COMP {Degree=(comp,as)=deg;}
        PP {WhForm=-;Slash=[];Commas=-;Mod=-;Mod_Elem=[];};

        %%% more men than women
NP {Degree=pos;NForm=(noun,pronoun,propername);Commas=-;} ->
         `NP {Degree=deg;Coord=*;Commas=*;}
        COMP {Degree=deg;} NP {WhForm=-;Adjunctive=-;Slash=[];Commas=-;};
        %%% more men than there were women
NP {Degree=pos;NForm=(noun,pronoun,propername);Commas=-;} ->
         `NP {Degree=deg;Coord=*;Commas=*;}
        COMP {Degree=deg;} S {SForm=decl;Slash=[];Commas=-;};
        %%% a bigger impact than in the past
NP {Degree=pos;NForm=(noun,pronoun,propername);Commas=-;} ->
         `NP {Degree=deg;Coord=*;Commas=*;}
        COMP {Degree=deg;} PP {WhForm=-;Slash=[];Commas=-;Mod=-;Mod_Elem=[];};


        %%% (as) big as John
NADJP {Degree=pos;} -> `ADJP {Degree=(pos,as);Coord=*;Commas=-;Pred=+;}
                COMP {Degree=as;} NP {WhForm=-;Adjunctive=-;Slash=[];Commas=-;};
        %%% (as) big as in Japan, (as) big as before
NADJP {Degree=pos;} -> `ADJP {Degree=(pos,as);Coord=*;Commas=-;Pred=+;}
                COMP {Degree=as;} PP {WhForm=-;Slash=[];Commas=-;Mod=-;Mod_Elem=[];};
        %%% as big as possible, higher than expected
NADJP {Degree=pos;} -> `ADJP {Degree=deg;Coord=*;Commas=-;Pred=+;}
                COMP {Degree=deg;} ADJP {WhForm=-;Degree=pos;};
        %%% big as John is
```

```
NADJP {Degree=pos;} -> ‘ADJP {Degree=(pos,as);HeadLex=h1;Coord=*;Commas=-;Pred=+;}
        COMP {Degree=as;}
        S {SForm=decl;Slash=[ADJP{Degree=pos;HeadLex=h1;Pred=+;WhForm=-;}];Commas=-;};
        %%% as big as which man was John?
NADJP {WhForm=quest;Degree=pos;} -> ‘ADJP {WhForm=-;Degree=(pos,as);Coord=*;Commas=-;Pred=+;}
                COMP {Degree=as;} NP {WhForm=quest;Adjunctive=-;Slash=[];Commas=-;};


        %%% as soon as possible
ADVP {Degree=pos;Coord=-;} -> ‘ADVP {Not=-;Degree=(pos,as);Coord=*;Commas=-;}
                COMP {Degree=as;} ADJP {WhForm=-;Degree=pos;};
ADVP {Degree=pos;} -> ‘ADVP {Not=-;Degree=as;Coord=*;Commas=-;}
                COMP {Degree=as;} NP {WhForm=-;Adjunctive=-;Slash=[];Commas=-;};
ADVP {Degree=pos;Coord=-;} -> ‘ADVP {Not=-;Degree=(pos,as);Coord=*;Commas=-;}
                COMP {Degree=as;} PP {WhForm=-;Slash=[];Commas=-;Mod=-;Mod_Elem=[];};
ADVP {Degree=pos;Coord=-;} -> ‘ADVP {Not=-;Degree=comp;Coord=*;Commas=-;}
                COMP {Degree=comp;} NP {WhForm=-;Adjunctive=-;Slash=[];Commas=-;};

        %%% big as John is
NADJP {Degree=pos;} -> ‘ADJP {Degree=(pos,as);HeadLex=h1;Coord=*;Commas=-;Pred=+;}
        COMP {Degree=as;}
        S {SForm=decl;Slash=[ADJP{Degree=pos;HeadLex=h1;Pred=+;WhForm=-;}];Commas=-;};

        %%% sales more than doubled
VBAR {Slash=[];} -> DEGP {Degree=comp;} COMP {Degree=comp;} ‘V {Aux=-;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            DEGP                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DEGP {} -> ‘DEG {};
DEGP {} -> NP {Slash=[];Adjunctive=-;WhForm=-;NForm=(noun,pronoun);Commas=-;NPLevel=0;}
         ‘DEG {Degree=comp;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            Commas                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


SBAR {Commas=lr;}     -> CM {HeadLex=h1;} ‘SBAR {Commas=-;} CM {HeadLex=h1;};
SBAR {Commas=right;}  ->       ‘SBAR {Commas=-;} CM {};
SBAR {Commas=left;}   -> CM {} ‘SBAR {Commas=-;};

S {Commas=lr;}        -> CM {HeadLex=h1;} ‘S {Commas=-;} CM {HeadLex=h1;};
S {Commas=right;}     ->       ‘S {Commas=-;} CM {};
S {Commas=left;}      -> CM {} ‘S {Commas=-;};

VP {Commas=lr;}       -> CM {HeadLex=h1;} ‘VP {Commas=-;Subcat=[*];} CM {HeadLex=h1;};
VP {Commas=right;}    ->       ‘VP {Commas=-;Subcat=[*];} CM {};
VP {Commas=left;}     -> CM {} ‘VP {Commas=-;Subcat=[*];};

ADVP {Commas=lr;}     -> CM {HeadLex=h1;} ‘ADVP {Commas=-;} CM {HeadLex=h1;};
ADVP {Commas=right;}  ->       ‘ADVP {Commas=-;} CM {};
ADVP {Commas=left;}   -> CM {} ‘ADVP {Commas=-;};
```

```
ADJP {Commas=lr;}     -> CM {HeadLex=h1;} `ADJP {Commas=-;} CM {HeadLex=h1;};
ADJP {Commas=right;}  ->       `ADJP {Commas=-;} CM {};
ADJP {Commas=left;}   -> CM {} `ADJP {Commas=-;};

PP {Commas=lr;}       -> CM {HeadLex=h1;} `PP {Commas=-;} CM {HeadLex=h1;};
PP {Commas=right;}    ->       `PP {Commas=-;} CM {};
PP {Commas=left;}     -> CM {} `PP {Commas=-;};

NP {Commas=lr;}       -> CM {HeadLex=h1;} `NP {Commas=-;} CM {HeadLex=h1;};
NP {Commas=right;}    ->       `NP {Commas=-;} CM {};
NP {Commas=left;}     -> CM {} `NP {Commas=-;};

THEP {Commas=lr;}     -> CM {HeadLex=h1;} `THEP {Commas=-;} CM {HeadLex=h1;};
THEP {Commas=right;}  ->       `THEP {Commas=-;} CM {};
THEP {Commas=left;}   -> CM {} `THEP {Commas=-;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            Quotation               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


TOP  {} -> Q {HeadLex=h1;Pos=left;} `TOP  {} Q {HeadLex=h1;Pos=right;};
SBAR {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `SBAR {Commas=-;} Q {HeadLex=h1;Pos=right;};
S    {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `S    {Commas=-;} Q {HeadLex=h1;Pos=right;};
NP   {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `NP   {Commas=-;} Q {HeadLex=h1;Pos=right;};
NBAR {} -> Q {HeadLex=h1;Pos=left;} `NBAR {} Q {HeadLex=h1;Pos=right;};
N    {} -> Q {HeadLex=h1;Pos=left;} `N    {} Q {HeadLex=h1;Pos=right;};
PP   {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `PP   {Commas=-;} Q {HeadLex=h1;Pos=right;};
P    {} -> Q {HeadLex=h1;Pos=left;} `P    {} Q {HeadLex=h1;Pos=right;};
ADVP {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `ADVP {Commas=-;} Q {HeadLex=h1;Pos=right;};
ADV  {} -> Q {HeadLex=h1;Pos=left;} `ADV  {} Q {HeadLex=h1;Pos=right;};
VP   {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `VP   {Commas=-;} Q {HeadLex=h1;Pos=right;};
VBAR {} -> Q {HeadLex=h1;Pos=left;} `VBAR {} Q {HeadLex=h1;Pos=right;};
V    {} -> Q {HeadLex=h1;Pos=left;} `V    {} Q {HeadLex=h1;Pos=right;};
ADJP {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `ADJP {Commas=-;} Q {HeadLex=h1;Pos=right;};
ADJ  {} -> Q {HeadLex=h1;Pos=left;} `ADJ  {} Q {HeadLex=h1;Pos=right;};
DTP  {} -> Q {HeadLex=h1;Pos=left;} `DTP  {} Q {HeadLex=h1;Pos=right;};
DT   {} -> Q {HeadLex=h1;Pos=left;} `DT   {} Q {HeadLex=h1;Pos=right;};
PDT  {} -> Q {HeadLex=h1;Pos=left;} `PDT  {} Q {HeadLex=h1;Pos=right;};
M    {} -> Q {HeadLex=h1;Pos=left;} `M    {} Q {HeadLex=h1;Pos=right;};
THEP {Commas=(-,lr);} -> Q {HeadLex=h1;Pos=left;} `THEP {Commas=-;} Q {HeadLex=h1;Pos=right;};
THE  {} -> Q {HeadLex=h1;Pos=left;} `THE  {} Q {HeadLex=h1;Pos=right;};
RP   {} -> Q {HeadLex=h1;Pos=left;} `RP   {} Q {HeadLex=h1;Pos=right;};
C    {} -> Q {HeadLex=h1;Pos=left;} `C    {} Q {HeadLex=h1;Pos=right;};
DEG  {} -> Q {HeadLex=h1;Pos=left;} `DEG  {} Q {HeadLex=h1;Pos=right;};
COMP {} -> Q {HeadLex=h1;Pos=left;} `COMP {} Q {HeadLex=h1;Pos=right;};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            Coordination            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% He was uncertain (SBAR whether to go or not)
```

```
SBAR {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
        `SBAR {CForm=argb;Slash=[];Coord=-;HeadLex=h1;}
            C {CForm=coord1;}
          ADVP {Not=+;Mod=verb;HeadLex=h2;};

SBAR {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         SBAR {CForm=cf;Slash=r;Coord=-;Commas=(-,right);HeadLex=h1;}
            C {CForm=coord1;}
        `SBAR {CForm=cf;Slash=r;Coord=*;Commas=-;HeadLex=h2;};
SBAR {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         SBAR {CForm=cf;Slash=r;Coord=-;Commas=(-,right);HeadLex=h1;}
            C {CForm=coord2;}
        `SBAR {CForm=cf;Slash=r;Coord=-;HeadLex=h2;};
SBAR {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         SBAR {CForm=cf;Slash=r;Coord=-;Commas=-;HeadLex=h1;}
            C {CForm=coord3;}
        `SBAR {CForm=cf;Slash=r;Coord=*;Commas=-;HeadLex=h2;};
SBAR {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            C {HeadLex=h;CForm=coord4;}
         SBAR {CForm=cf;Slash=r;Coord=-;Commas=-;HeadLex=h1;}
            C {HeadLex=h;CForm=coord5;}
        `SBAR {CForm=cf;Slash=r;Coord=*;Commas=-;HeadLex=h2;};

S {HeadLex=cat(h1,h2);Commas=-;Coord=+;} ->
            S {SForm=sf;Slash=[];Coord=-;Commas=(-,right);HeadLex=h1;}
            C {CForm=coord1;}
           `S {SForm=sf;Slash=[];Coord=*;Commas=-;HeadLex=h2;};
S {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            S {SForm=sf;Slash=[];Coord=-;HeadLex=h1;}
            C {CForm=coord2;}
           `S {SForm=sf;Slash=[];Coord=-;HeadLex=h2;};
S {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            S {SForm=sf;Slash=[];Coord=-;HeadLex=h1;}
            C {CForm=coord3;}
           `S {SForm=sf;Slash=[];Coord=*;Commas=-;HeadLex=h2;};
S {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            C {HeadLex=h;CForm=coord4;}
            S {SForm=sf;Slash=[];Coord=-;HeadLex=h1;}
            C {HeadLex=h;CForm=coord5;}
           `S {SForm=sf;Slash=[];Coord=*;Commas=-;HeadLex=h2;};

VP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
           VP {VForm=vf;Subcat=r;Slash=r2;Coord=-;HeadLex=h1;}
            C {CForm=coord1;}
          `VP {VForm=vf;Subcat=r;Slash=r2;Coord=*;Commas=-;HeadLex=h2;};
VP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
           VP {VForm=vf;Subcat=r;Slash=r2;Coord=-;HeadLex=h1;}
            C {CForm=coord2;}
          `VP {VForm=vf;Subcat=r;Slash=r2;Coord=-;HeadLex=h2;};
VP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
           VP {VForm=vf;Subcat=r;Slash=r2;Coord=-;HeadLex=h1;}
            C {CForm=coord3;}
          `VP {VForm=vf;Subcat=r;Slash=r2;Coord=+;HeadLex=h2;};
VP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            C  {HeadLex=h;CForm=coord4;}
           VP {VForm=vf;Subcat=r;Slash=r2;Coord=-;HeadLex=h1;}
            C {HeadLex=h;CForm=coord5;}
```

```
          'VP {VForm=vf;Subcat=r;Slash=r2;Coord=*;Commas=-;HeadLex=h2;};

V {Coord=+;HeadLex=cat(h1,h2);} ->
             V {VForm=vf;Aux=b;Subcat=r;Coord=-;HeadLex=h1;}
             C {CForm=coord1;}
            'V {VForm=vf;Aux=b;Subcat=r;Coord=*;HeadLex=h2;};
V {Coord=+;HeadLex=cat(h1,h2);} ->
             V {VForm=vf;Aux=b;Subcat=r;Coord=-;HeadLex=h1;}
             C {CForm=coord2;}
            'V {VForm=vf;Aux=b;Subcat=r;Coord=-;HeadLex=h2;};
V {Coord=+;HeadLex=cat(h1,h2);} ->
             V {VForm=vf;Aux=b;Subcat=r;Coord=-;HeadLex=h1;}
             C {CForm=coord3;}
            'V {VForm=vf;Aux=b;Subcat=r;Coord=+;HeadLex=h2;};
V {Coord=+;HeadLex=cat(h1,h2);} ->
             C {HeadLex=h;CForm=coord4;}
             V {VForm=vf;Aux=b;Subcat=r;Coord=-;HeadLex=h1;}
             C {HeadLex=h;CForm=coord5;}
            'V {VForm=vf;Aux=b;Subcat=r;Coord=*;HeadLex=h2;};

NP {Commas=-;Coord=+;Person=3rd;HeadLex=cat(h1,h2);} ->
            NP {WhForm=wf;Case=c;Coord=-;Slash=[];Commas=(-,right);HeadLex=h1;}
             C {HeadLex="or";CForm=coord1;}
           'NP {WhForm=wf;Case=c;Person=*;Elliptical=-;Coord=*;Commas=-;Slash=[];HeadLex=h2;};
NP {Commas=-;Number=pl;Coord=+;Person=3rd;HeadLex=cat(h1,h2);} ->
            NP {WhForm=wf;Case=c;Slash=[];Coord=-;Slash=[];Commas=(-,right);HeadLex=h1;}
             C {HeadLex="and";CForm=coord1;}
           'NP {WhForm=wf;Case=c;Number=*;Person=*;Elliptical=-;
       Coord=*;Commas=-;Slash=[];HeadLex=h2;};
NP {Commas=-;Coord=+;Person=3rd;HeadLex=cat(h1,h2);} ->
            NP {WhForm=wf;Case=c;Coord=-;Slash=[];HeadLex=h1;}
             C {CForm=coord2;}
           'NP {WhForm=wf;Case=c;Person=*;Elliptical=-;Coord=-;Slash=[];HeadLex=h2;};
NP {Commas=-;Coord=+;Person=3rd;HeadLex=cat(h1,h2);} ->
            NP {WhForm=wf;Case=c;Coord=-;Slash=[];HeadLex=h1;}
             C {CForm=coord3;}
           'NP {WhForm=wf;Case=c;Person=*;Elliptical=-;Coord=+;Slash=[];HeadLex=h2;};
NP {Commas=-;Coord=+;Person=3rd;HeadLex=cat(h1,h2);} ->
             C {HeadLex=h;CForm=coord4;}
            NP {WhForm=wf;Case=c;Coord=-;Slash=[];HeadLex=h1;}
             C {HeadLex=h;CForm=coord5;}
           'NP {WhForm=wf;Case=c;Person=*;Elliptical=-;Coord=*;Commas=-;Slash=[];HeadLex=h2;};
NP {Commas=-;Coord=+;Number=pl;Person=3rd;HeadLex=cat(h1,h2);} ->
             C {HeadLex="both_and";CForm=coord4;}
            NP {WhForm=wf;Case=c;Coord=-;Slash=[];HeadLex=h1;}
             C {HeadLex="both_and";CForm=coord5;}
           'NP {WhForm=wf;Number=*;Case=c;Person=*;Elliptical=-;
       Coord=*;Commas=-;Slash=[];HeadLex=h2;};


NBAR {Coord=+;HeadLex=cat(h1,h2);} ->
          NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Elliptical=-;
       Coord=-;Slash=[];Subcat=r;HeadLex=h1;}
             C {CForm=coord1;}
         'NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Coord=*;Slash=[];Subcat=r;HeadLex=h2;};
NBAR {Coord=+;HeadLex=cat(h1,h2);} ->
          NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Elliptical=-;
```

```
       Coord=-;Slash=[];Subcat=r;HeadLex=h1;}
            C {CForm=coord2;}
        `NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Coord=-;Slash=[];Subcat=r;HeadLex=h2;};
NBAR {Coord=+;HeadLex=cat(h1,h2);} ->
         NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Elliptical=-;
       Coord=-;Slash=[];Subcat=r;HeadLex=h1;}
            C {CForm=coord3;}
        `NBAR {WhForm=wf;Case=c;Number=n;Degree=deg;Coord=+;Slash=[];Subcat=r;HeadLex=h2;};


N {Coord=+;HeadLex=cat(h1,h2);} ->
            N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord1;HeadLex="and";}
           `N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h2;};
N {Coord=+;HeadLex=cat(h1,h2);} ->
            N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord1;HeadLex="&";}
           `N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h2;};
N {Coord=+;HeadLex=cat(h1,h2);} ->
            N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord2;}
           `N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h2;};
N {Coord=+;HeadLex=cat(h1,h2);} ->
            N {Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord3;}
           `N {Subcat=[];Coord=+;Mod_Elem=r;HeadLex=h2;};


PP {Coord=+;HeadLex=cat(h1,h2);} ->
           PP {Mod=mod;Mod_Elem=r2;Coord=-;Order=o;Slash=r;HeadLex=h1;}
             C {CForm=coord1;}
          `PP {Mod=mod;Mod_Elem=r2;Coord=*;Commas=-;Order=o;Slash=r;HeadLex=h2;};
PP {Coord=+;HeadLex=cat(h1,h2);} ->
           PP {Mod=mod;Mod_Elem=r2;Coord=-;Order=o;Slash=r;HeadLex=h1;}
             C {CForm=coord2;}
          `PP {Mod=mod;Mod_Elem=r2;Coord=-;Order=o;Slash=r;HeadLex=h2;};
PP {Coord=+;HeadLex=cat(h1,h2);} ->
           PP {Mod=mod;Mod_Elem=r2;Coord=-;Order=o;Slash=r;HeadLex=h1;}
             C {CForm=coord3;}
          `PP {Mod=mod;Mod_Elem=r2;Coord=+;Order=o;Slash=r;HeadLex=h2;};
PP {Coord=+;HeadLex=cat(h1,h2);} ->
             C {HeadLex=h;CForm=coord4;}
           PP {Mod=mod;Mod_Elem=r2;Coord=-;Order=o;Slash=r;HeadLex=h1;}
             C {HeadLex=h;CForm=coord5;}
          `PP {Mod=mod;Mod_Elem=r2;Order=o;Coord=*;Commas=-;Slash=r;HeadLex=h2;};


ADVP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         ADVP {Mod=mod;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord1;}
        `ADVP {Mod=mod;Coord=*;Commas=-;Mod_Elem=r;HeadLex=h2;};
ADVP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         ADVP {Mod=mod;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord2;}
        `ADVP {Mod=mod;Coord=-;Mod_Elem=r;HeadLex=h2;};
ADVP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         ADVP {Mod=mod;Coord=-;Mod_Elem=r;HeadLex=h1;}
```

```
            C {CForm=coord3;}
        `ADVP {Mod=mod;Coord=+;Mod_Elem=r;HeadLex=h2;};
ADVP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            C {HeadLex=h;CForm=coord4;}
         ADVP {Mod=mod;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {HeadLex=h;CForm=coord5;}
        `ADVP {Mod=mod;Coord=*;Commas=-;Mod_Elem=r;HeadLex=h2;};


ADJP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord1;}
        `ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=*;Commas=-;Mod_Elem=r;HeadLex=h2;};
ADJP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
 ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord2;}
        `ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h2;};
ADJP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
         ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord3;}
        `ADJP {WhForm=wf;Pred=b;Degree=deg;Coord=*;Commas=-;Mod_Elem=r;HeadLex=h2;};
ADJP {Commas=-;Coord=+;HeadLex=cat(h1,h2);} ->
            C {HeadLex=h;CForm=coord4;}
         ADJP {WhForm=wf;Pred=+;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {HeadLex=h;CForm=coord5;}
        `ADJP {WhForm=wf;Pred=+;Degree=deg;Coord=*;Commas=-;Mod_Elem=r;HeadLex=h2;};


ADJ {Order=pre;Coord=+;HeadLex=cat(h1,h2);} ->
          ADJ {Order=pre;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord1;}
         `ADJ {Pred=b;Degree=deg;Subcat=[];Coord=*;Mod_Elem=r;HeadLex=h2;};
ADJ {Order=pre;Coord=+;HeadLex=cat(h1,h2);} ->
          ADJ {Order=pre;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord2;}
         `ADJ {Pred=b;Degree=deg;Subcat=[];Coord=-;Mod_Elem=r;HeadLex=h2;};
ADJ {Order=pre;Coord=+;HeadLex=cat(h1,h2);} ->
          ADJ {Order=pre;Pred=b;Degree=deg;Coord=-;Mod_Elem=r;HeadLex=h1;}
            C {CForm=coord3;}
         `ADJ {Pred=b;Degree=deg;Subcat=[];Coord=+;Mod_Elem=r;HeadLex=h2;};


#include "verb-rules.yap"
#include "adjective-rules.yap"
#include "lexicon.yap"




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                     %
%  File:    verb-rules.yap                            %
%  Purpose: lexical transformation rules for the      %
%           English grammar for the YAP parser        %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart    %
%                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
V_fin {Coord=-;Subcat=[NP{Number=sg;Person=3rd;}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,NP{Number=sg;Person=3rd;}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=sg;Person=3rd;}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=sg;Person=3rd;}];} -> `_V1_3s {};

V_fin {Coord=-;Subcat=[SBAR{}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,SBAR{}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,*,SBAR{}];} -> `_V1_3s {};

V_fin {Coord=-;Subcat=[VP{}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,VP{}];} -> `_V1_3s {};
V_fin {Coord=-;Subcat=[*,*,VP{}];} -> `_V1_3s {};

V_fin {Coord=-;Subcat=[NP{Number=sg;Person=(1st,2nd);}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,NP{Number=sg;Person=(1st,2nd);}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=sg;Person=(1st,2nd);}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=sg;Person=(1st,2nd);}];} -> `_V1_bse {};

V_fin {Coord=-;Subcat=[NP{Number=pl;}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,NP{Number=pl;}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=pl;}];} -> `_V1_bse {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=pl;}];} -> `_V1_bse {};

V_fin {Coord=-;Subcat=[NP{Number=sg;Person=1st;}];} -> `_V1_1s {};
V_fin {Coord=-;Subcat=[*,NP{Number=sg;Person=1st;}];} -> `_V1_1s {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=sg;Person=1st;}];} -> `_V1_1s {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=sg;Person=1st;}];} -> `_V1_1s {};

V_fin {Coord=-;Subcat=[NP{Number=sg;Person=(1st,3rd);}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,NP{Number=sg;Person=(1st,3rd);}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=sg;Person=(1st,3rd);}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=sg;Person=(1st,3rd);}];} -> `_V1_13s {};

V_fin {Coord=-;Subcat=[SBAR{}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,SBAR{}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,*,SBAR{}];} -> `_V1_13s {};

V_fin {Coord=-;Subcat=[VP{}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,VP{}];} -> `_V1_13s {};
V_fin {Coord=-;Subcat=[*,*,VP{}];} -> `_V1_13s {};

V_fin {Coord=-;Subcat=[NP{Number=sg;Person=2nd;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,NP{Number=sg;Person=2nd;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=sg;Person=2nd;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=sg;Person=2nd;}];} -> `_V1_2s_pl {};

V_fin {Coord=-;Subcat=[NP{Number=pl;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,NP{Number=pl;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,*,NP{Number=pl;}];} -> `_V1_2s_pl {};
V_fin {Coord=-;Subcat=[*,*,*,NP{Number=pl;}];} -> `_V1_2s_pl {};

V_bse {Coord=-;} -> `_V1_bse {};
V_fin {Coord=-;} -> `_V1_past{};
V_prp {Coord=-;} -> `_V1_prp {};
V_pap {Coord=-;} -> `_V1_pap {};
V_pas {Coord=-;} -> `_V1_pas {};
```

```
%%%%%%%%%%%%%%%%%%%%%%%% DEFAULT %%%%%%%%%%%%%%%%%%%%%%%%%%%%

_V {ComlexFrame="INTRANS";} -> `_V {ComlexFrame="Default";};
_V {ComlexFrame="NP";}      -> `_V {ComlexFrame="Default";};
_V {ComlexFrame="PP";}      -> `_V {ComlexFrame="Default";};
_V {ComlexFrame="NP-PP";}   -> `_V {ComlexFrame="Default";};


%%%%%%%%%%%%%%%%%%%%%%%% INTRANS %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He went.
_V1 {Subcat=[NP_{}=ag];} ->
        `_V_npas {ComlexFrame="INTRANS";Subcat=[ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I bought the book
_V1 {Subcat=[pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP";Subcat=[pat,ag];};

% The book was bought
_V1 {Subcat=[NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP";Subcat=[pat,*];};
% The book was bought by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP";Subcat=[pat,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Peter gives Mary a book
_V1 {Subcat=[ben,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,ag];};
% Peter gives a book to Mary
_V1 {Subcat=[pat,PP_to{Arg=[ben];},NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,ag];};
% Peter gives to Mary a book he had read a long time ago
_V1 {Subcat=[PP_to{Arg=[ben];},pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,ag];};

% A book was given to Mary
_V1 {Subcat=[PP_to{Arg=[ben];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,*];};
% A book was given to Mary by Peter
_V1 {Subcat=[PP_to{Arg=[ben];},PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,ag];};
% Mary was given a book
_V1 {Subcat=[pat,ben];} ->
        `_V_pas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,*];};
% Mary was given a book by Peter
_V1 {Subcat=[pat,PP_by{Arg=[ag];},ben];} ->
        `_V_pas {ComlexFrame="NP-TO-NP";Subcat=[ben,pat,ag];};
```

```
%%%%%%%%%%%%%%%%%%%%%%% NP-FOR-NP %%%%%%%%%%%%%%%%%%%%%%%%%%

% Peter bought Mary a book
_V1 {Subcat=[ben,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-FOR-NP";Subcat=[ben,pat,ag];};
% Peter bought a book for Mary
_V1 {Subcat=[pat,PP_for{Arg=[ben];},NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-FOR-NP";Subcat=[ben,pat,ag];};
% Peter bought for Mary a book he had read a long time ago
_V1 {Subcat=[PP_for{Arg=[ben];},pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-FOR-NP";Subcat=[ben,pat,ag];};

% A book was bought for Mary
_V1 {Subcat=[ben,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-FOR-NP";Subcat=[ben,pat,*];};
% A book was bought for Mary by Peter
_V1 {Subcat=[ben,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-FOR-NP";Subcat=[ben,pat,ag];};


%%%%%%%%%%%%%%%%%%%%%%% NP-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She asked him his name
_V1 {Subcat=[ben,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-NP";Subcat=[ben,pat,ag];};

% He was asked his name
_V1 {Subcat=[pat,ben];} ->
        `_V_pas {ComlexFrame="NP-NP";Subcat=[ben,pat,*];};
% He was asked his name by Mary
_V1 {Subcat=[pat,PP_by{Arg=[ag];},ben];} ->
        `_V_pas {ComlexFrame="NP-NP";Subcat=[ben,pat,ag];};


%%%%%%%%%%%%%%%%%%%%%%% INTRANS-RECIP %%%%%%%%%%%%%%%%%%%%%%%

% They met
_V1 {Subcat=[NP{Number=pl;}=ag];} ->
        `_V_npas {ComlexFrame="INTRANS-RECIP";Subcat=[ag];};


%%%%%%%%%%%%%%%%%%%%%%% PP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They accounted for the drop in sales
_V1 {Subcat=[PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP";Subcat=[pp1,ag];};

% The drop in sales was accounted for
_V1 {Subcat=[RP{Phon=h1;HeadLex=h1;},NP{Phon=h2;WhForm=wf;}=np];} ->
        `_V_pas {ComlexFrame="PP";Subcat=[PP{Phon=cat(h1,h2);WhForm=wf;Order=pre;
 Commas=-;Coord=-;Slash=[];Mod=-;Mod_Elem=[];HeadLex=h1;Arg=[np];},*];};
% The drop in sales was accounted for by the company
_V1 {Subcat=[RP{HeadLex=h1;},PP_by{Arg=[ag];},np];} ->
        `_V_pas {ComlexFrame="PP";Subcat=[PP{HeadLex=h1;Arg=[np];},ag];};
```

```
% He got there
_V1 {Subcat=[PP_arg0{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%% PP-PP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They flew from London to Rome
_V1 {Subcat=[PP_arg{}=pp1,PP_arg{}=pp2,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-PP";Subcat=[pp1,pp2,ag];};

% They flew there from London
_V1 {Subcat=[PP_arg0{}=pp1,PP_arg{}=pp2,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-PP";Subcat=[pp1,pp2,ag];};


%%%%%%%%%%%%%%%%%%%%%%% NP-PP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She added the flowers to the bouquet
_V1 {Subcat=[pat,PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};
% She added to the bouquet the flowers she had picked yesterday
_V1 {Subcat=[PP_arg{}=pp1,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};

% Flowers were added to the bouquet
_V1 {Subcat=[PP_arg{}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-PP";Subcat=[pat,pp1,*];};
% Flowers were added to the bouquet by Mary
_V1 {Subcat=[PP_arg{}=pp1,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};


% She drop the flowers there
_V1 {Subcat=[pat,PP_arg0{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};
% She dropped there the flowers she had picked yesterday
_V1 {Subcat=[PP_arg0{}=pp1,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};

% Flowers were dropped there
_V1 {Subcat=[PP_arg0{}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-PP";Subcat=[pat,pp1,*];};
% Flowers were dropped there by Mary
_V1 {Subcat=[PP_arg0{}=pp1,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-PP";Subcat=[pat,pp1,ag];};



%%%%%%%%%%%%%%%%%%%%%%% S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They thought he was always late
_V1 {Subcat=[s,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="S";Subcat=[s,ag];};
% They thought that he was always late
_V1 {Subcat=[SBAR_that{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="S";Subcat=[sbar,ag];};
```

```
% That he was late was anticipated
_V1 {Subcat=[SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="S";Subcat=[sbar,*];};
% That he was late was anticipated by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="S";Subcat=[sbar,ag];};
% It was anticipated that he was late
_V1 {Subcat=[SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="S";Subcat=[sbar,*];};
% It was anticipated by Mary that he was late
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="S";Subcat=[sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% THAT-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He complained that they were coming
_V1 {Subcat=[SBAR_that{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="THAT-S";Subcat=[sbar,ag];};

% That they were coming was accepted
_V1 {Subcat=[SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="THAT-S";Subcat=[sbar,*];};
% That they were coming was accepted by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="THAT-S";Subcat=[sbar,ag];};
% It was accepted that they were coming
_V1 {Subcat=[SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="THAT-S";Subcat=[sbar,*];};
% It was accepted by Peter that they were coming
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="THAT-S";Subcat=[sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% TO-INF-SC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I wanted to come
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="TO-INF-SC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% TO-INF-AC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He helped to save the child
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="TO-INF-AC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% TO-INF-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He seemed to wilt
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_npas {ComlexFrame="TO-INF-RS";Subcat=[vp];};


%%%%%%%%%%%%%%%%%%%%%%%% WH-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% He asked whether he should come
_V1 {Subcat=[SBAR_argwb{}=sbar,NP_{}=ag];} ->
        ‘_V_npas {ComlexFrame="WH-S";Subcat=[sbar,ag];};

% Whether he should come was asked
_V1 {Subcat=[SBAR_argwb{}=sbar];} ->
        ‘_V_pas {ComlexFrame="WH-S";Subcat=[sbar,*];};
% Whether he should come was asked by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_argwb{}=sbar];} ->
        ‘_V_pas {ComlexFrame="WH-S";Subcat=[sbar,ag];};
% It was asked whether he should come
_V1 {Subcat=[SBAR_argwb{}=sbar,NP_it{}];} ->
        ‘_V_pas {ComlexFrame="WH-S";Subcat=[sbar,*];};
% It was asked by Peter whether he should come
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_argwb{}=sbar,NP_it{}];} ->
        ‘_V_pas {ComlexFrame="WH-S";Subcat=[sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% HOW-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He explained how she did it
_V1 {Subcat=[SBAR_how{}=sbar,NP_{}=ag];} ->
        ‘_V_npas {ComlexFrame="HOW-S";Subcat=[sbar,ag];};

% How she did it was explained
_V1 {Subcat=[SBAR_how{}=sbar];} ->
        ‘_V_pas {ComlexFrame="HOW-S";Subcat=[sbar,*];};
% How she did it was explained by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_how{}=sbar];} ->
        ‘_V_pas {ComlexFrame="HOW-S";Subcat=[sbar,ag];};
% It was explained how she did it
_V1 {Subcat=[SBAR_how{}=sbar,NP_it{}];} ->
        ‘_V_pas {ComlexFrame="HOW-S";Subcat=[sbar,*];};
% It was explained by Peter how she did it
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_how{}=sbar,NP_it{}];} ->
        ‘_V_pas {ComlexFrame="HOW-S";Subcat=[sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-HOW-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%

% He explained how she did it
_V1 {Subcat=[SBAR_how{}=sbar,NP_{}=ag];} ->
        ‘_V_npas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,*,ag];};
% He explained to them how she did it
_V1 {Subcat=[PP_arg{}=pp1,SBAR_how{}=sbar,NP_{}=ag];} ->
        ‘_V_npas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,pp1,ag];};

% How she did it was later explained
_V1 {Subcat=[SBAR_how{}=sbar];} ->
        ‘_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,*,*];};
% How she did it was explained by Peter
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_how{}=sbar];} ->
        ‘_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,*,ag];};
% It was explained how she did it
_V1 {Subcat=[SBAR_how{}=sbar,NP_it{}];} ->
        ‘_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,*,*];};
```

```
% It was explained by Peter how she did it
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_how{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,*,ag];};

% How she did it was later explained to them
_V1 {Subcat=[PP_arg{}=pp1,SBAR_how{}=sbar];} ->
        `_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,pp1,*];};
% How she did it was explained to them by Peter
_V1 {Subcat=[PP_arg{}=pp1,PP_by{Arg=[ag];},SBAR_how{}=sbar];} ->
        `_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,pp1,ag];};
% It was explained to them how she did it
_V1 {Subcat=[PP_arg{}=pp1,SBAR_how{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-HOW-TO-INF";Subcat=[sbar,pp1,*];};


%%%%%%%%%%%%%%%%%%%%%%%% ING-SC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She abandoned drinking
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="ING-SC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% BE-ING-SC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She began drinking
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="BE-ING-SC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% POSSING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He discussed writing novels
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="POSSING";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% ING-NP-OMIT %%%%%%%%%%%%%%%%%%%%%%%%%%

% His hair needs combing
_V1 {Subcat=[VP_prp{Slash=[rnp3_p];}=vp,NP_{}=rnp3_p=ag];} ->
        `_V_npas {ComlexFrame="ING-NP-OMIT";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% S-SUBJUNCT %%%%%%%%%%%%%%%%%%%%%%%%%%%

% She demanded that he leave immediately
_V1 {Subcat=[SBAR_subj{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="S-SUBJUNCT";Subcat=[sbar,ag];};

% That he leave immediately was demanded
_V1 {Subcat=[SBAR_subj{}=sbar];} ->
        `_V_pas {ComlexFrame="S-SUBJUNCT";Subcat=[sbar,*];};
% It was demanded that he leave immediately
_V1 {Subcat=[SBAR_subj{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="S-SUBJUNCT";Subcat=[sbar,*];};

% That he leave immediately was demanded by Peter
```

```
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_subj{}=sbar];} ->
        `_V_pas {ComlexFrame="S-SUBJUNCT";Subcat=[sbar,ag];};
% It was demanded by Peter that he leave immediately
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_subj{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="S-SUBJUNCT";Subcat=[sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I kept them laughing
_V1 {Subcat=[np,VP_prp{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ING";Subcat=[vp,ag];};

% They were kept laughing
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_pas {ComlexFrame="NP-ING";Subcat=[vp,*];};
% They were kept laughing by him
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[np];}=vp,PP_by{Arg=[ag];},np];} ->
        `_V_pas {ComlexFrame="NP-ING";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ING-OC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I caught him stealing
_V1 {Subcat=[pat,VP_prp{Slash=[];Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ING-OC";Subcat=[pat,vp,ag];};

% He was caught stealing
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-OC";Subcat=[pat,vp,*];};

% He was caught stealing by Peter
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[pat];}=vp,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-OC";Subcat=[pat,vp,ag];};
% He was caught by Peter stealing a book
_V1 {Subcat=[PP_by{Arg=[ag];},VP_prp{Slash=[];Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-OC";Subcat=[pat,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ING-SC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He combed the woods looking for her
_V1 {Subcat=[pat,VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ING-SC";Subcat=[pat,vp,ag];};

% The woods were combed looking for her
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-SC";Subcat=[pat,vp,ag];};

% The woods were combed by Peter looking for her
_V1 {Subcat=[PP_by{Arg=[ag];},VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-SC";Subcat=[pat,vp,ag];};
% The woods were combed looking for her by Peter
_V1 {Subcat=[VP_prp{Slash=[];Subcat=[ag];}=vp,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ING-SC";Subcat=[pat,vp,ag];};
```

```
%%%%%%%%%%%%%%%%%%%%%%%% NP-P-ING %%%%%%%%%%%%%%%%%%%%%%%%

% I prevented her from leaving
_V1 {Subcat=[np,PP{Arg=[VP_prp{Slash=[];Subcat=[np];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-ING";Subcat=[pp1,ag];};

% She was prevented from leaving
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[np];}];Mod_Elem=[];}=pp1,np];} ->
        `_V_pas {ComlexFrame="NP-P-ING";Subcat=[pp1,*];};
% She was prevented from leaving by this
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[np];}];Mod_Elem=[];}=pp1,PP_by{Arg=[ag];},np];} ->
        `_V_pas {ComlexFrame="NP-P-ING";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-P-ING-OC %%%%%%%%%%%%%%%%%%%%%%%%

% I accused her of murdering her husband
_V1 {Subcat=[pat,PP{Arg=[VP_prp{Slash=[];Subcat=[pat];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-ING-OC";Subcat=[pat,pp1,ag];};

% She was accused of murdering her husband
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[pat];}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-OC";Subcat=[pat,pp1,*];};

% She was accused by the jury of murdering her husband
_V1 {Subcat=[PP_by{Arg=[ag];},PP{Arg=[VP_prp{Slash=[];Subcat=[pat];}];
     Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-OC";Subcat=[pat,pp1,ag];};
% She was accused of murdering by the jury
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[pat];}];
     Mod_Elem=[];}=pp1,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-OC";Subcat=[pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-P-ING-SC %%%%%%%%%%%%%%%%%%%%%%%%

% He wasted time on fuzzing with his hair
_V1 {Subcat=[pat,PP{Arg=[VP_prp{Slash=[];Subcat=[ag];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-ING-SC";Subcat=[pat,pp1,ag];};

% Time was wasted on fuzzing with ones hair
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[ag];}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-SC";Subcat=[pat,pp1,ag];};
% Time was wasted by him on fuzzing with his hair
_V1 {Subcat=[PP_by{Arg=[ag];},PP{Arg=[VP_prp{Slash=[];Subcat=[ag];}];
     Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-SC";Subcat=[pat,pp1,ag];};
% Time was wasted on fuzzing by him
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[ag];}];
     Mod_Elem=[];}=pp1,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-ING-SC";Subcat=[pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% FOR-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%

% I prefer for her to do it
_V1 {Subcat=[PP_for{Arg=[np];},VP_inf{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
```

```
        `_V_npas {ComlexFrame="FOR-TO-INF";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%

% I want John to go
_V1 {Subcat=[np,VP_inf{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-INF";Subcat=[vp,ag];};

% John was allowed  to go
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF";Subcat=[vp,*];};
% John was allowed by Peter to go
_V1 {Subcat=[PP_by{Arg=[ag];},VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF";Subcat=[vp,ag];};
% John was allowed to go by Peter
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[np];}=vp,PP_by{Arg=[ag];},np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-INF-OC %%%%%%%%%%%%%%%%%%%%%%%%

% I advised Mary to go
_V1 {Subcat=[pat,VP_inf{Slash=[];Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-INF-OC";Subcat=[pat,vp,ag];};

% Mary was advised  to go
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-OC";Subcat=[vp,*];};
% Mary was advised by Peter to go
_V1 {Subcat=[PP_by{Arg=[ag];},VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-OC";Subcat=[vp,ag];};
% Mary was advised to go by Peter
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[np];}=vp,PP_by{Arg=[ag];},np];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-OC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-INF-SC %%%%%%%%%%%%%%%%%%%%%%%%

% John promised Mary to resign
_V1 {Subcat=[pat,VP_inf{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-INF-SC";Subcat=[pat,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-INF-VC %%%%%%%%%%%%%%%%%%%%%%%%

% They badgered him to go
_V1 {Subcat=[pat,VP_inf{Slash=[];Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-INF-VC";Subcat=[pat,vp,ag];};

% He was badgered to go
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[pat];}=vp,pat];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-VC";Subcat=[pat,vp,*];};
% He was badgered by her to go
_V1 {Subcat=[PP_by{Arg=[ag];},VP_inf{Slash=[];Subcat=[pat];}=vp,pat];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-VC";Subcat=[pat,vp,ag];};
% He was badgered to go by her
```

```
_V1 {Subcat=[VP_inf{Slash=[];Subcat=[pat];}=vp,PP_by{Arg=[ag];},pat];} ->
        `_V_pas {ComlexFrame="NP-TO-INF-VC";Subcat=[pat,vp,ag];};



%%%%%%%%%%%%%%%%%%%%%%%% NP-TOBE %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I found him to be a good doctor
_V1 {Subcat=[pat,VP_inf{Slash=[];HeadLex="be";Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,ag];};

% He was found to be a good doctor
_V1 {Subcat=[VP_inf{Slash=[];HeadLex="be";Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,*];};
% He was found by her to be a good doctor
_V1 {Subcat=[PP_by{Arg=[ag];},VP_inf{Slash=[];HeadLex="be";Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,ag];};


% I found him to have a good voice
_V1 {Subcat=[pat,VP_inf{Slash=[];HeadLex="have";Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,ag];};

% He was found to have a good voice
_V1 {Subcat=[VP_inf{Slash=[];HeadLex="have";Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,*];};
% He was found by her to have a good voice
_V1 {Subcat=[PP_by{Arg=[ag];},VP_inf{Slash=[];HeadLex="have";Subcat=[pat];}=vp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-TOBE";Subcat=[pat,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-INF %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He had her sing
_V1 {Subcat=[np,VP_bse{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-INF";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-INF-OC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He helped her bake the cake
_V1 {Subcat=[pat,VP_bse{Slash=[];Subcat=[pat];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-INF-OC";Subcat=[pat,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% INF-AC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He helped bake the cake
_V1 {Subcat=[VP_bse{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="INF-AC";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He told the audience he was leaving
_V1 {Subcat=[pat,s,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-S";Subcat=[pat,s,ag];};
```

```
% He told the audience that he was leaving
_V1 {Subcat=[ben,sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-S";Subcat=[ben,sbar,ag];};

% The audience was told that he was leaving
_V1 {Subcat=[sbar,ben];} ->
        `_V_pas {ComlexFrame="NP-S";Subcat=[ben,sbar,*];};
% The audience was told by the conductor that he was leaving
_V1 {Subcat=[PP_by{Arg=[ag];},sbar,ben];} ->
        `_V_pas {ComlexFrame="NP-S";Subcat=[ben,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-WH-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They asked him whether he was going
_V1 {Subcat=[pat,SBAR_argwb{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-WH-S";Subcat=[pat,sbar,ag];};

% He was asked whether he was going
_V1 {Subcat=[SBAR_argwb{}=sbar,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-WH-S";Subcat=[pat,sbar,*];};
% He was asked by her whether he was going
_V1 {Subcat=[PP_by{Arg=[ag];},SBAR_argwb{}=sbar,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-WH-S";Subcat=[pat,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-TO-INF-NP-OMIT %%%%%%%%%%%%%%%%%%%%

% He has good things to eat.
% He is the man to beat
_V1 {Subcat=[rnp3_p,VP_inf{Slash=[rnp3_p];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-TO-INF-NP-OMIT";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-ING-SC %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They failed in attempting the climb
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[ag];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-ING-SC";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-ING-PRED %%%%%%%%%%%%%%%%%%%%%%%%%%%

% It is for banning bullets
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];Subcat=[NP_{}];}];Mod_Elem=[];}=pp1|r];} ->
        `_V_npas {ComlexFrame="P-ING-PRED";Subcat=[pp1,*]=[*|r];};


%%%%%%%%%%%%%%%%%%%%%%%% P-POSSING %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He disapproved of killing the peasants
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-POSSING";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-NP-ING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% They worried about him drinking
_V1 {Subcat=[PP{Arg=[S_ger{}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-NP-ING";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-WH-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He thought about whether/how he wanted to go
_V1 {Subcat=[PP{Arg=[SBAR_argwb{}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-WH-S";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-NP-TO-INF-OC %%%%%%%%%%%%%%%%%%%%%%%

% He beckoned to him to come
_V1 {Subcat=[PP{Arg=[np];Mod_Elem=[];}=pp1,VP_inf{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-NP-TO-INF-OC";Subcat=[pp1,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-NP-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%%%

% He counted on him to come
_V1 {Subcat=[PP{Arg=[np];Mod_Elem=[];}=pp1,VP_inf{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-NP-TO-INF";Subcat=[pp1,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% P-NP-TO-INF-VC %%%%%%%%%%%%%%%%%%%%%%%

% She appealed to him to go
_V1 {Subcat=[PP{Arg=[np];Mod_Elem=[];}=pp1,VP_inf{Slash=[];Subcat=[np];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="P-NP-TO-INF-VC";Subcat=[pp1,vp,ag];};

% object control ignored (less frequent): She appealed to him to be freed


%%%%%%%%%%%%%%%%%%%%%%%% POSSING-PP %%%%%%%%%%%%%%%%%%%%%%%%%%%

% They limited smoking a pipe to the lounge
_V1 {Subcat=[VP_prp{Slash=[];}=vp,PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="POSSING-PP";Subcat=[vp,pp1,ag];};
% They limited to the lounge smoking a pipe
_V1 {Subcat=[PP_arg{}=pp1,VP_prp{Slash=[];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="POSSING-PP";Subcat=[vp,pp1,ag];};

% Smoking a pipe was limited to the lounge
_V1 {Subcat=[PP_arg{}=pp1,VP_prp{Slash=[];}=vp];} ->
        `_V_pas {ComlexFrame="POSSING-PP";Subcat=[vp,pp1,*];};
% Smoking a pipe was limited to the lounge by the management
_V1 {Subcat=[PP_arg{}=pp1,PP_by{Arg=[ag];},VP_prp{Slash=[];}=vp];} ->
        `_V_pas {ComlexFrame="POSSING-PP";Subcat=[vp,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-THAT-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They admitted to the authorities that they had entered illegally
_V1 {Subcat=[PP_arg{}=pp1,SBAR_that{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-THAT-S";Subcat=[pp1,sbar,ag];};
```

```
% They admitted to the authorities they had entered illegally
_V1 {Subcat=[PP_arg{}=pp1,s,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-THAT-S";Subcat=[pp1,s,ag];};

% That they had entered illegally was admitted to the authorities
_V1 {Subcat=[PP_arg{}=pp1,SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="PP-THAT-S";Subcat=[pp1,sbar,*];};
% It was admitted to the authorities that they had entered illegally
_V1 {Subcat=[PP_arg{}=pp1,SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-THAT-S";Subcat=[pp1,sbar,*];};

% That they had entered illegally was admitted by them to the authorities
_V1 {Subcat=[PP_by{Arg=[ag];},PP_arg{}=pp1,SBAR_that{}=sbar];} ->
        `_V_pas {ComlexFrame="PP-THAT-S";Subcat=[pp1,sbar,ag];};
% It was admitted by them to the authorities that they had entered illegally
_V1 {Subcat=[PP_by{Arg=[ag];},PP_arg{}=pp1,SBAR_that{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-THAT-S";Subcat=[pp1,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-WH-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They asked of everybody whether they had enrolled
_V1 {Subcat=[PP_arg{}=pp1,SBAR_argwb{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-WH-S";Subcat=[pp1,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-P-WH-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I agreed with him about whether he should kill the peasants
_V1 {Subcat=[PP_arg{}=pp1,PP{Arg=[SBAR_argwb{}];Mod_Elem=[];}=pp2,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-P-WH-S";Subcat=[pp1,pp2,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-THAT-S-SUBJUNCT %%%%%%%%%%%%%%%%%%%%

% They suggested to him that he go
_V1 {Subcat=[PP_arg{}=pp1,SBAR_subj{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-THAT-S-SUBJUNCT";Subcat=[pp1,sbar,ag];};

% It was suggested to him that he go
_V1 {Subcat=[PP_arg{}=pp1,SBAR_subj{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-THAT-S-SUBJUNCT";Subcat=[pp1,sbar,*];};
% It was suggested to him by Peter that he go
_V1 {Subcat=[PP_arg{}=pp1,PP_by{Arg=[ag];},SBAR_subj{}=sbar,NP_it{}];} ->
        `_V_pas {ComlexFrame="PP-THAT-S-SUBJUNCT";Subcat=[pp1,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-TO-INF-RS %%%%%%%%%%%%%%%%%%%%%%%%%

% He appeared to her to be ill
_V1 {Subcat=[PP_arg{}=pp1,VP_inf{Slash=[];Subcat=[np];}=vp,np];} ->
        `_V_npas {ComlexFrame="PP-TO-INF-RS";Subcat=[pp1,vp];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-FOR-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%%

% They contracted with him for the man to go
```

```
_V1 {Subcat=[PP_arg{}=pp1,SBAR_for{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-FOR-TO-INF";Subcat=[pp1,sbar,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-P-WH-S %%%%%%%%%%%%%%%%%%%%%%%%

% They made a great fuss about whether they should participate
_V1 {Subcat=[pat,PP{Arg=[SBAR_argwb{}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-WH-S";Subcat=[pat,pp1,ag];};

% A great fuss was made about whether they should participate
_V1 {Subcat=[PP{Arg=[SBAR_argwb{}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-WH-S";Subcat=[pat,pp1,*];};
% A great fuss was made by him about whether they should participate
_V1 {Subcat=[PP_by{Arg=[ag];},PP{Arg=[SBAR_argwb{}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-WH-S";Subcat=[pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-P-POSSING %%%%%%%%%%%%%%%%%%%%%%%%

% She told him about climbing Everest
_V1 {Subcat=[pat,PP{Arg=[VP_prp{Slash=[];}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-POSSING";Subcat=[pat,pp1,ag];};

% He was told about climbing Everest
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];}];Mod_Elem=[];}=pp1,pat];} ->
        `_V_pas {ComlexFrame="NP-P-POSSING";Subcat=[pat,pp1,*];};
% He was told by her about climbing Everest
_V1 {Subcat=[PP{Arg=[VP_prp{Slash=[];}];Mod_Elem=[];}=pp1,PP_by{Arg=[ag];},pat];} ->
        `_V_pas {ComlexFrame="NP-P-POSSING";Subcat=[pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-P-NP-ING %%%%%%%%%%%%%%%%%%%%%%%%

% He attributed his failure to noone buying his books
_V1 {Subcat=[pat,PP{Arg=[S_ger{}];Mod_Elem=[];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-P-NP-ING";Subcat=[pat,pp1,ag];};

% His failure was attributed to noone buying his books
_V1 {Subcat=[PP{Arg=[S_ger{}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-NP-ING";Subcat=[pat,pp1,*];};
% His failure was attributed by him to noone buying his books
_V1 {Subcat=[PP_by{Arg=[ag];},PP{Arg=[S_ger{}];Mod_Elem=[];}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-P-NP-ING";Subcat=[pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She gave up
_V1 {Subcat=[rp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART";Subcat=[rp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART-ING-SC %%%%%%%%%%%%%%%%%%%%%%%%

% He rules out paying her debts
_V1 {Subcat=[rp,VP_prp{Slash=[];Subcat=[ag];}=vp,NP_{}=ag];} ->
```

```
        `_V_npas {ComlexFrame="PART-ING-SC";Subcat=[rp,vp,ag];};

% Paying her debts was ruled out
_V1 {Subcat=[rp,VP_prp{Slash=[];Subcat=[ag];}=vp];} ->
        `_V_pas {ComlexFrame="PART-ING-SC";Subcat=[rp,vp,ag];};
% Paying her debts was ruled out by her
_V1 {Subcat=[rp,PP_by{Arg=[ag];},VP_prp{Slash=[];Subcat=[ag];}=vp];} ->
        `_V_pas {ComlexFrame="PART-ING-SC";Subcat=[rp,vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I looked up the entry
_V1 {Subcat=[rp,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-NP";Subcat=[rp,pat,ag];};
% I looked the entry up
_V1 {Subcat=[pat,rp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-NP";Subcat=[rp,pat,ag];};

% The entry was looked up
_V1 {Subcat=[rp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="PART-NP";Subcat=[rp,pat,*];};
% The entry was looked up by the program
_V1 {Subcat=[rp,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="PART-NP";Subcat=[rp,pat,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART-PP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She looked in on her friend
_V1 {Subcat=[rp,PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-PP";Subcat=[rp,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART-NP-PP %%%%%%%%%%%%%%%%%%%%%%%%%%

% I separated out the three boys from the crowd
_V1 {Subcat=[rp,pat,PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-NP-PP";Subcat=[rp,pat,pp1,ag];};
% I separated the three boys out from the crowd
_V1 {Subcat=[pat,rp,PP_arg{}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-NP-PP";Subcat=[rp,pat,pp1,ag];};

% The three boys were separated out from the crowd
_V1 {Subcat=[rp,PP_arg{}=pp1,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="PART-NP-PP";Subcat=[rp,pat,pp1,*];};
% The three boys were separated out from the crowd by the teacher
_V1 {Subcat=[rp,PP_arg{}=pp1,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="PART-NP-PP";Subcat=[rp,pat,pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PART-THAT-S %%%%%%%%%%%%%%%%%%%%%%%%%

% They figured out that she had n't done her job
_V1 {Subcat=[rp,SBAR_argx{}=sbar,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-THAT-S";Subcat=[rp,sbar,ag];};
% They figured out she had n't done her job
```

```
_V1 {Subcat=[rp,S{}=s,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PART-THAT-S";Subcat=[rp,s,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-NP-PRED %%%%%%%%%%%%%%%%%%%%%%%%

% They appointed him professor
_V1 {Subcat=[pat,np,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-NP-PRED";Subcat=[pat,np,ag];};

% He was appointed professor
_V1 {Subcat=[np,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-NP-PRED";Subcat=[pat,np,*];};
% He was appointed professor by the committee
_V1 {Subcat=[np,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-NP-PRED";Subcat=[pat,np,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ADJP-PRED %%%%%%%%%%%%%%%%%%%%%%%%

% She considered him foolish
_V1 {Subcat=[np,ADJP{Mod_Elem=[np];}=adjp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ADJP-PRED";Subcat=[adjp,ag];};

% He was considered foolish
_V1 {Subcat=[ADJP{Mod_Elem=[np];}=adjp,NP_{}=np];} ->
        `_V_pas {ComlexFrame="NP-ADJP-PRED";Subcat=[adjp,*];};
% He was considered foolish by Mary
_V1 {Subcat=[ADJP{Mod_Elem=[np];}=adjp,PP_by{Arg=[ag];},NP_{}=np];} ->
        `_V_pas {ComlexFrame="NP-ADJP-PRED";Subcat=[adjp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ADVP-PRED %%%%%%%%%%%%%%%%%%%%%%%%

% They mistakenly thought him here
_V1 {Subcat=[np,ADVP{Mod_Elem=[np];}=advp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ADVP-PRED";Subcat=[advp,ag];};

% He was mistakenly thought here
_V1 {Subcat=[ADVP{Mod_Elem=[np];}=advp,NP_{}=np];} ->
        `_V_pas {ComlexFrame="NP-ADVP-PRED";Subcat=[advp,*];};
% He was mistakenly thought here by them
_V1 {Subcat=[ADVP{Mod_Elem=[np];}=advp,PP_by{Arg=[ag];},NP_{}=np];} ->
        `_V_pas {ComlexFrame="NP-ADVP-PRED";Subcat=[advp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-PP-PRED %%%%%%%%%%%%%%%%%%%%%%%%

% I considered that problem of little concern
_V1 {Subcat=[np,PP{Arg=[NP{}];Mod_Elem=[np];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP-PRED";Subcat=[pp1,ag];};
_V1 {Subcat=[np,PP{Arg=[];Mod_Elem=[np];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PP-PRED";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-VEN-NP-OMIT %%%%%%%%%%%%%%%%%%%%%%%%
```

```
% We wanted the children found
_V1 {Subcat=[rnp3_p,VP_pas{Slash=[];Subcat=[rnp3_p];}=vp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-VEN-NP-OMIT";Subcat=[vp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ADJP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He painted the car black
_V1 {Subcat=[pat,ADJP{Mod_Elem=[pat];}=adjp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ADJP";Subcat=[pat,adjp,ag];};
% He scrubbed clean that old pot he had bought at the auction
_V1 {Subcat=[ADJP{Mod_Elem=[pat];}=adjp,pat,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ADJP";Subcat=[pat,adjp,ag];};

% The car was painted black
_V1 {Subcat=[ADJP{Mod_Elem=[pat];}=adjp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ADJP";Subcat=[pat,adjp,*];};
% The car was painted black by John
_V1 {Subcat=[ADJP{Mod_Elem=[pat];}=adjp,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ADJP";Subcat=[pat,adjp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-ADVP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He put it there
_V1 {Subcat=[pat,ADVP{Mod_Elem=[pat];}=advp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-ADVP";Subcat=[pat,advp,ag];};

% It has been put there
_V1 {Subcat=[ADVP{Mod_Elem=[pat];}=advp,NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ADVP";Subcat=[pat,advp,*];};
% It has been put there by Peter
_V1 {Subcat=[ADVP{Mod_Elem=[pat];}=advp,PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-ADVP";Subcat=[pat,advp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% ADJP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% His reputation sank low
_V1 {Subcat=[ADJP{Numerical=-;Mod_Elem=[ag];}=adjp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="ADJP";Subcat=[adjp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% ADVP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% She meant well
_V1 {Subcat=[ADVP{Mod_Elem=[];}=advp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="ADVP";Subcat=[advp,ag];};

% It was meant well
_V1 {Subcat=[ADVP{Mod_Elem=[];}=advp,NP_it{}];} ->
        `_V_pas {ComlexFrame="ADVP";Subcat=[advp,*];};
% It was meant well by Mary
_V1 {Subcat=[ADVP{Mod_Elem=[];}=advp,PP_by{Arg=[ag];},NP_it{}];} ->
        `_V_pas {ComlexFrame="ADVP";Subcat=[advp,ag];};
```

```
%%%%%%%%%%%%%%%%%%%%%%%% NP-PRED %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He became a secretary
_V1 {Subcat=[np,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PRED";Subcat=[np,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% ADJP-PRED %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He became clever
_V1 {Subcat=[ADJP{Mod_Elem=[ag];}=adjp,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="ADJP-PRED";Subcat=[adjp,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-PRED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He acted out of dispair
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[ag];}=pp1,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="PP-PRED";Subcat=[pp1,ag];};


%%%%%%%%%%%%%%%%%%%%%%%% NP-PRED-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%

% He seemed a fool
_V1 {Subcat=[NP_{NForm=(noun,pronoun,propername);}=np,NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-PRED-RS";Subcat=[np,ag];};
% What it did there is the question
_V1 {Subcat=[NP_{NForm=(noun,pronoun,propername);}=np,SBAR_argx{Slash=[];}=sbar];} ->
        `_V_npas {ComlexFrame="NP-PRED-RS";Subcat=[np,sbar];};
% To do it is a necessity
_V1 {Subcat=[NP_{NForm=(noun,pronoun,propername);}=np,VP_inf{Slash=[];}=vp];} ->
        `_V_npas {ComlexFrame="NP-PRED-RS";Subcat=[np,vp];};


%%%%%%%%%%%%%%%%%%%%%%%% ADJP-PRED-RS %%%%%%%%%%%%%%%%%%%%%%%%%%

% He appears crazy
% Defining combat aircraft is even tougher .
_V1 {Subcat=[ADJP{Mod_Elem=[*]=r;}=adjp|r];} ->
        `_V_npas {ComlexFrame="ADJP-PRED-RS";Subcat=[adjp];};

% It is uncertain whether he will come
_V1 {Subcat=[ADJP{Mod_Elem=[sbar];}=adjp,sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="ADJP-PRED-RS";Subcat=[adjp];};


%%%%%%%%%%%%%%%%%%%%%%%% PP-PRED-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%

% The situation seems out of control
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[np];}=pp1,NP_{}=np];} ->
        `_V_npas {ComlexFrame="PP-PRED-RS";Subcat=[pp1];};
% That he was there is out of question
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[sbar];}=pp1,sbar];} ->
        `_V_npas {ComlexFrame="PP-PRED-RS";Subcat=[pp1];};%
% To go there is out of question
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[vp];}=pp1,vp];} ->
        `_V_npas {ComlexFrame="PP-PRED-RS";Subcat=[pp1];};
```

```
%%%%%%%%%%%%%%%%%%%%%%% AS-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I worked as an apprentice cook
_V1 {Subcat=[PP_as{Arg=[np];},NP_{}=ag];} ->
        `_V_npas {ComlexFrame="AS-NP";Subcat=[np,ag];};


%%%%%%%%%%%%%%%%%%%%%%% NP-AS-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I sent him as a messenger
_V1 {Subcat=[pat,PP_as{Arg=[np];},NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-AS-NP";Subcat=[pat,np,ag];};

% He was sent as a messenger
_V1 {Subcat=[PP_as{Arg=[np];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-AS-NP";Subcat=[pat,np,*];};
% He was sent as a messenger by the king
_V1 {Subcat=[PP_as{Arg=[np];},PP_by{Arg=[ag];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-AS-NP";Subcat=[pat,np,ag];};
% He was sent by the king as a messenger
_V1 {Subcat=[PP_by{Arg=[ag];},PP_as{Arg=[np];},NP_{}=pat];} ->
        `_V_pas {ComlexFrame="NP-AS-NP";Subcat=[pat,np,ag];};


%%%%%%%%%%%%%%%%%%%%%%% NP-AS-NP-SC %%%%%%%%%%%%%%%%%%%%%%%%%

% She served the firm as a researcher
_V1 {Subcat=[ben,PP_as{Arg=[np];},NP_{}=ag];} ->
        `_V_npas {ComlexFrame="NP-AS-NP-SC";Subcat=[ben,np,ag];};


%%%%%%%%%%%%%%%%%%%%%%% EXTRAP-NP-S %%%%%%%%%%%%%%%%%%%%%%%%%

% It annoys them that she left
_V1 {Subcat=[pat,SBAR_that{}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-NP-S";Subcat=[pat,sbar];};

% That she left annoys them
_V1 {Subcat=[pat,SBAR_that{}=sbar];} ->
        `_V_npas {ComlexFrame="EXTRAP-NP-S";Subcat=[pat,sbar];};

% It pleases them to read
_V1 {Subcat=[pat,VP_inf{Slash=[];Subcat=[pat];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-NP-S";Subcat=[pat,vp];};

% To read pleases them
_V1 {Subcat=[pat,VP_inf{Slash=[];Subcat=[pat];}=vp];} ->
        `_V_npas {ComlexFrame="EXTRAP-NP-S";Subcat=[pat,vp];};


%%%%%%%%%%%%%%%%%%%%%%% EXTRAP-TO-NP-S %%%%%%%%%%%%%%%%%%%%%%%

% It matters to them that she left
_V1 {Subcat=[PP_to{Arg=[np];},SBAR_that{}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-NP-S";Subcat=[np,sbar];};
```

```
% that she left matters to them
_V1 {Subcat=[PP_to{Arg=[np];},SBAR_that{}=sbar];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-NP-S";Subcat=[np,sbar];};

% It occured to them to watch tv
_V1 {Subcat=[PP_to{Arg=[np];},VP_inf{Slash=[];Subcat=[np];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-NP-S";Subcat=[np,vp];};

% To come in first mattered to them
_V1 {Subcat=[PP_to{Arg=[np];},VP_inf{Slash=[];Subcat=[np];}=vp];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-NP-S";Subcat=[np,vp];};


%%%%%%%%%%%%%%%%%%%%%%% SEEM-S %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It seems that they left
_V1 {Subcat=[SBAR_that{}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="SEEM-S";Subcat=[sbar];};

% It seems they left
_V1 {Subcat=[s,NP_it{}];} ->
        `_V_npas {ComlexFrame="SEEM-S";Subcat=[s];};


%%%%%%%%%%%%%%%%%%%%%%% SEEM-TO-NP-S %%%%%%%%%%%%%%%%%%%%%%%

% It seems to her that they left
_V1 {Subcat=[PP_to{Arg=[ben];},SBAR_that{}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="SEEM-TO-NP-S";Subcat=[ben,sbar];};

% It seems to her they left
_V1 {Subcat=[PP_to{Arg=[ben];},s,NP_it{}];} ->
        `_V_npas {ComlexFrame="SEEM-TO-NP-S";Subcat=[ben,s];};


%%%%%%%%%%%%%%%%%%%%%%% EXTRAP-TO-INF %%%%%%%%%%%%%%%%%%%%%%%

% It remains to find the cure
_V1 {Subcat=[VP_inf{Slash=[];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-INF";Subcat=[vp];};

% It remains for us to find the cure
_V1 {Subcat=[SBAR_for{}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="EXTRAP-TO-INF";Subcat=[sbar];};


%%%%%%%%%%%%%%%%%%%%%%% S-SUBJ-S-OBJ %%%%%%%%%%%%%%%%%%%%%%%

% For him to report the theft indicates that he wasn't guilty
_V1 {Subcat=[SBAR_argx{}=sbar,SBAR_argx{}=sbar2];} ->
        `_V_npas {ComlexFrame="S-SUBJ-S-OBJ";Subcat=[sbar,sbar2];};


%%%%%%%%%%%%%%%%%%%%%%% TO-INF-NP-OMIT %%%%%%%%%%%%%%%%%%%%%%%

        %%% He is to blame NP*
```

```
_V1 {Subcat=[VP_inf{Slash=[rnp3_p];}=vp,NP_{}=rnp3_p];} ->
        `_V_npas {ComlexFrame="TO-INF-NP-OMIT";Subcat=[vp];};


%%%%%%%%%%%%%%%%%%%%%%%% THERE-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% There are three men
_V1 {Subcat=[NP_{}=np,NP_there{}];} ->
        `_V_npas {ComlexFrame="THERE-NP";Subcat=[np];};


%%%%%%%%%%%%%%%%%%%%%%%% THERE-NP-PP-PRED %%%%%%%%%%%%%%%%%%%%%%%%

        %%% There appeared in their lives an oppressive mischance
_V1 {Subcat=[NP_{}=np,PP{Arg=[NP{}];Mod_Elem=[np];}=pp1,NP_there{}];} ->
        `_V_npas {ComlexFrame="THERE-NP-PP-PRED";Subcat=[pp1];};

        %%% There is/appeared a unicorn in the garden
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[np];}=pp1,NP_{}=np,NP_there{}];} ->
        `_V_npas {ComlexFrame="THERE-NP-PP-PRED";Subcat=[pp1];};


%%%%%%%%%%%%%%%%%%%%%%%% THERE-NP-ING %%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% There is a unicorn grazing in the garden
_V1 {Subcat=[NP_{}=np,VP_prp{Subcat=[np];Slash=[];}=vp,NP_there{}];} ->
        `_V_npas {ComlexFrame="THERE-NP-ING";Subcat=[vp];};


%%%%%%%%%%%%%%%%%%%%%%%% IT-ADJP-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% It is good that he came .
_V1 {Subcat=[ADJP{Mod_Elem=[sbar];}=adjp,SBAR_arg{Slash=[];}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-ADJP-RS";Subcat=[adjp];};
        %%% It is good he came .
_V1 {Subcat=[ADJP{Mod_Elem=[s];}=adjp,S{Slash=[];}=s,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-ADJP-RS";Subcat=[adjp];};
        %%% It is good to come home .
_V1 {Subcat=[ADJP{Mod_Elem=[vp];}=adjp,VP_inf{Slash=[];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-ADJP-RS";Subcat=[adjp];};


%%%%%%%%%%%%%%%%%%%%%%%% IT-NP-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%% It is a problem that he left .
_V1 {Subcat=[np,SBAR_arg{Slash=[];}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-NP-RS";Subcat=[np,sbar];};
        %%% It is a problem he left .
_V1 {Subcat=[np,S{Slash=[];}=s,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-NP-RS";Subcat=[np,s];};
        %%% It is a problem to pass the test .
_V1 {Subcat=[np,VP_inf{Slash=[];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-NP-RS";Subcat=[np,vp];};


%%%%%%%%%%%%%%%%%%%%%%%% IT-PP-RS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %%% It is out of question that he left .
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[sbar];}=pp1,SBAR_arg{Slash=[];}=sbar,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-PP-RS";Subcat=[pp1];};
        %%% It is out of question he left .
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[s];}=pp1,S{Slash=[];}=s,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-PP-RS";Subcat=[pp1];};
        %%% It is out of question not to pass the test .
_V1 {Subcat=[PP{Arg=[NP{}];Mod_Elem=[vp];}=pp1,VP_inf{Slash=[];}=vp,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-PP-RS";Subcat=[pp1];};


%%%%%%%%%%%%%%%%%%%%%%% IT-INTRANS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It rains
_V1 {Subcat=[NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-INTRANS";Subcat=[];};

%%%%%%%%%%%%%%%%%%%%%%% IT-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It rains cats and dogs
_V1 {Subcat=[pat,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-NP";Subcat=[pat];};

%%%%%%%%%%%%%%%%%%%%%%% IT-ADVP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% It rains heavily
_V1 {Subcat=[ADVP{Mod_Elem=[];}=advp,NP_it{}];} ->
        `_V_npas {ComlexFrame="IT-ADVP";Subcat=[advp];};




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                 %
%  File:    adjective-rules.yap                                   %
%  Purpose: lexical transformation rules for the                  %
%           English grammar for the YAP parser                    %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart                %
%                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%% PREDICATIVE %%%%%%%%%%%%%%%%%%%%%%%%

% The sea is blue .
ADJ_ {Subcat=[];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="";};

% 10 points higher
ADJ_ {Subcat=[NP{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {Degree=comp;ComlexFrame="";};


%%%%%%%%%%%%%%%%%% ATTRIBUTIVE %%%%%%%%%%%%%%%%%%%%%%%%

% The old man laughs
ADJ_ {Subcat=[];Mod_Elem=[NBAR{Elliptical=-;}];} ->
```

```
        `_ADJ_attr {ComlexFrame="";Nominal=-;};

% The rich and the beautiful
ADJ_ {Subcat=[];Mod_Elem=[NBAR{Elliptical=+;}];} ->
        `_ADJ_attr {ComlexFrame="";Nominal=+;};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-S %%%%%%%%%%%%%%%%%%%%%%

% It is probable he left .
ADJ_ {Subcat=[];Mod_Elem=[S{}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-S";};

% It is probable that he left .
% That he left is probable .
ADJ_ {Subcat=[];Mod_Elem=[SBAR_that{}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-S";};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-THAT-S %%%%%%%%%%%%%%%%%

% It is curious that he left .
% That he left is curious .
ADJ_ {Subcat=[];Mod_Elem=[SBAR_that{}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-THAT-S";};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-S-SUBJUNCT %%%%%%%%%%%%%%

% It is imperative that they leave
% That they leave is imperative
ADJ_ {Subcat=[];Mod_Elem=[SBAR_subj{}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-S-SUBJUNCT";};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-WH-S %%%%%%%%%%%%%%%%%%%%

% It is uncertain whether he will come
% Whether he will come is uncertain
ADJ_ {Subcat=[];Mod_Elem=[SBAR_argwb{}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-WH-S";};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-FOR-TO-INF %%%%%%%%%%%%%%

% It is practical for Evans to go to school
% For Evans to go to school is practical
ADJ_ {Subcat=[];Mod_Elem=[SBAR_for{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF";};

% It is practical to go to school
% To go to school is practical
ADJ_ {Subcat=[];Mod_Elem=[VP_inf{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF";};


%%%%%%%%%%%%%%%%% EXTRAP-ADJ-FOR-TO-INF-NP-OMIT %%%%%%%
```

```
% The race was easy for her to win .
ADJ_ {Subcat=[];Mod_Elem=[SBAR_for{Slash=[NP_{}];}];} ->
        `_ADJ {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-NP-OMIT";};

% The race was easy to win .
ADJ_ {Subcat=[];Mod_Elem=[VP_inf{Slash=[NP_{}];}];} ->
        `_ADJ {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-NP-OMIT";};

% For her to win the race was easy .
ADJ_ {Subcat=[];Mod_Elem=[SBAR_for{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-NP-OMIT";};

% To win the race was easy .
ADJ_ {Subcat=[];Mod_Elem=[VP_inf{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-NP-OMIT";};

% Defining combat aircraft is even tougher .
ADJ_ {Subcat=[];Mod_Elem=[VP_prp{Subcat=[NP_{}];}];} ->
        `_ADJ {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-NP-OMIT";};


%%%%%%%%%%%%%%%% EXTRAP-ADJ-FOR-TO-INF-RS %%%%%%%%%%

% For Joan to invite me was kind .
ADJ_ {Subcat=[];Mod_Elem=[SBAR_for{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-RS";};

% To invite me was kind .
ADJ_ {Subcat=[];Mod_Elem=[VP_inf{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-RS";};

% Joan was kind to invite me .
ADJ_ {Subcat=[VP_inf{Subcat=[np];Slash=[];}];Mod_Elem=[NP_{}=np];} ->
        `_ADJ_pred {ComlexFrame="EXTRAP-ADJ-FOR-TO-INF-RS";};


%%%%%%%%%%%%%%%% S-ADJ %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% They were certain the team would lose .
ADJ_ {Subcat=[S{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="S-ADJ";};

% They were certain that the team would lose .
ADJ_ {Subcat=[SBAR_that{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="S-ADJ";};


%%%%%%%%%%%%%%%% THAT-S-ADJ %%%%%%%%%%%%%%%%%%%%%%%

% They were aware that he was sick .
ADJ_ {Subcat=[SBAR_that{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="THAT-S-ADJ";};


%%%%%%%%%%%%%%%% FOR-TO-ADJ %%%%%%%%%%%%%%%%%%%%%%%
```

```
% He was anxious for her to succeed .
ADJ_ {Subcat=[SBAR_for{Slash=[];}];} ->
        `_ADJ_pred {ComlexFrame="FOR-TO-ADJ";};

% He was anxious to succeed .
ADJ_ {Subcat=[VP_inf{Subcat=[np];Slash=[];}];Mod_Elem=[np];} ->
        `_ADJ_pred {ComlexFrame="FOR-TO-ADJ";};


%%%%%%%%%%%%%%%%% S-SUBJUNCT-ADJ %%%%%%%%%%%%%%%%%%%%

% I am insistent that he study .
ADJ_ {Subcat=[SBAR_subj{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="S-SUBJUNCT-ADJ";};


%%%%%%%%%%%%%%%%% S-WH-ADJ %%%%%%%%%%%%%%%%%%%%%%%%%%

% They were uncertain if it would work .
ADJ_ {Subcat=[SBAR_argb{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="S-WH-ADJ";};


%%%%%%%%%%%%%%%%% ADJ-TO-INF %%%%%%%%%%%%%%%%%%%%%%%%

% She was able to climb the mountain .
ADJ_ {Subcat=[VP_inf{Subcat=[np];Slash=[];}];Mod_Elem=[NP_{}=np];} ->
        `_ADJ_pred {ComlexFrame="ADJ-TO-INF";};


%%%%%%%%%%%%%%%%% ADJ-PP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He was happy for her .
ADJ_ {Subcat=[PP{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="ADJ-PP";};

%%%%%%%%%%%%%%%%% NP-ADJ %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% He was ten years old
ADJ_ {Subcat=[NP_acc{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="NP-ADJ";};

%%%%%%%%%%%%%%%%% NP-ADJ-PP %%%%%%%%%%%%%%%%%%%%%%%%%

% He was miles ahead of the others
ADJ_ {Subcat=[PP{},NP_acc{}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="NP-ADJ-PP";};

%%%%%%%%%%%%%%%%% ADJ-NP %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% payable October 1st
ADJ_ {Subcat=[NP_{Adjunctive=+;}];Mod_Elem=[NP_{}];} ->
        `_ADJ_pred {ComlexFrame="NP-ADJ";};
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                   %
%  File:    lexicon.yap                             %
%  Purpose: Lexical rules for my English YAP grammar %
%  Author:  Helmut Schmid, IMS, Univ. of Stuttgart   %
%                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Default Entries             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


<default>       : N_   {HeadLex=h1;Phon=h1;Subcat=[];};
<default>       : ADJ_ {HeadLex=h1;Phon=h1;Subcat=[];Degree=(pos,comp,sup);Subcat=[];};
<default>       : ADV_pos {HeadLex=h1;Phon=h1;};
<default>       : _V   {HeadLex=h1;Phon=h1;};

<propername>    : N_   {NForm=propername;HeadLex=h1;Phon=h1;Subcat=[];};
<ordinal>       : ADJ_ord {};
<cardinal>      : ADJ_card {};

"for"           : FOR  {HeadLex="for";};
"the"           : THE  {HeadLex="the";};

"than"          : COMP {HeadLex="than";Degree=comp;};
"as"            : COMP {HeadLex="as";Degree=as;};
"like"          : COMP {HeadLex="like";Degree=pos;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              PDT                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

"all"           : PDT {HeadLex="all";};
"both"          : PDT {HeadLex="both";Number=pl;};
"half"          : PDT {HeadLex="half";Number=sg;};
"quite"         : PDT {HeadLex="quite";};
"such"          : PDT {HeadLex="such";Number=sg;};
"yet"           : PDT {HeadLex="yet";Number=sg;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              DT                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


"a"             : DT_sg {HeadLex="a";};
"an"            : DT_sg {HeadLex="a";};
"another"       : DT_sg {HeadLex="another";};
"each"          : DT_sg {HeadLex="each";};
"each"          : DT_pl {HeadLex="each";};  % for each 100 shares owned
"either"        : DT_sg {HeadLex="either";};
"every"         : DT_sg {HeadLex="every";};
```

```
"neither"       : DT_sg {HeadLex="neither";};
"that"          : DT_sg {HeadLex="that";};
"this"          : DT_sg {HeadLex="this";};

"these"         : DT_pl {HeadLex="this";};
"those"         : DT_pl {HeadLex="that";};
"all"           : DT_pl {HeadLex="all";};
"both"          : DT_pl {HeadLex="both";};

"any"           : DT {HeadLex="any"; WhForm=-;};
"no"            : DT {HeadLex="no";  WhForm=-;};
"some"          : DT {HeadLex="some";WhForm=-;};
"the"           : DT {HeadLex="the"; WhForm=-;};

"which"         : DT {HeadLex="which";WhForm=quest;};
"what"          : DT {HeadLex="what"; WhForm=quest;};


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             PRO                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Null-head determiners


"more"          : NP {HeadLex="more";NForm=pronoun;WhForm=-;Elliptical=-;Adjunctive=-;
                      Degree=comp;Coord=-;Commas=-;NPLevel=0;Slash=[];
                      Person=3rd;Case=(nom,acc);};
"most"          : NP {HeadLex="most";NForm=pronoun;WhForm=-;Elliptical=-;Adjunctive=-;
                      Degree=sup;Coord=-;Commas=-;NPLevel=0;Slash=[];
                      Person=3rd;Case=(nom,acc);};
"less"          : NP {HeadLex="less";NForm=pronoun;WhForm=-;Elliptical=-;Adjunctive=-;
                      Degree=comp;Coord=-;Commas=-;NPLevel=0;Slash=[];
                      Person=3rd;Case=(nom,acc);};
"fewer"         : NP {HeadLex="fewer";NForm=pronoun;WhForm=-;Elliptical=-;Adjunctive=-;
                      Degree=comp;Coord=-;Commas=-;NPLevel=0;Slash=[];
                      Person=3rd;Case=(nom,acc);};

"few"           : PRO_3p {HeadLex="few";};
"many"          : PRO_3p {HeadLex="many";};
"little"        : PRO_3s {HeadLex="little";};
"much"          : PRO_3s {HeadLex="much";};

"all"           : PRO_3p {HeadLex="all";};
"both"          : PRO_3p {HeadLex="both";};
"certain"       : PRO_3p {HeadLex="certain";};
"several"       : PRO_3p {HeadLex="several";};
"some"          : PRO_3p {HeadLex="some";};

"another"       : PRO_3s {HeadLex="another";};
"any"           : PRO_3s {HeadLex="any";};
"each"          : PRO_3s {HeadLex="each";};
"either"        : PRO_3s {HeadLex="either";};
"neither"       : PRO_3s {HeadLex="neither";};
"such"          : PRO_3s {HeadLex="such";};

"anybody"       : PROO_3s {HeadLex="anybody";};
```

```
"anyone"        : PRO0_3s {HeadLex="anyone";};
"anything"      : PRO0_3s {HeadLex="anything";};
"everybody"     : PRO0_3s {HeadLex="everybody";};
"everyone"      : PRO0_3s {HeadLex="everyone";};
"everything"    : PRO0_3s {HeadLex="everything";};
"nobody"        : PRO0_3s {HeadLex="nobody";};
"none"          : PRO0_3s {HeadLex="none";};
"noone"         : PRO0_3s {HeadLex="noone";};
"nothing"       : PRO0_3s {HeadLex="nothing";};
"somebody"      : PRO0_3s {HeadLex="somebody";};
"someone"       : PRO0_3s {HeadLex="someone";};
"something"     : PRO0_3s {HeadLex="something";};


%%% Demonstratives

"that"          : PRO {HeadLex="that";NPLevel=2;WhForm=-;Number=sg;Person=3rd;
                       Case=(nom,acc);};
"these"         : PRO_3p {HeadLex="this";};
"this"          : PRO_3s {HeadLex="this";};
"those"         : PRO_3p {HeadLex="that";};


%% Personal pronouns

"he"            : PPRO_3s {HeadLex="he";Case=nom;};
"her"           : PPRO_3s {HeadLex="she";Case=acc;};
"him"           : PPRO_3s {HeadLex="he";Case=acc;};
"I"             : PPRO_sg {HeadLex="I";Case=nom;Person=1st;};
"it"            : PPRO_3s {HeadLex="it";};
"me"            : PPRO_sg {HeadLex="I";Case=acc;Person=1st;};
"one"           : PPRO_3s {HeadLex="one";};   %%% One can do ...
"she"           : PPRO_3s {HeadLex="she";Case=nom;};
"them"          : PPRO_3p {HeadLex="they";Case=acc;};
"'em"           : PPRO_3p {HeadLex="they";Case=acc;};
"they"          : PPRO_3p {HeadLex="they";Case=nom;};
"us"            : PPRO_pl {HeadLex="we";Case=acc;Person=1st;};
"'s"            : PPRO_pl {HeadLex="we";Case=acc;Person=1st;};
"we"            : PPRO_pl {HeadLex="we";Case=nom;Person=1st;};
"you"           : PPRO_2  {HeadLex="you";};
"ya"            : PPRO_2  {HeadLex="you";};

%%% Possessive pronouns

"her"           : PPRO_sg {HeadLex="her";  Case=gen;Person=3rd;};
"his"           : PPRO_sg {HeadLex="his";  Case=gen;Person=3rd;};
"its"           : PPRO_sg {HeadLex="its";  Case=gen;Person=3rd;};
"my"            : PPRO_sg {HeadLex="my";   Case=gen;Person=3rd;};
"our"           : PPRO_pl {HeadLex="our";  Case=gen;Person=3rd;};
"their"         : PPRO_pl {HeadLex="their";Case=gen;Person=3rd;};
"ones"          : PPRO_sg {HeadLex="ones"; Case=gen;Person=3rd;};
"your"          : PPRO_   {HeadLex="your"; Case=gen;Person=2nd;};

%%% Reflexive Pronouns

"herself"       : PPRO_3s {HeadLex="<refpro>";};
"himself"       : PPRO_3s {HeadLex="<refpro>";};
```

```
"itself"        : PPRO_3s {HeadLex="<refpro>";};
"oneself"       : PPRO_3s {HeadLex="<refpro>";};
"myself"        : PPRO_1s {HeadLex="<refpro>";};
"ourselves"     : PPRO_1p {HeadLex="<refpro>";};
"themselves"    : PPRO_3p {HeadLex="<refpro>";};
"thyself"       : PPRO_2s {HeadLex="<refpro>";};
"yourself"      : PPRO_2s {HeadLex="<refpro>";};
"yourselves"    : PPRO_2p {HeadLex="<refpro>";};

%%% expletive pronouns

"it"            : NP_it {};
"there"         : NP_there {};


%%%%%%%%%% WHPRO %%%%%%%%%%%%%%%%%%%%

"what"          : WHNP {HeadLex="what";Case=(nom,acc);Number=sg;};
"which"         : WHNP {HeadLex="which";Case=(nom,acc);Number=sg;};
"who"           : WHNP {HeadLex="who";Case=(nom,acc);Number=sg;};
"whom"          : WHNP {HeadLex="who";Case=acc;Number=sg;};
"whose"         : WHNP {HeadLex="who";Case=gen;Number=sg;};

"when"          : WHPP {HeadLex="when";};
"where"         : WHPP {HeadLex="where";};


%%%%%%%%%% RELPRO %%%%%%%%%%%%%%%%%%%%

"who"           : RELNP {Case=(nom,acc);};
"whom"          : RELNP {Case=acc;};
"whose"         : RELNP {Case=gen;};
"which"         : RELNP {Case=(nom,acc);};

"whereby"       : RELPP {HeadLex="by";};
"wherein"       : RELPP {HeadLex="in";};
"where"         : RELPP {HeadLex="where";};
"when"          : RELPP {HeadLex="when";};


%%%%%%%%%% LOCPRO %%%%%%%%%%%%%%%%%%%%

"around"        : PP0 {HeadLex="around";Mod=(noun,verb);};
"now"           : PP0 {HeadLex="now";Mod=(noun,verb);};
"then"          : PP0 {HeadLex="then";Mod=(noun,verb);};
"today"         : PP0 {HeadLex="today";Mod=(noun,verb);};
"tomorrow"      : PP0 {HeadLex="tomorrow";Mod=(noun,verb);};
"yesterday"     : PP0 {HeadLex="yesterday";Mod=(noun,verb);};


"here"          : PP0 {HeadLex="here";};
"there"         : PP0 {HeadLex="there";};
"ahead"         : PP0 {HeadLex="ahead";};
"afterward"     : PP0 {HeadLex="afterward";};
"afterwards"    : PP0 {HeadLex="afterwards";};
"backward"      : PP0 {HeadLex="backward";};
"before"        : PP0 {HeadLex="before";};
```

```
"downward"      : PP0 {HeadLex="downward";};
"downtown"      : PP0 {HeadLex="downtown";};
"eastward"      : PP0 {HeadLex="eastward";};
"elsewhere"     : PP0 {HeadLex="elsewhere";};
"forward"       : PP0 {HeadLex="forward";};
"heavenward"    : PP0 {HeadLex="heavenward";};
"henceforward"  : PP0 {HeadLex="henceforward";};
"homeward"      : PP0 {HeadLex="homeward";};
"inward"        : PP0 {HeadLex="inward";};
"landward"      : PP0 {HeadLex="landward";};
"leeward"       : PP0 {HeadLex="leeward";};
"onward"        : PP0 {HeadLex="onward";};
"out"           : PP0 {HeadLex="out";Mod=-;};
"outward"       : PP0 {HeadLex="outward";};
"oversea"       : PP0 {HeadLex="oversea";};
"overseas"      : PP0 {HeadLex="overseas";};
"skyward"       : PP0 {HeadLex="skyward";};
"somewhere"     : PP0 {HeadLex="somewhere";};
"southward"     : PP0 {HeadLex="southward";};
"thenceforward" : PP0 {HeadLex="thenceforward";};
"upward"        : PP0 {HeadLex="upward";};
"westward"      : PP0 {HeadLex="westward";};


%%%%%%%%%% P %%%%%%%%%%%%%%%%%%%%%%

"after"         : P {HeadLex="after";Arg=[*];};
"before"        : P {HeadLex="before";Arg=[*];};
"notwithstanding" : P {HeadLex="notwithstanding";Arg=[*];};

"ago"           : P {HeadLex="ago";Order=post;Arg=[*];Mod=(verb,noun);};
"away"          : P {HeadLex="away";Order=post;Arg=[*];Mod=(verb,noun);};
"earlier"       : P {HeadLex="earlier";Order=*;Arg=[*];Mod=(verb,noun);};
"later"         : P {HeadLex="later";Order=(pre,post);Arg=[*];Mod=(verb,noun);};

"about"         : P_ {HeadLex="about";Mod=(-,noun);};
"against"       : P_ {HeadLex="against";};
"as"            : P_ {HeadLex="as";Mod=(verb,noun,\-);};
"at"            : P_ {HeadLex="at";};
"besides"       : P_ {HeadLex="besides";};
"between"       : P_ {HeadLex="between";};
"beyond"        : P_ {HeadLex="beyond";};
"by"            : P_ {HeadLex="by";Mod=(verb,-);};
"by"            : P_ {HeadLex="by";Mod_Elem=[NP{NForm=noun;}];Mod=noun;};
"despite"       : P_ {HeadLex="despite";};
"during"        : P_ {HeadLex="during";};
"except"        : P_ {HeadLex="except";};
"for"           : P_ {HeadLex="for";};
"from"          : P_ {HeadLex="from";};
"in"            : P_ {HeadLex="in";};
"into"          : P_ {HeadLex="into";};
"like"          : P_ {HeadLex="like";Mod=(verb,-);};
"of"            : P_ {HeadLex="of";Mod=(noun,\-);};
"off"           : P_ {HeadLex="off";};
"on"            : P_ {HeadLex="on";};
"over"          : P_ {HeadLex="over";};
"since"         : P_ {HeadLex="since";};
```

```
"to"            : P_ {HeadLex="to";Mod=(noun,-);};
"through"       : P_ {HeadLex="through";};
"toward"        : P_ {HeadLex="toward";};
"towards"       : P_ {HeadLex="towards";};
"under"         : P_ {HeadLex="under";};
"unlike"        : P_ {HeadLex="unlike";};
"until"         : P_ {HeadLex="until";};
"'til"          : P_ {HeadLex="until";};
"upon"          : P_ {HeadLex="upon";};
"with"          : P_ {HeadLex="with";};
"within"        : P_ {HeadLex="within";};
"without"       : P_ {HeadLex="without";};
"worth"         : P_ {HeadLex="worth";Mod=-;};

"aboard"        : P_ {HeadLex="aboard";Mod=(noun,verb);Arg=[NP{}];};
"above"         : P_ {HeadLex="above";Mod=(noun,verb,-);Arg=[NP{}];};
"across"        : P_ {HeadLex="across";Mod=(noun,verb,-);Arg=[NP{}];};
"along"         : P_ {HeadLex="along";Mod=(noun,verb,-);Arg=[NP{}];};
"alongside"     : P_ {HeadLex="alongside";Mod=(noun,verb,-);Arg=[NP{}];};
"amid"          : P_ {HeadLex="amid";Mod=(noun,verb,-);Arg=[NP{}];};
"amidst"        : P_ {HeadLex="amidst";Mod=(noun,verb,-);Arg=[NP{}];};
"among"         : P_ {HeadLex="among";Mod=(noun,verb,-);Arg=[NP{}];};
"amongst"       : P_ {HeadLex="amongst";Mod=(noun,verb,-);Arg=[NP{}];};
"around"        : P_ {HeadLex="around";Mod=(noun,verb,-);Arg=[NP{}];};
"astride"       : P_ {HeadLex="astride";Mod=(noun,verb);Arg=[NP{}];};
"atop"          : P_ {HeadLex="atop";Mod=(noun,verb,-);Arg=[NP{}];};
"behind"        : P_ {HeadLex="behind";Mod=(noun,verb,-);Arg=[NP{}];};
"below"         : P_ {HeadLex="below";Mod=(noun,verb,-);Arg=[NP{}];};
"beneath"       : P_ {HeadLex="beneath";Mod=(noun,verb,-);Arg=[NP{}];};
"beside"        : P_ {HeadLex="beside";Mod=(noun,verb,-);Arg=[NP{}];};
"down"          : P_ {HeadLex="down";Mod=(noun,verb,-);Arg=[NP{}];};
"inside"        : P_ {HeadLex="inside";Mod=(noun,verb,-);Arg=[NP{}];};
"less"          : P_ {HeadLex="less";Mod=noun;Arg=[NP{}];};
"minus"         : P_ {HeadLex="minus";Mod=noun;Arg=[NP{}];};
"near"          : P_ {HeadLex="near";Mod=(noun,verb,-);Arg=[NP{}];};
"nearer"        : P_ {HeadLex="nearer";Mod=(noun,verb,-);Arg=[NP{}];};
"nearest"       : P_ {HeadLex="nearest";Mod=(noun,verb,-);Arg=[NP{}];};
"onto"          : P_ {HeadLex="onto";Mod=(noun,verb,-);Arg=[NP{}];};
"opposite"      : P_ {HeadLex="opposite";Mod=(noun,verb,-);Arg=[NP{}];};
"outside"       : P_ {HeadLex="outside";Mod=(noun,verb,-);Arg=[NP{}];};
"past"          : P_ {HeadLex="past";Mod=(noun,verb,-);Arg=[NP{}];};
"pending"       : P_ {HeadLex="pending";Mod=verb;Arg=[NP{}];};
"per"           : P_ {HeadLex="per";Mod=(noun,verb);Arg=[NP{}];};
"plus"          : P_ {HeadLex="plus";Mod=noun;Arg=[NP{}];};
"round"         : P_ {HeadLex="round";Mod=(noun,verb,-);Arg=[NP{}];};
"throughout"    : P_ {HeadLex="throughout";Mod=(noun,verb);Arg=[NP{}];};
"till"          : P_ {HeadLex="till";Mod=(noun,verb,-);Arg=[NP{}];};
"times"         : P_ {HeadLex="times";Mod=noun;Arg=[NP{}];};
"underneath"    : P_ {HeadLex="underneath";Mod=(noun,verb,-);Arg=[NP{}];};
"unto"          : P_ {HeadLex="unto";Mod=(noun,verb,-);Arg=[NP{}];};
"up"            : P_ {HeadLex="up";Mod=(noun,verb,-);Arg=[NP{}];};
"v."            : P_ {HeadLex="versus";Mod=noun;Arg=[NP{}];};
"versus"        : P_ {HeadLex="versus";Mod=noun;Arg=[NP{}];};
"via"           : P_ {HeadLex="via";Mod=noun;Arg=[NP{}];};
"vs."           : P_ {HeadLex="versus";Mod=noun;Arg=[NP{}];};
"within"        : P_ {HeadLex="within";Mod=(noun,verb,-);Arg=[NP{}];};
```

```
%%%%%%%%%% CS %%%%%%%%%%%%%%%%%%%%%%

"that"          : C {HeadLex="that";CForm=(rel,arg,subj);};

"if"            : C {HeadLex="if";CForm=(argb,adj);};
"whether"       : C {HeadLex="whether";CForm=argb;};

"after"         : C_ {HeadLex="after";};
"although"      : C_ {HeadLex="although";};
"as"            : C_ {HeadLex="as";};
"because"       : C_ {HeadLex="because";};
"before"        : C_ {HeadLex="before";};
"except"        : C_ {HeadLex="except";};
"for"           : C_ {HeadLex="for";};
"however"       : C_ {HeadLex="however";};
"lest"          : C_ {HeadLex="lest";};
"once"          : C_ {HeadLex="once";};
"since"         : C_ {HeadLex="since";};
"so"            : C_ {HeadLex="so";};
"though"        : C_ {HeadLex="though";};
"till"          : C_ {HeadLex="till";};
"unless"        : C_ {HeadLex="unless";};
"until"         : C_ {HeadLex="until";};
"when"          : C_ {HeadLex="when";};
"whenever"      : C_ {HeadLex="whenever";};
"whereas"       : C_ {HeadLex="whereas";};
"whereupon"     : C_ {HeadLex="whereupon";};
"wherever"      : C_ {HeadLex="wherever";};
"while"         : C_ {HeadLex="while";};
"whilst"        : C_ {HeadLex="whilst";};


%%%%%%%%%% CC %%%%%%%%%%%%%%%%%%%%%%

"and"           : C {HeadLex="and";CForm=coord1;};
"or"            : C {HeadLex="or";CForm=coord1;};
"&"             : C {HeadLex="&";CForm=coord1;};
"but"           : C {HeadLex="but";CForm=coord2;};
"yet"           : C {HeadLex="yet";CForm=coord2;};
";"             : C {HeadLex=";";CForm=coord3;};
","             : C {HeadLex=",";CForm=coord3;};

"either"        : C {HeadLex="either_or";CForm=coord4;};
"both"          : C {HeadLex="both_and";CForm=coord4;};
"neither"       : C {HeadLex="neither_nor";CForm=coord4;};
"between"       : C {HeadLex="between_and";CForm=coord4;};

"and"           : C {HeadLex="between_and";CForm=coord5;};
"and"           : C {HeadLex="both_and";CForm=coord5;};
"nor"           : C {HeadLex="neither_nor";CForm=coord5;};
"or"            : C {HeadLex="either_or";CForm=coord5;};


%%%%%%%%%% SM %%%%%%%%%%%%%%%%%%%%%%
```

```
"."             : SM {HeadLex=".";SForm=decl;};
"..."           : SM {HeadLex="...";SForm=decl;};
":"             : SM {HeadLex=":";SForm=decl;};
"!"             : SM {HeadLex="!";SForm=imp;};
"?"             : SM {HeadLex="?";SForm=quest;};


%%%%%%%%%% CM %%%%%%%%%%%%%%%%%%%%%%

","             : CM {HeadLex=",";};
"-"             : CM {HeadLex="-";};
"--"            : CM {HeadLex="-";};


%%%%%%%%%% Q  %%%%%%%%%%%%%%%%%%%%%%

"\""            : Q {HeadLex="\"";};
"``"            : Q {HeadLex="``";Pos=left;};
"''"            : Q {HeadLex="``";Pos=right;};
"("             : Q {HeadLex="(";Pos=left;};
")"             : Q {HeadLex="(";Pos=right;};
"["             : Q {HeadLex="[";Pos=left;};
"]"             : Q {HeadLex="[";Pos=right;};
"'"             : Q {HeadLex="'";};


%%%%%%%%%% GM %%%%%%%%%%%%%%%%%%%%%%

"'s"            : GM {HeadLex="'s";};
"'"             : GM {HeadLex="'s";};

%%%%%%%%%% CURR %%%%%%%%%%%%%%%%%%%%%

"$"             : CURR {HeadLex="$";Subcat=[];};
"US$"           : CURR {HeadLex="US$";Subcat=[];};
"CAN$"          : CURR {HeadLex="CAN$";Subcat=[];};
"DM"            : CURR {HeadLex="DM";Subcat=[];};
"#"             : CURR {HeadLex="#";Subcat=[];};

"$"             : CURR {HeadLex="$";Subcat=[NP_curr{}];};
"US$"           : CURR {HeadLex="US$";Subcat=[NP_curr{}];};
"CAN$"          : CURR {HeadLex="CAN$";Subcat=[NP_curr{}];};
"DM"            : CURR {HeadLex="DM";Subcat=[NP_curr{}];};
"#"             : CURR {HeadLex="#";Subcat=[NP_curr{}];};

%%%%%%%%%% ADV %%%%%%%%%%%%%%%%%%%%%%

"how"           : WHADVP {HeadLex="how";};
"why"           : WHADVP {HeadLex="why";Mod=verb;};

"not"           : ADV {HeadLex="not";Not=+;Degree=pos;Mod=(verb,adv,adj,noun,sbar);};
"n't"           : ADV {HeadLex="not";Not=+;Degree=pos;Mod=clitic;};
"no"            : ADV {HeadLex="no";Not=+;Degree=pos;Mod=adv;};
"no"            : ADV {HeadLex="no";Not=+;Degree=pos;Mod=adj;Mod_Elem=[ADJP{Pred=+;}];};

"about"         : ADV_pos {HeadLex="about";Mod=adj;Mod_Elem=[ADJP{Numerical=+;}];};
"over"          : ADV_pos {HeadLex="over";Mod=adj;Mod_Elem=[ADJP{Numerical=+;}];};
```

```
"some"          : ADV_pos {HeadLex="some";Mod=adj;Mod_Elem=[ADJP{Numerical=+;}];};

"too"           : ADV_verb {HeadLex="too";};


%%%%%%%%%% DEG %%%%%%%%%%%%%%%%%%%%%

"more"          : DEG {HeadLex="more";Degree=comp;};
"less"          : DEG {HeadLex="less";Degree=comp;};
"most"          : DEG {HeadLex="most";Degree=sup;};
"as"            : DEG {HeadLex="as";Degree=as;};
"too"           : DEG {HeadLex="too";Degree=too;};
"very"          : DEG {HeadLex="very";Degree=pos;};
"pretty"        : DEG {HeadLex="pretty";Degree=pos;};
"rather"        : DEG {HeadLex="rather";Degree=pos;};
"real"          : DEG {HeadLex="real";Degree=pos;};


%%%%%%%%%% ADJ %%%%%%%%%%%%%%%%%%%%%


"many"      : _ADJ_pos {HeadLex="many";ComlexFrame="";Nominal=-;};
"much"      : _ADJ_pos {HeadLex="much";ComlexFrame="";Nominal=-;};
"more"      : _ADJ_cmp {HeadLex="more";ComlexFrame="";Nominal=-;};
"more"      : _ADJ_cmp {HeadLex="more";ComlexFrame="NP-ADJ";};
"less"      : _ADJ_cmp {HeadLex="less";ComlexFrame="";Nominal=-;};
"less"      : _ADJ_cmp {HeadLex="less";ComlexFrame="NP-ADJ";};
"most"      : _ADJ_sup {HeadLex="most";ComlexFrame="";};
"old"       : _ADJ_pos {HeadLex="old";ComlexFrame="NP-ADJ";};

"on"        : _ADJ_pos {HeadLex="on";Pred=+;ComlexFrame="";};
"over"      : _ADJ_pos {HeadLex="over";Pred=+;ComlexFrame="";};
"only"      : _ADJ_pos {HeadLex="only";Pred=-;ComlexFrame="";};
"other"     : _ADJ_pos {HeadLex="other";Pred=-;ComlexFrame="";};
"several"   : _ADJ_pos {HeadLex="several";ComlexFrame="";Nominal=-;};
"such"      : _ADJ_pos {HeadLex="such";ComlexFrame="";Nominal=-;};

"ahead"     : _ADJ_pos {HeadLex="ahead";ComlexFrame="NP-ADJ";};
"ahead"     : _ADJ_pos {HeadLex="ahead";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="of";}];};
"ahead"     : _ADJ_pos {HeadLex="ahead";ComlexFrame="NP-ADJ-PP";Subcat=[PP{HeadLex="of";},NP{}];};

"away"      : _ADJ_pos {HeadLex="away";ComlexFrame="";};
"away"      : _ADJ_pos {HeadLex="away";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="of";}];};
"away"      : _ADJ_pos {HeadLex="away";ComlexFrame="NP-ADJ-PP";Subcat=[PP{HeadLex="from";},NP{}];};

"effective" : ADJ_pos {HeadLex="effective";Subcat=[NP_{HeadLex="<cardinal>";}];};
"inevitable": _ADJ_pos {HeadLex="inevitable";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="for";}];};
"full"      : _ADJ_pos {HeadLex="full";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="of";}];};
"payable"   : _ADJ_pos {HeadLex="payable";ComlexFrame="ADJ-NP";};
"payable"   : _ADJ_pos {HeadLex="payable";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="in";}];};
"related"   : _ADJ_pos {HeadLex="related";ComlexFrame="NP-ADJ";};


"up"        : _ADJ_pos_pred {HeadLex="up";ComlexFrame="";};
"up"        : _ADJ_pos_pred {HeadLex="up";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="from";}];};
"up"        : _ADJ_pos_pred {HeadLex="up";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="to";}];};
"down"      : _ADJ_pos_pred {HeadLex="down";ComlexFrame="";};
```

```
"down"      : _ADJ_pos_pred {HeadLex="down";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="from";}];};
"down"      : _ADJ_pos_pred {HeadLex="down";ComlexFrame="ADJ-PP";Subcat=[PP{HeadLex="to";}];};

"junior"    : ADJ_post {HeadLex="junior";Mod_Elem=[NP{WhForm=-;NForm=(propername,noun);}];};
"senior"    : ADJ_post {HeadLex="senior";Mod_Elem=[NP{WhForm=-;NForm=(propername,noun);}];};
"designate" : ADJ_post {HeadLex="designate";Mod_Elem=[NP{WhForm=-;NForm=noun;}];};
"outstanding": ADJ_post {HeadLex="outstanding";Mod_Elem=[NP{WhForm=-;NForm=noun;}];};
"emeritus"  : ADJ_post {HeadLex="emeritus";Mod_Elem=[NP{WhForm=-;NForm=noun;}];};
"else"      : ADJ_post {HeadLex="else";Mod_Elem=[NP{NForm=pronoun;}];};
"a.m."      : ADJ_post {HeadLex="a.m.";Mod_Elem=[NP{WhForm=-;NForm=noun;}];};
"p.m."      : ADJ_post {HeadLex="p.m.";Mod_Elem=[NP{WhForm=-;NForm=noun;}];};


%%%%%%%%%% M %%%%%%%%%%%%%%%%%%%%%

"dozen"     : M {HeadLex="dozen";};
"hundred"   : M {HeadLex="hundred";};
"thousand"  : M {HeadLex="thousand";};
"million"   : M {HeadLex="million";};
"billion"   : M {HeadLex="billion";};
"trillion"  : M {HeadLex="trillion";};
"percent"   : M {HeadLex="percent";};
"%"         : M {HeadLex="percent";};



%%%%%%%%%% N %%%%%%%%%%%%%%%%%%%%%

%%% I bought the cheap one to save money
"ones"      : NN_pl {HeadLex="one";Subcat=[];};
"one"       : NN_sg {HeadLex="one";Subcat=[];};

        %%% Three times the expected number
"time"      : NN_sg {HeadLex="time";Subcat=[NP_acc{}];};
"times"     : NN_pl {HeadLex="time";Subcat=[NP_acc{}];};

"matter"    : N {HeadLex="matter";NForm=noun;Subcat=[SBAR_argw{}];
                    Number=sg;Coord=-;Compound=-;};
%%% ... in common with ...
"common"    : NN_sg {HeadLex="common";Subcat=[PP{HeadLex="with";}];};


%%%%%%%%%% RP %%%%%%%%%%%%%%%%%%%%%

"about"     : RP {HeadLex="about";};
"across"    : RP {HeadLex="across";};
"along"     : RP {HeadLex="along";};
"apart"     : RP {HeadLex="apart";};
"around"    : RP {HeadLex="around";};
"aside"     : RP {HeadLex="aside";};
"away"      : RP {HeadLex="away";};
"back"      : RP {HeadLex="back";};
"behind"    : RP {HeadLex="behind";};
"by"        : RP {HeadLex="by";};
"down"      : RP {HeadLex="down";};
"for"       : RP {HeadLex="for";};
"forth"     : RP {HeadLex="forth";};
```

```
"forward"   : RP {HeadLex="forward";};
"in"        : RP {HeadLex="in";};
"off"       : RP {HeadLex="off";};
"on"        : RP {HeadLex="on";};
"open"      : RP {HeadLex="open";};
"out"       : RP {HeadLex="out";};
"over"      : RP {HeadLex="over";};
"through"   : RP {HeadLex="through";};
"to"        : RP {HeadLex="to";};
"together"  : RP {HeadLex="together";};
"up"        : RP {HeadLex="up";};
"upon"      : RP {HeadLex="upon";};


%%%%%%%%%% V %%%%%%%%%%%%%%%%%%%%%%


"am"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NP_sg{Person=1st;}];};
"'m"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NP_sg{Person=1st;}];};
"are"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NP_sg{Person=2nd;}];};
"'re"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NP_sg{Person=2nd;}];};
"is"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_3s{}];};
"is"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},SBAR{}];};
"is"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},VP{}];};
"'s"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_3s{}];};
"'s"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},SBAR{}];};
"'s"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},VP{}];};
"ai"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_3s{}];};
"ai"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},SBAR{}];};
"ai"        : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},VP{}];};
"are"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_pl{}];};
"'re"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_pl{}];};
"was"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_sg{Person=(1st,3rd);}];};
"was"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},SBAR{}];};
"was"       : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},VP{}];};
"were"      : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_sg{Person=2nd;}];};
"were"      : BE {VForm=fin;Subcat=[VP_prp_pas_inf{},NPe_pl{}];};
"be"        : BE {VForm=bse;Subcat=[VP_prp_pas_inf{},*];};
"being"     : BE {VForm=prp;Subcat=[VP_prp_pas_inf{},*];};
"been"      : BE {VForm=pap;Subcat=[VP_prp_pas_inf{},*];};


"has"       : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_3s{}];};
"'s"        : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_3s{}];};
"have"      : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_n3s{}];};
"have"      : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_pl{}];};
"'ve"       : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_n3s{}];};
"'ve"       : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_pl{}];};
"had"       : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_nom{}];};
"'d"        : HAVE {VForm=fin;Subcat=[VP_pap{},NPe_nom{}];};
"have"      : HAVE {VForm=bse;Subcat=[VP_pap{},NP{}];};
"having"    : HAVE {VForm=prp;Subcat=[VP_pap{},NP{}];};

"does"      : DO {VForm=fin;Subcat=[VP_bse{},NPe_3s{}];};
"do"        : DO {VForm=fin;Subcat=[VP_bse{},NPe_n3s{}];};
"do"        : DO {VForm=fin;Subcat=[VP_bse{},NPe_pl{}];};
"did"       : DO {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
```

```
"do"        : DO {VForm=bse;Subcat=[VP_bse{},NP_nom{Person=2nd;}];};

"shall"     : WILL {VForm=fin;Subcat=[VP_bse{},NP_nom{Person=1st;}];};
"will"      : WILL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"'ll"       : WILL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"wo"        : WILL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"would"     : WILL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"'d"        : WILL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};

"will"      : WILL {VForm=fin;Subcat=[VP_bse{},SBAR_argx{}];};
"'ll"       : WILL {VForm=fin;Subcat=[VP_bse{},SBAR_argx{}];};
"wo"        : WILL {VForm=fin;Subcat=[VP_bse{},SBAR_argx{}];};
"would"     : WILL {VForm=fin;Subcat=[VP_bse{},SBAR_argx{}];};
"'d"        : WILL {VForm=fin;Subcat=[VP_bse{},SBAR_argx{}];};

"will"      : WILL {VForm=fin;Subcat=[VP_bse{},VP_prp_inf{}];};
"'ll"       : WILL {VForm=fin;Subcat=[VP_bse{},VP_prp_inf{}];};
"wo"        : WILL {VForm=fin;Subcat=[VP_bse{},VP_prp_inf{}];};
"would"     : WILL {VForm=fin;Subcat=[VP_bse{},VP_prp_inf{}];};
"'d"        : WILL {VForm=fin;Subcat=[VP_bse{},VP_prp_inf{}];};

"can"       : CAN {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"cannot"    : CAN {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"ca"        : CAN {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"could"     : CAN {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};

"shall"     : SHALL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"sha"       : SHALL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};
"should"    : SHALL {VForm=fin;Subcat=[VP_bse{},NPe_nom{}];};

"may"       : AUX_fin {HeadLex="may";Subcat=[VP_bse{},NPe_nom{}];};
"might"     : AUX_fin {HeadLex="may";Subcat=[VP_bse{},NPe_nom{}];};

"must"      : AUX_fin {HeadLex="must";Subcat=[VP_bse{},NPe_nom{}];};
"ought"     : AUX_fin {HeadLex="ought";Subcat=[VP_inf{},NPe_nom{}];};
"used"      : AUX_fin {HeadLex="use";Subcat=[VP_inf{},NPe_nom{}];};


"to"        : AUX {HeadLex="to";VForm=inf;Subcat=[VP_bse{},NP{}];};
"na"        : AUX {HeadLex="to";VForm=inf;Subcat=[VP_bse{},NP{}];};

"going"     : AUX {HeadLex="go";VForm=prp;Subcat=[VP_inf{},*];};
"gon"       : AUX {HeadLex="go";VForm=prp;Subcat=[VP_inf{},*];};

"gets"      : GET {VForm=fin;Subcat=[VP_pas{},NPe_3s{}];};
"get"       : GET {VForm=fin;Subcat=[VP_pas{},NPe_n3s{}];};
"get"       : GET {VForm=fin;Subcat=[VP_pas{},NPe_pl{}];};
"got"       : GET {VForm=fin;Subcat=[VP_pas{},NPe_nom{}];};
"get"       : GET {VForm=bse;Subcat=[VP_pas{},NP{}];};
"getting"   : GET {VForm=prp;Subcat=[VP_pas{},NP{}];};



%%%%%%%%%% Multi Word Units %%%%%%%%%%%%%%%%%%%%%%

category MWL1 {};
category MWL2 {};
```

```
category MWL3 {};
category MWL4 {};
category MWL5 {};
category MWL6 {};
category MWL7 {};
category MWL8 {};
category MWL9 {};
category MWL10 {};
category MWL11 {};
category MWL12 {};
category MWL13 {};
category MWL14 {};
category MWL15 {};
category MWL16 {};
category MWL17 {};
category MWL18 {};
category MWL19 {};
category MWL20 {};
category MWL21 {};
category MWL22 {};
category MWL23 {};
category MWL24 {};
category MWL25 {};
category MWL26 {};
category MWL27 {};
category MWL28 {};
category MWL29 {};
category MWL30 {};
category MWL31 {};
category MWL32 {};
category MWL33 {};
category MWL34 {};
category MWL35 {};
category MWL36 {};
category MWL37 {};
category MWL38 {};
category MWL39 {};

"a"         : MWL20 {HeadLex="a";};
"according" : MWL22 {HeadLex="according";};
"along"     : MWL1  {HeadLex="along";};
"as"        : MWL25 {HeadLex="as";};
"at"        : MWL18 {HeadLex="at";};
"because"   : MWL3  {HeadLex="because";};
"between"   : MWL38 {HeadLex="between";};
"course"    : MWL37 {HeadLex="course";};
"except"    : MWL5  {HeadLex="except";};
"few"       : MWL21 {HeadLex="few";};
"for"       : MWL6  {HeadLex="for";};
"if"        : MWL28 {HeadLex="if";};
"in"        : MWL7  {HeadLex="in";};
"inside"    : MWL12 {HeadLex="inside";};
"instead"   : MWL30 {HeadLex="instead";};
"just"      : MWL33 {HeadLex="just";};
"least"     : MWL19 {HeadLex="least";};
"lieu"      : MWL8  {HeadLex="lieu";};
"long"      : MWL36 {HeadLex="long";};
```

```
"matter"    : MWL35 {HeadLex="matter";};
"next"      : MWL13 {HeadLex="next";};
"no"        : MWL34 {HeadLex="no";};
"not"       : MWL29 {HeadLex="not";};
"of"        : MWL4  {HeadLex="of";};
"one"       : MWL39 {HeadLex="one";};
"order"     : MWL10 {HeadLex="order";};
"out"       : MWL14 {HeadLex="out";};
"outside"   : MWL15 {HeadLex="outside";};
"per"       : MWL16 {HeadLex="per";};
"rather"    : MWL31 {HeadLex="rather";};
"se"        : MWL17 {HeadLex="se";};
"so"        : MWL23 {HeadLex="so";};
"spite"     : MWL9  {HeadLex="spite";};
"such"      : MWL27 {HeadLex="such";};
"than"      : MWL32 {HeadLex="than";};
"that"      : MWL24 {HeadLex="that";};
"to"        : MWL11 {HeadLex="to";};
"well"      : MWL26 {HeadLex="well";};
"with"      : MWL2  {HeadLex="with";};


P_{HeadLex="along_with";}  -> MWL1 {} `MWL2 {HeadLex=*;};
P_{HeadLex="according_to";}-> MWL22{} `MWL11 {HeadLex=*;};
P_{HeadLex="because_of";}  -> MWL3 {} `MWL4 {HeadLex=*;};
P_{HeadLex="except_for";}  -> MWL5 {} `MWL6 {HeadLex=*;};
P_{HeadLex="in_lieu_of";}  -> MWL7 {} MWL8 {} `MWL4 {HeadLex=*;};
P_{HeadLex="in_order_to";} -> MWL7 {} MWL10 {} `MWL11 {HeadLex=*;};
P_{HeadLex="in_spite_of";} -> MWL7 {} MWL9 {} `MWL4 {HeadLex=*;};
P_{HeadLex="inside_of";}   -> MWL12 {} `MWL4 {HeadLex=*;};
P_{HeadLex="next_to";}     -> MWL13 {} `MWL11 {HeadLex=*;};
P_{HeadLex="out_of";}      -> MWL14 {} `MWL4 {HeadLex=*;};
P_{HeadLex="outside_of";}  -> MWL15 {} `MWL4 {HeadLex=*;};


ADV_ {Degree=pos;HeadLex="at_least";Mod=(noun,verb,prep,adj,adv);}  ->
                                MWL18 {} `MWL19 {HeadLex=*;};
ADV_verb {HeadLex="per_se";}    -> MWL16 {} `MWL17 {HeadLex=*;};
ADV_verb {HeadLex="of_course";} -> MWL4  {} `MWL37 {HeadLex=*;};

DT {HeadLex="a_few";WhForm=-;Number=pl;}  -> MWL20 {} `MWL21 {HeadLex=*;};

PRO_3s {HeadLex="noone";} ->
        MWL34{} `MWL39 {HeadLex=*;};

C {HeadLex="along_with";CForm=coord2;}  -> MWL1{}  `MWL2  {HeadLex=*;};
C {HeadLex="as_well_as";CForm=coord2;}  -> MWL25{} `MWL26 {HeadLex=*;} MWL25{};
C {HeadLex="if_not";CForm=coord2;}      -> MWL28{} `MWL29 {HeadLex=*;};
C {HeadLex="instead_of";CForm=coord2;}  -> MWL30{} `MWL4  {HeadLex=*;};
C {HeadLex="rather_than";CForm=coord2;} -> MWL31{} `MWL32 {HeadLex=*;};
C {HeadLex="not_just";CForm=coord2;}    -> MWL29{} `MWL33 {HeadLex=*;};

C {HeadLex="so_that";CForm=adj;}        -> MWL23{} `MWL24 {HeadLex=*;};
C {HeadLex="as_long_as";CForm=adj;}     -> MWL25{}  MWL36 {} `MWL25{HeadLex=*;};

P_ {HeadLex="such_as";}                 -> MWL27{} `MWL25 {HeadLex=*;};
```

```
ADVP {WhForm=-;Not=-;Mod=verb;Degree=pos;} ->
        MWL34{} `MWL35 {HeadLex=*;} SBAR {CForm=quest;Slash=[];Commas=-;};

PP0 {HeadLex="in_between";Mod=(noun,verb,-);} ->
        `MWL7 {HeadLex=*;} MWL38 {};


#include "sub-lexicon"  % other, more regular lexical entries
```

Appendix C

The English LPCF Grammar

```
ADJ= -> ADJ_'
ADJ= -> ADV= ADJ_'
ADJ= -> DEG_ ADJ_'
ADJC1 -> ADJC1 COM_ ADJC1
         CONJ_ ADJC1'
ADJC1 -> ADJC1 COM_ ADJC1'
ADJC1 -> ADJC1 CONJ_ ADJC1'
ADJC1 -> ADJC1' PUN=
ADJC1 -> ADJ_C'
ADJC1 -> ADJ_C' PC1
ADJC1 -> ADJ_C' THATC1
ADJC1 -> ADJ_C' VTOC1
ADJC1 -> PCONJ_ ADJC1 COM_ ADJC1
         CONJ_ ADJC1'
ADJC1 -> PCONJ_ ADJC1 CONJ_ ADJC1'
ADJC1 -> PUNL_ ADJC1' PUNR_
ADJMOD -> ADJ_C'
ADJMOD -> ORD_'
ADJMOD -> VN_C'
ADJMOD -> VPASS_C'
ADJR_ -> ADJR'
ADJR_ -> ADV= ADJR'
ADJS_ -> ADJS'
ADJ_ -> ADJ'
ADJ_ -> ADJR_'
ADJ_ -> ADJS_'
ADJ_ -> ORD_'
ADJ_C -> ADJ= ADJ='
ADJ_C -> ADJ='
ADJ_C -> ADJ_C CONJ_ ADJ_C'
ADJ_C -> NC1 CONJ_ ADJ_C'
ADV= -> ADV= CONJ_ ADV='
ADV= -> ADV_'
ADV_ -> ADV'
ADV_ -> ADV_ ADV'
ADV_ -> DEG_ ADV'
ADV_C -> ADV='
ADV_C -> ADV=' PUN=
ADV_C -> DEG_'
ADV_C -> PUNL_ ADV=' PUNR_
ADV_C -> WHA_C'
AS_C -> AS' ADJC1
AS_C -> AS' NC1
AS_C -> AS' VGC1
AUX-ADV -> ADV='
AUX-ADV -> PC1'
CD_ -> ADV= CD_'
CD_ -> CD'
CD_ -> CD_ CD'
CD_ -> CD_ CONJ_ CD_'
CD_ -> DEG_ THAN_ CD_'
CD_ -> DEG_ THAN_ DETPL_'
COM_ -> COM'
CONJ_ -> CONJ'
CONJ_C -> CONJ'
CONJ_C -> CONJ' PUN=
DEG_ -> ADV= DEG'
DEG_ -> DEG'
DEG_ -> DEG_ DEG'
DETPL= -> DETPL_ CONJ_ DETPL_'
DETPL= -> DETPL_'
DETPL= -> DETPL_' ADV=
DETPL= -> S_ADV_ DETPL_'
DETPL_ -> ADV=' POS_
DETPL_ -> CD_ PREP_ CD_'
DETPL_ -> CD_'
DETPL_ -> DETPL'
DETPL_ -> DETPL' CD_
DETPL_ -> DETPL' NPL_ POS_
DETPL_ -> DETPL_ OF_ DETPL_'
DETPL_ -> DETPL_ OF_ N_C POS'
DETPL_ -> DETPL_ OF_ PRO$'
DETPL_ -> NC1 POS'
DETPL_ -> N_C POS'
DETPL_ -> PDET_ DETPL_'
DETPL_ -> PN= POS'
DETPL_ -> PRO$_'
DETSG= -> DETSG_ CONJ_ DETSG_'
```

```
DETSG= -> DETSG_'
DETSG= -> DETSG_' ADV=
DETSG= -> S_ADV_ DETSG_'
DETSG_ -> ADV'
DETSG_ -> ADV=' POS_
DETSG_ -> DETSG'
DETSG_ -> DETSG' NPL_ POS_
DETSG_ -> DETSG' NSG_ POS_
DETSG_ -> DETSG_' CD_
DETSG_ -> NC1 POS'
DETSG_ -> N_C POS'
DETSG_ -> PDET_ DETSG_'
DETSG_ -> PN= POS'
DETSG_ -> PRO$_'
IART_ -> IART'
ITJ= -> ITJ'
ITJ= -> ITJ= ITJ'
ITJ= -> ITJ= PUN= ITJ'
ITJ_C -> ITJ='
ITJ_C -> ITJ=' PUN=
ITJ_C -> PUNL_ ITJ=' PUNR_
M= -> ADV= M='
M= -> IART_ MSG_'
MD= -> MD_'
MD= -> MD_' AUX-ADV
MD_ -> ADV= MD'
MD_ -> MD'
MD_ -> VDF='
MSG_ -> ADJC1 MSG'
MSG_ -> MSG'
N-CHAIN -> N-CHAIN NPL='
N-CHAIN -> N-CHAIN NSG='
N-CHAIN -> N-CHAIN PN='
N-CHAIN -> NPL='
N-CHAIN -> NSG='
N-CHAIN -> PN='
NC1 -> ADJC1 CONJ_ NC1'
NC1 -> DETPL= PUNL_ NC1' PUNR_
NC1 -> DETSG= PUNL_ NC1' PUNR_
NC1 -> DETSG= VGC1'
NC1 -> DETSG= VGPC1'
NC1 -> NC1 COM_ NC1 COM_ CONJ_ NC1'
NC1 -> NC1 COM_ NC1 CONJ_ NC1'
NC1 -> NC1 CONJ_ NC1'
NC1 -> NC1' PUN=
NC1 -> NC1' VGC1
NC1 -> NC1' VPASSC1
NC1 -> N_C'
NC1 -> N_C' PC1
NC1 -> N_C' PC1 PC1
NC1 -> N_C' PC1 VTOC1
NC1 -> N_C' PC1 VTOPC1
NC1 -> N_C' RELC1
NC1 -> N_C' THATC1
NC1 -> N_C' VTOC1
NC1 -> N_C' VTOPC1
NC1 -> PCONJ_ NC1 CONJ_ NC1'
NC1 -> PN='
NC1 -> PN_ CD_'
NC1 -> POS= VGC1'
NC1 -> PROPL_'
NC1 -> PROSG_'
NC1 -> PUNL_ NC1' PUNR_
NC1 -> VGC1'
NC1 -> VGPC1'
NPL= -> ADJMOD ADJMOD NPL_'
NPL= -> ADJMOD COM_ ADJMOD NPL_'
NPL= -> ADJMOD CONJ_ ADJMOD NPL_'
NPL= -> ADJMOD NPL_'
NPL= -> NPL_'
NPL= -> PUNL_ NPL=' PUNR_
NPL_ -> ADJC1 NPL_'
NPL_ -> CD_ NPL_'
NPL_ -> N-CHAIN NPL'
NPL_ -> NPL'
NPL_ -> NPL_ NPL_'
NPL_ -> NSG_ NPL_'
NPL_ -> PN_ NPL_'
NPL_ -> VPASS_ NPL_'
NSG= -> ADJMOD ADJMOD NSG_'
NSG= -> ADJMOD COM_ ADJMOD NSG_'
NSG= -> ADJMOD CONJ_ ADJMOD NSG_'
NSG= -> ADJMOD NSG_'
NSG= -> NSG_'
NSG= -> PUNL_ NSG=' PUNR_
NSG_ -> ADJC1 NSG_'
NSG_ -> CD_ NSG_'
NSG_ -> CD_'
NSG_ -> N-CHAIN NSG_'
NSG_ -> NPL_ NSG_'
NSG_ -> NSG'
NSG_ -> NSG_ NSG_'
NSG_ -> PN_ NSG_'
NSG_ -> VPASS_ NSG_'
N_C -> ADJC1 CONJ_ N_C'
N_C -> CD_'
N_C -> DETPL= ADJC1'
N_C -> DETPL= ADJS_'
N_C -> DETPL= NPL='
N_C -> DETPL= OF_ PROPL'
N_C -> DETPL= PN='
N_C -> DETPL= PROPL_'
N_C -> DETSG= ADJC1'
N_C -> DETSG= ADJS_'
N_C -> DETSG= CD_'
N_C -> DETSG= NSG='
```

```
N_C -> DETSG= ORD_'
N_C -> DETSG= PN='
N_C -> DETSG= PROSG_'
N_C -> M= NPL='
N_C -> M= OF_ NPL='
N_C -> M= OF_ NSG='
N_C -> NPL='
N_C -> NSG='
N_C -> N_C COM_ N_C COM_ CONJ_ N_C'
N_C -> N_C COM_ N_C CONJ_ N_C'
N_C -> N_C CONJ_ N_C'
N_C -> N_C POS_'
N_C -> N_C' PUN=
N_C -> PCONJ_ N_C CONJ_ N_C'
N_C -> PUNL_ N_C' PUNR_
N_C -> WHN_C'
OF_ -> OF'
ORD_ -> ORD'
PART_C -> ADV= PART'
PART_C -> PART'
PART_C -> PART_C CONJ_ PART_C'
PART_C -> PART_C PART_C'
PART_C -> PART_C' PUN=
PC1 -> PC1 COM_ PC1 COM_ CONJ_ PC1'
PC1 -> PC1 COM_ PC1 CONJ_ PC1'
PC1 -> PC1 CONJ_ PC1'
PC1 -> PC1' PUN=
PC1 -> PCONJ_ PC1 CONJ_ PC1'
PC1 -> PCONJ_ P_ST_C CONJ_ PC1'
PC1 -> PUNL_ PC1' PUNR_
PC1 -> P_C'
PC1 -> P_ST_C CONJ_ PC1'
PCONJ_ -> PCONJ'
PDET_ -> ADV= PDET'
PDET_ -> ADV= PDET_'
PDET_ -> PDET'
PER_C -> PER'
PN= -> ADJC1 PN_'
PN= -> PN_'
PN= -> PN_' COM_ PN_
PN= -> VN= PN_'
PN_ -> ADJC1 PN_'
PN_ -> CD_ PN_'
PN_ -> NPL_ PN_'
PN_ -> NSG_ PN_'
PN_ -> PN'
PN_ -> PN' CD_
PN_ -> PN_ PN'
PN_ -> VPASS_ PN_'
POS= -> N_C POS_'
POS_ -> POS'
PREP_ -> ADV= PREP'
PREP_ -> PREP'
PRO$_ -> PRO$'
PRO$_ -> PRO$' DETPL_
PRO$_ -> PRO$' DETSG_
PROPL_ -> PROPL'
PROSG_ -> ADV= PROSG'
PROSG_ -> PROSG'
PUN= -> LB'
PUN= -> PUN_'
PUN= -> RB'
PUNL_ -> LB'
PUNL_ -> PUN_ PUNL_'
PUNL_ -> PUN_'
PUNR_ -> COM_ PUNR_'
PUNR_ -> PUN_'
PUNR_ -> RB'
PUN_ -> COM_'
PUN_ -> PUN'
P_C -> PREP_ CONJ_ PREP_' NC1
P_C -> PREP_' ADV=
P_C -> PREP_' NC1
P_C -> PREP_' P_C
P_C -> P_ST_C' COM_
P_ST_C -> PREP_'
P_ST_C -> P_ST_C' PUN=
RELC1 -> PUNL_ RELC1' PUNR_
RELC1 -> RELC1 CONJ_ RELC1'
RELC1 -> RELC1' PUN=
RELC1 -> THAT_C NC1 VFC1'
RELC1 -> THAT_C NC1 VFPC1'
RELC1 -> THAT_C VFC1'
RELC1 -> THAT_C VFPC1'
RELC1 -> WHA_C NC1 VFC1'
RELC1 -> WHA_C NC1 VFPC1'
RELC1 -> WHN_C NC1 VFC1'
RELC1 -> WHN_C NC1 VFPC1'
RELC1 -> WHN_C VFC1'
RELC1 -> WHN_C VFPC1'
RELC1 -> WHP_C NC1 VFC1'
RELC1 -> WHP_C NC1 VFPC1'
RELC1 -> WHP_C VFC1'
RELC1 -> WHP_C VFPC1'
S -> ADJC1' PER_C
S -> ADV_C' PER_C
S -> ITJ_C' PER_C
S -> NC1' PER_C
S -> PC1' PER_C
S -> RELC1' PER_C
S -> SMAJ' PER_C
S -> SUBC1' PER_C
S -> VFC1' PER_C
SMAJ -> CONJ_ S_C'
SMAJ -> PCONJ_ S_C CONJ_ S_C'
SMAJ -> PUN= S_C'
```

```
SMAJ -> PUNL_ S_C' PUN= PUNR_
SMAJ -> PUNL_ S_C' PUNR_
SMAJ -> S_C CONJ_C S_C'
SMAJ -> S_C'
SMAJ -> S_C' PUN=
SUBC1 -> PUNL_ SUBC1' PUNR_
SUBC1 -> SUBC1' PUN=
SUBC1 -> SUB_C' SMAJ
SUBC1 -> SUB_C' VBASEC1
SUBC1 -> SUB_C' VGC1
SUBC1 -> SUB_C' VPASSC1
SUB_C -> ADV= SUB'
SUB_C -> SUB'
SUB_C -> SUB' ADV=
S_ADV_ -> ADV'
S_C -> ADV= NC1 VFC1'
S_C -> ADV= NC1 VFPC1'
S_C -> NC1 RELC1 VFC1'
S_C -> NC1 RELC1 VFPC1'
S_C -> NC1 VFC1'
S_C -> NC1 VFPC1'
S_C -> NC1 VPASSC1 VFC1'
S_C -> NC1 VPASSC1 VFPC1'
S_C -> PC1 NC1 RELC1 VFC1'
S_C -> PC1 NC1 VFC1'
S_C -> PC1 NC1 VFPC1'
S_C -> PC1 NC1 VPASSC1 VFC1'
S_C -> PC1 NC1 VPASSC1 VFPC1'
S_C -> SUBC1 NC1 RELC1 VFC1'
S_C -> SUBC1 NC1 VFC1'
S_C -> SUBC1 NC1 VFPC1'
S_C -> WHN_C VFC1'
S_C -> WHN_C VFPC1'
THAN_ -> THAN'
THATC1 -> THATC1' PUN=
THATC1 -> THAT_C'
THATC1 -> THAT_C' SMAJ
THAT_C -> THAT'
TO= -> ADV= TO_'
TO= -> TO_'
TO_ -> TO'
VBASE= -> ADV= VBASE_'
VBASE= -> VBASE_'
VBASEC1 -> PUNL_ VBASEC1' PUNR_
VBASEC1 -> VBASEC1 CONJ_ VBASEC1'
VBASEC1 -> VBASEC1 CONJ_ VBASEC1' ADV=
VBASEC1 -> VBASEC1' PUN=
VBASEC1 -> VBASEC1' PUN= ADV=
VBASEC1 -> VBASE_C'
VBASEC1 -> VBASE_C' ADJC1
VBASEC1 -> VBASE_C' ADJC1 ADV=
VBASEC1 -> VBASE_C' ADJC1 PC1
VBASEC1 -> VBASE_C' ADJC1 PC1 ADV=
VBASEC1 -> VBASE_C' ADV=
VBASEC1 -> VBASE_C' ADV= NC1
VBASEC1 -> VBASE_C' ADV= PC1
VBASEC1 -> VBASE_C' ITJ_C
VBASEC1 -> VBASE_C' NC1
VBASEC1 -> VBASE_C' NC1 ADJC1
VBASEC1 -> VBASE_C' NC1 ADV=
VBASEC1 -> VBASE_C' NC1 AS_C
VBASEC1 -> VBASE_C' NC1 NC1
VBASEC1 -> VBASE_C' NC1 NC1 ADV=
VBASEC1 -> VBASE_C' NC1 PART_C
VBASEC1 -> VBASE_C' NC1 PART_C ADV=
VBASEC1 -> VBASE_C' NC1 PART_C PC1
VBASEC1 -> VBASE_C' NC1 PC1
VBASEC1 -> VBASE_C' NC1 PC1 ADV=
VBASEC1 -> VBASE_C' NC1 VBASEC1
VBASEC1 -> VBASE_C' NC1 VGC1
VBASEC1 -> VBASE_C' NC1 VTOC1
VBASEC1 -> VBASE_C' PART_C
VBASEC1 -> VBASE_C' PART_C ADV=
VBASEC1 -> VBASE_C' PART_C NC1
VBASEC1 -> VBASE_C' PART_C NC1 PC1
VBASEC1 -> VBASE_C' PART_C PC1
VBASEC1 -> VBASE_C' PART_C PC1 ADV=
VBASEC1 -> VBASE_C' PC1
VBASEC1 -> VBASE_C' PC1 ADV=
VBASEC1 -> VBASE_C' PC1 PC1
VBASEC1 -> VBASE_C' PC1 SMAJ
VBASEC1 -> VBASE_C' PC1 VGC1
VBASEC1 -> VBASE_C' PC1 VTOC1
VBASEC1 -> VBASE_C' SMAJ
VBASEC1 -> VBASE_C' SMAJ ADV=
VBASEC1 -> VBASE_C' THATC1
VBASEC1 -> VBASE_C' THATC1 ADV=
VBASEC1 -> VBASE_C' VGC1
VBASEC1 -> VBASE_C' VTOC1
VBASEC1 -> VBASE_C' VTOC1 ADV=
VBASEC1 -> VBASE_C' VTOC1 VTOC1
VBASEPC1 -> VBASEPC1' PUN=
VBASEPC1 -> VBASEP_C'
VBASEPC1 -> VBASEP_C' ADJC1
VBASEPC1 -> VBASEP_C' ADV=
VBASEPC1 -> VBASEP_C' ADV= NC1
VBASEPC1 -> VBASEP_C' ADV= PC1
VBASEPC1 -> VBASEP_C' AS_C
VBASEPC1 -> VBASEP_C' NC1
VBASEPC1 -> VBASEP_C' NC1 PC1
VBASEPC1 -> VBASEP_C' PART_C
VBASEPC1 -> VBASEP_C' PART_C ADV=
VBASEPC1 -> VBASEP_C' PART_C PC1
VBASEPC1 -> VBASEP_C' PC1
VBASEPC1 -> VBASEP_C' PC1 ADV=
VBASEPC1 -> VBASEP_C' PC1 PC1
```

```
VBASEPC1 -> VBASEP_C' PC1 SMAJ
VBASEPC1 -> VBASEP_C' PC1 THATC1
VBASEPC1 -> VBASEP_C' PC1 VTOC1
VBASEPC1 -> VBASEP_C' SUBC1
VBASEPC1 -> VBASEP_C' THATC1
VBASEPC1 -> VBASEP_C' VTOC1
VBASEP_C -> VBASEP|'
VBASEP| -> VBBASE= VPASS='
VBASE_ -> VBASE'
VBASE_C -> VBASE|'
VBASE| -> VBASE= CONJ_C VBASE='
VBASE| -> VBASE='
VBASE| -> VBBASE= VG|'
VBASE| -> VBBASE='
VBASE| -> VHBASE= VN|'
VBBASE= -> VBBASE_'
VBBASE= -> VBBASE_' ADV=
VBBASE_ -> ADV= VBBASE'
VBBASE_ -> VBBASE'
VBBASE| -> VBBASE='
VBBASE| -> VHBASE= VBN='
VBF= -> MD= VBBASE|'
VBF= -> VBF_ VBG='
VBF= -> VBF_'
VBF= -> VBF_' AUX-ADV
VBF= -> VHF= VBN='
VBF_ -> ADV= VBF'
VBF_ -> VBF'
VBG= -> VBG_'
VBG_ -> ADV= VBG'
VBG_ -> VBG'
VBN= -> VBN_'
VBN= -> VBN_' ADV=
VBN= -> VBN_' ADV= VBG=
VBN= -> VBN_' VBG=
VBN_ -> ADV= VBN'
VBN_ -> VBN'
VDF= -> VDF_'
VDF= -> VDF_' AUX-ADV
VDF_ -> ADV= VDF'
VDF_ -> VDF'
VF= -> ADV= VF_'
VF= -> VF_'
VFC1 -> PUNL_ VFC1' PUNR_
VFC1 -> VBF=' PART_C VTOC1
VFC1 -> VBF=' PART_C VTOPC1
VFC1 -> VFC1 COM_ VFC1 COM_
        CONJ_ VFC1'
VFC1 -> VFC1 COM_ VFC1 CONJ_ VFC1'
VFC1 -> VFC1 CONJ_ VFC1'
VFC1 -> VFC1' PUN=
VFC1 -> VF_C'
VFC1 -> VF_C' ADJC1
VFC1 -> VF_C' ADJC1 ADV=
VFC1 -> VF_C' ADJC1 PC1
VFC1 -> VF_C' ADJC1 PC1 ADV=
VFC1 -> VF_C' ADJC1 THATC1
VFC1 -> VF_C' ADJC1 THATC1 ADV=
VFC1 -> VF_C' ADJC1 VTOC1
VFC1 -> VF_C' ADJC1 VTOC1 ADV=
VFC1 -> VF_C' ADJC1 VTOPC1
VFC1 -> VF_C' ADJC1 VTOPC1 ADV=
VFC1 -> VF_C' ADV=
VFC1 -> VF_C' ADV= NC1
VFC1 -> VF_C' ADV= PC1
VFC1 -> VF_C' NC1
VFC1 -> VF_C' NC1 ADJC1
VFC1 -> VF_C' NC1 ADJC1 ADV=
VFC1 -> VF_C' NC1 ADV=
VFC1 -> VF_C' NC1 AS_C
VFC1 -> VF_C' NC1 NC1
VFC1 -> VF_C' NC1 NC1 ADV=
VFC1 -> VF_C' NC1 NC1 PC1
VFC1 -> VF_C' NC1 NC1 PC1 ADV=
VFC1 -> VF_C' NC1 PART_C
VFC1 -> VF_C' NC1 PART_C ADV=
VFC1 -> VF_C' NC1 PART_C PC1
VFC1 -> VF_C' NC1 PART_C PC1 ADV=
VFC1 -> VF_C' NC1 PC1
VFC1 -> VF_C' NC1 PC1 ADV=
VFC1 -> VF_C' NC1 PC1 NC1
VFC1 -> VF_C' NC1 PC1 NC1 ADV=
VFC1 -> VF_C' NC1 PC1 PC1
VFC1 -> VF_C' NC1 THATC1
VFC1 -> VF_C' NC1 VBASEC1
VFC1 -> VF_C' NC1 VBASEC1 ADV=
VFC1 -> VF_C' NC1 VGC1
VFC1 -> VF_C' NC1 VGC1 ADV=
VFC1 -> VF_C' NC1 VGPC1
VFC1 -> VF_C' NC1 VTOC1
VFC1 -> VF_C' NC1 VTOC1 ADV=
VFC1 -> VF_C' NC1 VTOPC1
VFC1 -> VF_C' NC1 VTOPC1 ADV=
VFC1 -> VF_C' PART_C
VFC1 -> VF_C' PART_C ADV=
VFC1 -> VF_C' PART_C NC1
VFC1 -> VF_C' PART_C NC1 ADV=
VFC1 -> VF_C' PART_C NC1 PC1
VFC1 -> VF_C' PART_C NC1 PC1 ADV=
VFC1 -> VF_C' PART_C PC1
VFC1 -> VF_C' PART_C PC1 ADV=
VFC1 -> VF_C' PART_C PC1 THATC1
VFC1 -> VF_C' PART_C SMAJ
VFC1 -> VF_C' PART_C THATC1
VFC1 -> VF_C' PC1
VFC1 -> VF_C' PC1 ADV=
```

```
VFC1 -> VF_C' PC1 NC1
VFC1 -> VF_C' PC1 NC1 NC1
VFC1 -> VF_C' PC1 NC1 NC1 ADV=
VFC1 -> VF_C' PC1 PC1
VFC1 -> VF_C' PC1 PC1 ADV=
VFC1 -> VF_C' PC1 SMAJ
VFC1 -> VF_C' PC1 SMAJ ADV=
VFC1 -> VF_C' PC1 THATC1
VFC1 -> VF_C' PC1 VGC1
VFC1 -> VF_C' PC1 VGC1 ADV=
VFC1 -> VF_C' PC1 VTOC1
VFC1 -> VF_C' PC1 VTOC1 ADV=
VFC1 -> VF_C' PC1 VTOPC1
VFC1 -> VF_C' PC1 VTOPC1 ADV=
VFC1 -> VF_C' P_ST_C
VFC1 -> VF_C' P_ST_C ADV=
VFC1 -> VF_C' SMAJ
VFC1 -> VF_C' SMAJ ADV=
VFC1 -> VF_C' SUBC1
VFC1 -> VF_C' SUBC1 ADV=
VFC1 -> VF_C' THATC1
VFC1 -> VF_C' THATC1 ADV=
VFC1 -> VF_C' VBASEC1
VFC1 -> VF_C' VBASEC1 ADV=
VFC1 -> VF_C' VGC1
VFC1 -> VF_C' VGC1 ADV=
VFC1 -> VF_C' VTOC1
VFC1 -> VF_C' VTOC1 ADV=
VFC1 -> VF_C' VTOC1 VTOC1
VFC1 -> VF_C' VTOC1 VTOPC1
VFC1 -> VF_C' VTOPC1
VFC1 -> VF_C' VTOPC1 ADV=
VFC1 -> VF_C' VTOPC1 VTOC1
VFC1 -> VF_C' VTOPC1 VTOPC1
VFPC1 -> PUNL_ VFPC1' PUNR_
VFPC1 -> VFPC1' PUN=
VFPC1 -> VFPC1' PUN= ADV=
VFPC1 -> VFP_C'
VFPC1 -> VFP_C' ADJC1
VFPC1 -> VFP_C' ADV=
VFPC1 -> VFP_C' ADV= NC1
VFPC1 -> VFP_C' ADV= PC1
VFPC1 -> VFP_C' AS_C
VFPC1 -> VFP_C' NC1
VFPC1 -> VFP_C' NC1 ADV=
VFPC1 -> VFP_C' NC1 NC1
VFPC1 -> VFP_C' NC1 NC1 ADV=
VFPC1 -> VFP_C' NC1 PC1
VFPC1 -> VFP_C' NC1 PC1 ADV=
VFPC1 -> VFP_C' NC1 PC1 PC1
VFPC1 -> VFP_C' PART_C
VFPC1 -> VFP_C' PART_C ADV=
VFPC1 -> VFP_C' PART_C PC1
VFPC1 -> VFP_C' PART_C PC1 ADV=
VFPC1 -> VFP_C' PART_C SMAJ
VFPC1 -> VFP_C' PC1
VFPC1 -> VFP_C' PC1 ADV=
VFPC1 -> VFP_C' PC1 PC1
VFPC1 -> VFP_C' PC1 PC1 ADV=
VFPC1 -> VFP_C' PC1 SMAJ
VFPC1 -> VFP_C' PC1 THATC1
VFPC1 -> VFP_C' PC1 THATC1 ADV=
VFPC1 -> VFP_C' PC1 VGC1
VFPC1 -> VFP_C' PC1 VGC1 ADV=
VFPC1 -> VFP_C' PC1 VTOC1
VFPC1 -> VFP_C' P_ST_C
VFPC1 -> VFP_C' SMAJ
VFPC1 -> VFP_C' SMAJ ADV=
VFPC1 -> VFP_C' SUBC1
VFPC1 -> VFP_C' SUBC1 ADV=
VFPC1 -> VFP_C' THATC1
VFPC1 -> VFP_C' THATC1 ADV=
VFPC1 -> VFP_C' VTOC1
VFPC1 -> VFP_C' VTOC1 ADV=
VFP_C -> VBF= VPASS='
VFP_C -> VHF= VNP='
VF_ -> VF'
VF_C -> MD= VBASE|'
VF_C -> MD='
VF_C -> VBF= VG|'
VF_C -> VBF='
VF_C -> VDF= VBASE|'
VF_C -> VDF='
VF_C -> VF='
VF_C -> VF_C CONJ_ VF_C'
VF_C -> VHF= VN|'
VF_C -> VHF='
VG= -> ADV= VG_'
VG= -> VG_'
VGC1 -> PUNL_ VGC1' PUNR_
VGC1 -> VGC1 COM_ VGC1 COM_ CONJ_ VGC1'
VGC1 -> VGC1 COM_ VGC1 CONJ_ VGC1'
VGC1 -> VGC1 CONJ_ VGC1'
VGC1 -> VGC1' PUN=
VGC1 -> VG_C'
VGC1 -> VG_C' ADJC1
VGC1 -> VG_C' ADJC1 THATC1
VGC1 -> VG_C' ADJC1 VTOC1
VGC1 -> VG_C' NC1
VGC1 -> VG_C' NC1 ADJC1
VGC1 -> VG_C' NC1 AS_C
VGC1 -> VG_C' NC1 NC1
VGC1 -> VG_C' NC1 PART_C
VGC1 -> VG_C' NC1 PART_C PC1
VGC1 -> VG_C' NC1 PC1
VGC1 -> VG_C' NC1 PC1 PC1
```

```
VGC1 -> VG_C' NC1 VBASEC1
VGC1 -> VG_C' NC1 VGC1
VGC1 -> VG_C' NC1 VTOC1
VGC1 -> VG_C' PART_C
VGC1 -> VG_C' PART_C NC1
VGC1 -> VG_C' PART_C NC1 PC1
VGC1 -> VG_C' PART_C PC1
VGC1 -> VG_C' PART_C SMAJ
VGC1 -> VG_C' PART_C THATC1
VGC1 -> VG_C' PC1
VGC1 -> VG_C' PC1 PC1
VGC1 -> VG_C' PC1 SMAJ
VGC1 -> VG_C' PC1 THATC1
VGC1 -> VG_C' PC1 VGC1
VGC1 -> VG_C' PC1 VTOC1
VGC1 -> VG_C' SMAJ
VGC1 -> VG_C' THATC1
VGC1 -> VG_C' VTOC1
VGC1 -> VG_C' VTOC1 VTOC1
VGPC1 -> VGPC1' PUN=
VGPC1 -> VGP_C'
VGPC1 -> VGP_C' AS_C
VGPC1 -> VGP_C' PC1
VGP_C -> VBG= VPASS_C'
VG_ -> VG'
VG_C -> VBG= VPASS='
VG_C -> VBG='
VG_C -> VG='
VG_C -> VG_C COM_ VG_C CONJ_ VG_C'
VG_C -> VG_C CONJ_ VG_C'
VG_C -> VHG=' VN|
VG| -> VBG='
VG| -> VG='
VHBASE= -> VHBASE_'
VHBASE_ -> ADV= VHBASE'
VHBASE_ -> VHBASE'
VHF= -> MD= VHBASE_'
VHF= -> VBF= VHG='
VHF= -> VHF= VHN='
VHF= -> VHF_'
VHF= -> VHF_' AUX-ADV
VHF_ -> ADV= VHF'
VHF_ -> VHF'
VHG= -> VHG_'
VHG_ -> VHG'
VHN= -> ADV= VHN_'
VHN= -> VHN_'
VHN_ -> VHN'
VN= -> ADV= VN_'
VN= -> VN_'
VNP= -> VBN= VPASS='
VN_ -> VN'
VN_C -> VN_C CONJ_ VN_C'
VN_C -> VN|'
VN| -> VBN= VG|'
VN| -> VBN='
VN| -> VN='
VPASS= -> ADV= VPASS_'
VPASS= -> VPASS_'
VPASSC1 -> PUNL_ VPASSC1' PUNR_
VPASSC1 -> VPASSC1 CONJ_ VPASSC1'
VPASSC1 -> VPASSC1' PUN=
VPASSC1 -> VPASS_C'
VPASSC1 -> VPASS_C' ADJC1
VPASSC1 -> VPASS_C' AS_C
VPASSC1 -> VPASS_C' NC1
VPASSC1 -> VPASS_C' NC1 ADJC1
VPASSC1 -> VPASS_C' NC1 NC1
VPASSC1 -> VPASS_C' NC1 PART_C
VPASSC1 -> VPASS_C' NC1 PC1
VPASSC1 -> VPASS_C' NC1 VGC1
VPASSC1 -> VPASS_C' NC1 VTOC1
VPASSC1 -> VPASS_C' PART_C
VPASSC1 -> VPASS_C' PART_C NC1
VPASSC1 -> VPASS_C' PART_C PC1
VPASSC1 -> VPASS_C' PC1
VPASSC1 -> VPASS_C' PC1 PC1
VPASSC1 -> VPASS_C' PC1 VGC1
VPASSC1 -> VPASS_C' PC1 VTOC1
VPASSC1 -> VPASS_C' SMAJ
VPASSC1 -> VPASS_C' THATC1
VPASSC1 -> VPASS_C' VGC1
VPASSC1 -> VPASS_C' VTOC1
VPASS_ -> ADV= VPASS'
VPASS_ -> VPASS'
VPASS_C -> VBN= VPASS_'
VPASS_C -> VPASS_'
VPASS_C -> VPASS_C CONJ_ VPASS_C'
VTOC1 -> PUNL_ VTOC1' PUNR_
VTOC1 -> VTOC1 CONJ_ VTOC1'
VTOC1 -> VTOC1 CONJ_ VTOC1' ADV=
VTOC1 -> VTOC1' PUN=
VTOC1 -> VTOC1' PUN= ADV=
VTOC1 -> VTO_C'
VTOC1 -> VTO_C' ADJC1
VTOC1 -> VTO_C' ADJC1 ADV=
VTOC1 -> VTO_C' ADJC1 PC1
VTOC1 -> VTO_C' ADJC1 VTOC1
VTOC1 -> VTO_C' ADJC1 VTOC1 ADV=
VTOC1 -> VTO_C' ADV=
VTOC1 -> VTO_C' ADV= NC1
VTOC1 -> VTO_C' ADV= PC1
VTOC1 -> VTO_C' ITJ_C
VTOC1 -> VTO_C' NC1
VTOC1 -> VTO_C' NC1 ADJC1
VTOC1 -> VTO_C' NC1 ADV=
```

```
VTOC1 -> VTO_C' NC1 AS_C
VTOC1 -> VTO_C' NC1 NC1
VTOC1 -> VTO_C' NC1 NC1 ADV=
VTOC1 -> VTO_C' NC1 PART_C
VTOC1 -> VTO_C' NC1 PART_C ADV=
VTOC1 -> VTO_C' NC1 PART_C PC1
VTOC1 -> VTO_C' NC1 PC1
VTOC1 -> VTO_C' NC1 PC1 ADV=
VTOC1 -> VTO_C' NC1 PC1 PC1
VTOC1 -> VTO_C' NC1 VBASEC1
VTOC1 -> VTO_C' NC1 VGC1
VTOC1 -> VTO_C' NC1 VGC1 ADV=
VTOC1 -> VTO_C' NC1 VTOC1
VTOC1 -> VTO_C' NC1 VTOC1 ADV=
VTOC1 -> VTO_C' PART_C
VTOC1 -> VTO_C' PART_C ADV=
VTOC1 -> VTO_C' PART_C NC1
VTOC1 -> VTO_C' PART_C NC1 PC1
VTOC1 -> VTO_C' PART_C PC1
VTOC1 -> VTO_C' PART_C PC1 ADV=
VTOC1 -> VTO_C' PART_C SMAJ
VTOC1 -> VTO_C' PART_C THATC1
VTOC1 -> VTO_C' PC1
VTOC1 -> VTO_C' PC1 ADV=
VTOC1 -> VTO_C' PC1 PC1
VTOC1 -> VTO_C' PC1 PC1 ADV=
VTOC1 -> VTO_C' PC1 SMAJ
VTOC1 -> VTO_C' PC1 THATC1
VTOC1 -> VTO_C' PC1 VGC1
VTOC1 -> VTO_C' PC1 VGC1 ADV=
VTOC1 -> VTO_C' PC1 VTOC1
VTOC1 -> VTO_C' SMAJ
VTOC1 -> VTO_C' SMAJ ADV=
VTOC1 -> VTO_C' SUBC1
VTOC1 -> VTO_C' SUBC1 ADV=
VTOC1 -> VTO_C' THATC1
VTOC1 -> VTO_C' THATC1 ADV=
VTOC1 -> VTO_C' VGC1
VTOC1 -> VTO_C' VGC1 ADV=
VTOC1 -> VTO_C' VTOC1
VTOC1 -> VTO_C' VTOC1 ADV=
VTOPC1 -> VTOPC1' PUN=
VTOPC1 -> VTOPC1' PUN= ADV=
VTOPC1 -> VTOP_C'
VTOPC1 -> VTOP_C' ADJC1
VTOPC1 -> VTOP_C' ADV=
VTOPC1 -> VTOP_C' ADV= NC1
VTOPC1 -> VTOP_C' ADV= PC1
VTOPC1 -> VTOP_C' AS_C
VTOPC1 -> VTOP_C' NC1
VTOPC1 -> VTOP_C' NC1 ADV=
VTOPC1 -> VTOP_C' NC1 PC1
VTOPC1 -> VTOP_C' NC1 PC1 PC1
VTOPC1 -> VTOP_C' PART_C
VTOPC1 -> VTOP_C' PART_C ADV=
VTOPC1 -> VTOP_C' PART_C PC1
VTOPC1 -> VTOP_C' PC1
VTOPC1 -> VTOP_C' PC1 ADV=
VTOPC1 -> VTOP_C' PC1 PC1
VTOPC1 -> VTOP_C' PC1 SMAJ
VTOPC1 -> VTOP_C' SUBC1
VTOPC1 -> VTOP_C' SUBC1 ADV=
VTOPC1 -> VTOP_C' THATC1
VTOPC1 -> VTOP_C' VTOC1
VTOP_C -> TO= VBASEP|'
VTO_C -> TO= VBASE|'
VTO_C -> TO= VBBASE='
VTO_C -> TO='
WHA_C -> ADV= WHADV'
WHA_C -> WHADV'
WHA_C -> WHA_C CONJ_ WHA_C'
WHA_C -> WHDEG_ ADJC1'
WHA_C -> WHDEG_ ADV='
WHDEG_ -> WHDEG'
WHDET_ -> WHDET'
WHDET_ -> WHPRO$'
WHN_C -> WHDET_ NPL='
WHN_C -> WHDET_ NPL_'
WHN_C -> WHDET_ NSG='
WHN_C -> WHDET_ PN='
WHN_C -> WHDET_ PROPL_'
WHN_C -> WHDET_ PROSG_'
WHN_C -> WHN_C' PUN=
WHN_C -> WHPRO_'
WHPRO_ -> ADV= WHPRO'
WHPRO_ -> WHPRO'
WHP_C -> PREP_' WHN_C
WHP_C -> PUNL_ WHP_C' PUNR_
WHP_C -> WHP_C' PUN=
```

Appendix D

Disambiguation

D.1 Test Sentences

These are the 100 sentences which have been used in the disambiguation experiment reported in section 6.3.4.

Rockwell said the agreement calls for it to supply 200 additional so-called shipsets for the planes.

Rockwell, based in El Segundo, Calif., is an aerospace, electronics, automotive and graphics concern.

Mr. Carlucci, 59 years old, served as defense secretary in the Reagan administration.

In January, he accepted the position of vice chairman of Carlyle Group, a merchant banking concern.

Thomas E. Meador, 42 years old, was named president and chief operating officer of Balcor Co., a Skokie, Ill., subsidiary of this New York investment banking firm.

Balcor, which has interests in real estate, said the position is newly created.

Mr. Meador had been executive vice president of Balcor.

In addition to his previous real-estate investment and asset-management duties, Mr. Meador takes responsibility for development and property management.

Those duties had been held by Van Pell, 44, who resigned as an executive vice president.

Before the loan-loss addition, it said, it had operating profit of $ 10 million for the quarter.

The move followed a round of similar increases by other lenders against Arizona real estate loans, reflecting a continuing decline in that market.

Arbitragers were n't the only big losers in the collapse of UAL Corp. stock.

When bank financing for the buy-out collapsed last week, so did UAL's stock.

By yesterday's close of trading, it was good for a paltry $ 43.5 million.

Mr. Johnson succeeds Harry W. Sherman, who resigned to pursue other interests, in both positions.

Manville is a building and forest products concern.

US Facilities Corp. said Robert J. Percival agreed to step down as vice chairman of the insurance holding company.

There was a difference of opinion as to the future direction of the company, a spokeswoman said.

Mr. Percival declined to comment.

In a statement, US Facilities said Mr. Percival's employment contract calls for him to act as a consultant to the company for two years.

Mr. Percival will be succeeded on an interim basis by George Kadonada, US Facilities chairman and president.

In the same statement, US Facilities also said it had bought back 112,000 of its common shares in a private transaction.

The buy-back represents about 3 % of the company's shares, based on the 3.7 million shares outstanding as of Sept. 30.

In national over-the-counter trading yesterday, US Facilities closed at $ 3.625, unchanged.

Merck & Co. reported a 25 % increase in earnings; Warner-Lambert Co.'s profit rose 22 % and Eli Lilly & Co.'s net income rose 24 %.

The results were in line with analysts' expectations.

Merck, Rahway, N.J., continued to lead the industry with a strong sales performance in the human and animal health-products segment.

International sales accounted for 47 % of total company sales for the nine months, compared with 50 % a year earlier.

Sales for the quarter rose to $ 1.63 billion from $ 1.47 billion.

Intense competition, however, led to unit sales declines for a group of Merck's established human and animal-health products, including Aldomet and Indocin.

In New York Stock Exchange composite trading yesterday, Merck shares closed at $ 75.25, up 50 cents.

Sales for the quarter rose to $ 1.11 billion from $ 1.03 billion.

World-wide sales of Warner-Lambert's non-prescription health-care products, such as Halls cough tablets, Rolaids antacid, and Lubriderm skin lotion, increased 3 % to $ 362 million in the third quarter; U.S. sales rose 5 %.

Confectionery products sales also had strong growth in the quarter.

Third-quarter sales of the Indianapolis, Ind., company rose 11 % to $ 1.045 billion from $ 940.6 million.

Nine-month sales grew 12 % to $ 3.39 billion from $ 3.03 billion a year earlier.

Sales of Prozac, an anti-depressant, led drug-sales increases.

Higher sales of pesticides and other plant-science products more than offset a slight decline in the sales of animal-health products to fuel the increase in world-wide agricultural product sales, Lilly said.

Advanced Cardiovascular Systems Inc. and Cardiac Pacemakers Inc. units led growth in the medical-instrument systems division.

Lilly shares closed yesterday in composite trading on the Big Board at $ 62.25, down 12.5 cents.

Analysts estimate Colgate's world-wide third-quarter sales rose about 8 % to $ 1.29 billion.

Mr. Mark attributed the earnings growth to strong sales in Latin America, Asia and Europe.

Results were also bolstered by a very meaningful increase in operating profit by Colgate's U.S. business, Mr. Mark said.

Operating profit at Colgate's U.S. household products and personal-care businesses jumped 25 % in the quarter, Mr. Mark added.

He said the improvement was a result of cost savings achieved by consolidating manufacturing operations, blending two sales organizations and focusing more carefully the company's promotional activities.

The estimated improvement in Colgate's U.S. operations took some analysts by surprise.

Colgate's household products business, which includes such brands as Fab laundry detergent and Ajax cleanser, has been a weak performer.

Analysts estimate Colgate's sales of household products in the U.S. were flat for the quarter, and they estimated operating margins at only 1 % to 3 %.

But it's not mediocre, it's a real problem.

To focus on its global consumer-products business, Colgate sold its Kendall health-care business in 1988.

H. Anthony Ittleson was elected a director of this company, which primarily has interests in radio and television stations, increasing the number of seats to five.

Osborn also operates Muzak franchises, entertainment properties and small cable-television systems.

Mr. Ittleson is executive, special projects, at CIT Group Holdings Inc., which is controlled by Manufacturers Hanover Corp.

The Boston Globe says its newly redesigned pages have a crisper look with revamped fixtures aimed at making the paper more consistent and easier to read.

By late last night, Globe Managing Editor Thomas Mulvoy, bending to the will of his troops, scrapped the new drawings.

Trouble was, nobody thought they looked right.

Lynn Staley, the Globe's assistant managing editor for design, acknowledges that the visages were on the low end of the likeness spectrum.

Is such a view justified?

About 20,000 years ago the last ice age ended.

Enormous ice sheets retreated from the face of North America, northern Europe and Asia.

This global warming must have been entirely natural – nobody would blame it on a few hundred thousand hunter-gatherers hunting mammoths and scratching around in caves.

Furthermore, no bell has yet rung to announce the end of this immense episode of natural global warming.

The Internal Revenue Service plans to restructure itself more like a private corporation.

The IRS also said that it would create the position of chief financial officer, who will be hired from within the agency.

The IRS hopes to fill the new positions soon.

Although the jobs will probably pay between $ 70,000 and $ 80,000 a year, IRS officials are confident that they can attract top-notch candidates from the private sector.

When Maj. Moises Giroldi, the leader of the abortive coup in Panama, was buried, his body bore several gunshot wounds, a cracked skull and broken legs and ribs.

They were the signature of his adversary, Panamanian leader Manuel Antonio Noriega.

The rebel officer's slow and painful death, at the headquarters of Panama's Battalion-2000 squad, was personally supervised by Gen. Noriega, says a U.S. official with access to intelligence reports.

And he is collecting the names of those who telephoned the coup-makers to congratulate them during their brief time in control of his headquarters.

In the two weeks since the rebellion, which the U.S. hesitantly backed, Mr. Noriega has been at his most brutal - and efficient - in maintaining power.

America's war on the dictator over the past two years, following his indictment on drug charges in February 1988, is the legacy of that relationship.

Before American foreign policy set out to destroy Noriega, it helped create him out of the crucible of Panama's long history of conspirators and pirates.

For most of the past 30 years, the marriage was one of convenience.

The woman had nearly died.

Mr. Noriega's tips on emerging leftists at his school were deemed more important to U.S. interests.

The 55-year-old Mr. Noriega is n't as smooth as the shah of Iran, as well-born as Nicaragua's Anastasio Somoza, as imperial as Ferdinand Marcos of the Philippines or as bloody as Haiti's Baby Doc Duvalier.

Yet he has proved more resilient than any of them.

In keeping with America's long history of propping up Mr. Noriega, recent U.S. actions have extended rather than shortened his survival.

If the U.S. had sat back and done nothing, he might not have made it through 1988, Mr. Moss contends.

One Colombian drug boss, upon hearing in 1987 that Gen. Noriega was negotiating with the U.S. to abandon his command for a comfortable exile, sent him a hand-sized mahogany coffin engraved with his name.

He is cornered, says the Rev. Fernando Guardia, who has led Catholic Church opposition against Noriega.

It is easy to fight when you do n't have any other option.

Mr. Noriega often tells friends that patience is the best weapon against the gringos, who have a short attention span and little stomach for lasting confrontation.

The U.S. discovered the young Tony Noriega in late 1959, when he was in his second year at the Chorrillos Military Academy in Lima, according to former U.S. intelligence officials.

Tony was four years older than most of his fellow cadets, and gained admission to the academy because his brother had falsified his birth certificate.

He had an elegant uniform with gold buttons in a country where there was a cult of militarism, where officers were the elite with special privileges, recalls Darien Ayala, a fellow student in Peru and a lifelong friend.

Mr. Noriega's relationship to American intelligence agencies became contractual in either 1966 or 1967, intelligence officials say.

United Fruit was one of the two largest contributors to Panama's national income.

Mr. Noriega's initial retainer was only $ 50 to $ 100 a month, plus occasional gifts of liquor or groceries from the American PX, a former intelligence official says.

It was modest pay by American standards, but a healthy boost to his small military salary, which fellow officers remember as having been $ 300 to $ 400 monthly.

He did it very well, recalls Boris Martinez, a former Panamanian colonel who managed Mr. Noriega and his operation.

He started building the files that helped him gain power.

A National Guard job assumed by Capt. Noriega in 1964 – as chief of the transit police in David City, capital of the Chiriqui Province – was tailor-made for an aspiring super-spy.

By pressuring taxi and bus drivers who needed licenses, he gained a ready cache of information.

He knew which local luminaries had been caught driving drunk, which had been found with their mistresses.

Mr. Noriega had learned that a local union leader was sleeping with the wife of his deputy.

So he splashed the information on handbills that he distributed throughout the banana-exporting city of Puerto Armuelles, which was ruled by United Fruit Co.

It was like a play on Broadway, recalls Mr. Martinez.

Noriega was an expert at bribing and blackmailing people.

D.2 Results

The following table shows the results of the disambiguation experiment. The contents of the columns are as follows:

No.: number of the sentence
MaxScore: highest score for this sentence
Score: score of the correct analysis
Analyses: number of analyses for this sentence
AsGood: number of analyses with a score as high as or higher than the correct one
%AsGood: the same value expressed in percent of all analyses
Better: number of analyses with a score higher than the correct one
%Better: the same value expressed in percent of all analyses

No.	MaxScore	Score	Analyses	AsGood	%AsGood	Better	%Better
1	44635	42015	37	24	64.86	23	62.16
2	860	860	675	404	59.85	184	27.26
3	38972	36086	40	24	60.00	16	40.00
4	9833	9739	129	8	6.20	4	3.10
5	114340	111295	17997650	27540	0.15	27416	0.15
6	13079	12556	50	22	44.00	12	24.00
7	4126	4126	3	1	33.33	0	0.00
8	19318	13440	11013	951	8.64	947	8.60
9	7795	3832	411	76	18.49	68	16.55
10	27449	27449	356	2	0.56	0	0.00
11	24284	8907	3700	396	10.70	387	10.46
12	39188	2530	27	6	22.22	4	14.81
13	9705	2470	198	124	62.63	122	61.62
14	96717	2343	193	45	23.32	44	22.80
15	79086	67506	18	8	44.44	5	27.78

File	MaxScore	Score	Analyses	AsGood	%AsGood	Better	%Better
16	465	435	13	2	15.38	1	7.69
17	64420	15229	912	178	19.52	166	18.20
18	13560	11121	624	180	28.85	168	26.92
19	141	141	2	2	100.00	0	0.00
20	159921	62645	57372	5008	8.73	5004	8.72
21	34415	30394	1010	576	57.03	562	55.64
22	9498	5676	11327	4271	37.71	4267	37.67
23	25419	24714	142	39	27.46	34	23.94
24	1208	897	460	340	73.91	268	58.26
25	83457	83225	954	22	2.31	21	2.20
26	37814	1265	7	6	85.71	5	71.43
27	1958	1525	3179	500	15.73	485	15.26
28	55768	48377	42854	1800	4.20	1790	4.18
29	5940	5940	32	2	6.25	0	0.00
30	43242	42846	121358	119	0.10	114	0.09
31	82513	1975	10136	5228	51.58	5204	51.34
32	5940	5940	32	2	6.25	0	0.00
33	12546	7190	298414	250316	83.88	249958	83.76
34	4094	3828	8	8	100.00	4	50.00
35	10777	6362	48	28	58.33	26	54.17
36	6422	726	81	81	100.00	74	91.36
37	1661	1599	19	5	26.32	1	5.26
38	87267	70250	137996	16870	12.22	16862	12.22
39	2901	2901	1204	36	2.99	0	0.00
40	26260	2497	1500	164	10.93	160	10.67
41	26853	24222	138	41	29.71	40	28.99
42	9747	9741	39	6	15.38	3	7.69
43	91091	61268	79622	17956	22.55	17954	22.55
44	51896	5374	22460	6482	28.86	6452	28.73
45	50329	47339	536122	66528	12.41	66498	12.40
46	37967	4303	70	28	40.00	24	34.29
47	1137	965	1262	952	75.44	940	74.48
48	152439	15841	37356	21236	56.85	21234	56.84
49	1646	1646	6	6	100.00	0	0.00
50	113515	113347	576	86	14.93	78	13.54
51	77953	75778	11910	924	7.76	916	7.69
52	1225	1225	6	3	50.00	0	0.00
53	6572	1079	3208	864	26.93	848	26.43
54	47153	2380	5602	1142	20.39	1140	20.35
55	10522	5957	576	107	18.58	104	18.06
56	655	655	8	1	12.50	0	0.00
57	104923	95802	971	302	31.10	298	30.69
58	169	158	2	2	100.00	1	50.00
59	50687	50687	19	1	5.26	0	0.00
60	3533	3489	46	15	32.61	14	30.43

File	MaxScore	Score	Analyses	AsGood	%AsGood	Better	%Better
61	7172	7172	1939	2	0.10	0	0.00
62	72423	40585	11	5	45.45	4	36.36
63	2094	2094	54	2	3.70	0	0.00
64	5844	5755	10	3	30.00	1	10.00
65	492	492	8	1	12.50	0	0.00
66	12835	12092	1026	828	80.70	812	79.14
67	41714	5259	3789	862	22.75	854	22.54
68	124033	124027	39	4	10.26	2	5.13
69	68869	31726	492472	307182	62.38	307166	62.37
70	42105	7129	900	108	12.00	82	9.11
71	185068	179057	1580	250	15.82	246	15.57
72	120484	64602	65647	6875	10.47	6869	10.46
73	19686	14412	25504	5046	19.79	5034	19.74
74	151520	56311	100	25	25.00	24	24.00
75	626	626	3	1	33.33	0	0.00
76	24440	7731	1166	1004	86.11	992	85.08
77	66702	62716	22721	19865	87.43	19491	85.78
78	1045	1045	2	1	50.00	0	0.00
79	46614	9152	23513	9128	38.82	9125	38.81
80	12596	12476	1352	17	1.26	13	0.96
81	238430	107755	4445800	239774	5.39	239758	5.39
82	1287	1287	10	2	20.00	0	0.00
83	38723	35974	40	28	70.00	24	60.00
84	100576	5481	4216	288	6.83	282	6.69
85	261525	179775	395281	194604	49.23	194598	49.23
86	36189	36189	100	3	3.00	0	0.00
87	53369	14769	5751994	1243308	21.62	1243270	21.61
88	249781	211575	1528	566	37.04	564	36.91
89	15445	14166	54	10	18.52	9	16.67
90	6086	2920	559180	138883	24.84	138849	24.83
91	8271	5917	21444	7984	37.23	7956	37.10
92	70550	70304	168	4	2.38	2	1.19
93	1037	1037	7	1	14.29	0	0.00
94	193909	115087	3300316	1797396	54.46	1797348	54.46
95	9018	8982	23	5	21.74	4	17.39
96	2894	1652	2913	1529	52.49	1521	52.21
97	4247	4067	90	8	8.89	7	7.78
98	11908	2828	1595	1199	75.17	1180	73.98
99	13678	1204	32	21	65.62	20	62.50
100	6971	548	14	14	100.00	13	92.86