A MODEL OF GRAMMAR BASED ON PRINCIPLES OF GOVERNMENT AND BINDING

by

RANDALL MARTIN SHARP

B.Sc., Simon Fraser University, 1977

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Computer Science

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October, 1985

© Randall Martin Sharp, 1985
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at
The University of British Columbia, I agree that the Library shall make it freely available for
reference and study. I further agree that permission for extensive copying of this thesis for
scholarly purposes may be granted by the Head of my Department or by his or her
representatives. It is understood that copying or publication of this thesis for financial gain
shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: October 1985
Abstract

This thesis describes an implementation of a model of natural language grammar based on current theories of transformational grammar, collectively referred to as Government and Binding (GB) theory. A description is presented of the principles of GB, including X-bar syntax and the theories of Case, Theta, Binding, Bounding, and Government. The principles, in effect, constitute an embodiment of "universal grammar" (UG), i.e. the abstract characterization of the innately endowed human language faculty. Associated with the principles is a set of parameters that alter the effect of the principles. The "core grammar" of a specific language is an instantiation of UG with the parameters set in a particular way. To demonstrate the cross-linguistic nature of the theory, a subset of the "core grammars" of Spanish and English is implemented, including their parametric values and certain language-specific transformations required to characterize grammatical sentences. Sentences in one language are read in and converted through a series of reverse transformations to a base representation in the target language. To this representation, transformations are applied that produce a set of output sentences. The well-formedness of these sentences is verified by the general principles of UG as controlled by the parameters. Any that fail to meet the conditions are rejected so that only grammatical sentences are displayed. The model is written in the Prolog programming language.
# Table of Contents

1. INTRODUCTION .................................................................................................................. 1

2. LINGUISTIC FRAMEWORK .................................................................................................. 4
    2.1 Rule Systems .................................................................................................................. 5

2.2 Systems of Principles ....................................................................................................... 7
    2.2.1 X-Bar Theory ........................................................................................................... 9
    2.2.2 Theta Theory ........................................................................................................... 14
    2.2.3 Case Theory ............................................................................................................. 15
    2.2.4 Government Theory ............................................................................................... 17
    2.2.5 Binding Theory ....................................................................................................... 20
    2.2.6 Bounding Theory .................................................................................................... 21
    2.2.7 Control Theory ....................................................................................................... 23

3. REPRESENTATIONS ............................................................................................................. 24
    3.1 Lexicon .......................................................................................................................... 24
        3.1.1 Dictionary ............................................................................................................. 24
        3.1.2 Suffix Table ......................................................................................................... 27
        3.1.3 Contraction Table .............................................................................................. 28
        3.1.4 Collocation Table .............................................................................................. 28
    3.2 Phrase Structure ............................................................................................................ 29
        3.2.1 COMP .................................................................................................................. 29
        3.2.2 INFL .................................................................................................................... 30
        3.2.3 VP ....................................................................................................................... 30
        3.2.4 NP, PP, AP ........................................................................................................... 31
        3.2.5 S-Adjunction ....................................................................................................... 31
        3.2.6 Example .............................................................................................................. 32
    3.3 Empty Categories .......................................................................................................... 32

3.4 Grammar .......................................................................................................................... 33
    3.4.1 Phrase Structure Rules ........................................................................................... 34
Acknowledgements

I wish to acknowledge the tremendous support and guidance I received from my supervisor, Dr. Michael Rochemont, who first introduced me to syntax and linguistic endeavors. Words cannot express the measure of my gratitude.

I would also like to thank Dr. Harvey Abramson for technical assistance, Dr. Karl Kobbervig for assistance with Spanish, the graduate students in the Department of Linguistics for their encouragement and spirit, and various friends who have helped in their own special way, including Donald Acton (a wizard), Gladys Wong (bless her heart), María Perez, and Michael Tropp.
Chapter 1

INTRODUCTION

One of the aims of linguistic inquiry is the analysis of human languages and the underlying systems of grammar. An increasingly prevalent trend among modern linguists is the construction of a theory of grammar that is appropriate for the description of all human languages, rather than merely descriptive of any one language or family of languages. A significant factor in the formulation of a general theory of grammar is the degree to which it provides an explanation for the empirical facts of the language. The explanatory power of a theory is important to those researchers who, for example, are curious about how language is acquired by children.

In the development of a theory of natural grammar, there is a clear preference away from rule-oriented descriptions of grammar to principle-based descriptions. This has a number of consequences. Firstly, it simplifies the descriptions of natural grammars, since the interaction of only a few principles can account for a wide range of data that would otherwise require numerous distinct rules. Secondly, it contributes very strongly to the concept of a genetically realized "language faculty", where the principles of a universal grammar (UG) are somehow inherent in the biological endowment of human beings and serve to account for the rapidity with which normal children acquire language. Thirdly, it reduces the variety of possible grammars by restricting the applications of rules.

According to some linguists, transformational generative grammar still holds the best promise of a concise, encompassing principle-based theory of natural grammar. Transformational theory has undergone considerable change since the publication of Chomsky’s Syntactic Structures (1957) and the subsequent enrichment generally referred to as the "Extended Standard Theory." In particular, the emphasis on the principles of UG rather than rules has effectively reduced the set of transformations to the single rule Move-α, where α arbitrarily
selects across the set of grammatical categories, and ill-formed results are filtered out by virtue of the principles. The principles act as conditions on the well-formedness of linguistic representations.

Cross-linguistic description can now be attempted by specifying parameters of variation to the principles that would account for language differences. The grammar of any one particular language, referred to as the "core grammar", would be represented solely by the settings of the parameters. The core grammars in conjunction with UG, modified by idiosyncracies of markedness, determine the set of actual grammars. This holds conceptually at the individual level as well as the language family level.

This discussion of grammars is restricted to issues relating to syntax, although clearly other aspects, such as semantics, morphology, pragmatics, phonology, etc., combine to yield the empirical facts of human language. These other aspects may also be expressed in terms of basic principles and rule systems, not necessarily the same as those for syntax. The principle of the autonomy of syntax (Chomsky 1977a) maintains a conceptual separation, though interrelatedness, among these components, and allows for independent investigation of each area.

This thesis looks at a particular subset of some of the principles and rules involved in a theory of transformational grammar drawing primarily on the theory of government and binding (GB) as developed by Chomsky (1981) and several other researchers. A model is constructed that attempts to simulate the effects of UG, modified by parametric variation and combined with language-specific rules, in the context of parsing, generation, and translation of natural language. The primary goal of the research is to investigate the feasibility of representing the theory computationally, thereby formalizing it in some, albeit prototypical, sense.

Because of their inherent similarities, the languages of Spanish and English were chosen to demonstrate the model. This minimizes the variability one might expect in two distinct language grammars, and makes the computational realization more tractable. If the endeavor succeeds, under some reasonable criteria for success, then several possibilities are opened up.
The formalization of UG can be improved, extending the range of grammatical constructions. The core grammars can be improved to further develop the coverage of Spanish and English sentences. Other core grammars could be added, which would have the dual effect of verifying the principles and their realization, and of enhancing our understanding of the theory of UG so that fundamental refinements to it may be made. Finally, in the area of applications research, the model may contribute to the eventual realization of a language processing device, not only for translation, but also for the intelligent recognition and production of written and spoken communication.

The computational system described here makes use of Prolog, a logic programming language specifically designed for the manipulation of non-numeric symbolic data. Numerous natural language applications have been developed in Prolog (Dahl 1983, Schwind 1984), though none are known to exist that are based on the concepts underlying the current work.

The thesis is organized as follows: Chapter 2 presents the linguistic framework underlying the theory of UG. Chapter 3 covers the representations in Prolog of the linguistic structures. Chapter 4 describes in detail the various transformations being modeled. Chapter 5 concerns the conditions that determine well-formed structures and how they are modeled. Chapter 6 presents a breakdown of the overall system, including the stages of parsing, translation, and generation. Chapter 7 evaluates the work and provides a discussion of some of the issues that have arisen in the construction of the model. Chapter 8 concludes the study with a summary of results and areas for future research. Appendix A contains sample dialogues illustrating the performance of the system, and the remaining appendices contain the entire Prolog program.
The description of natural grammars can be viewed from two perspectives. One is the rule-based approach. We use rules in learning, for example, that past tense is normally formed by adding \(-ed\) to words. We hear children follow the rules, and learn the exceptions, often by over-application of the rule and subsequent retreat. Syntactic rules are learned in forming questions, and we follow semantic rules when we learn meanings. An on-going activity among many researchers in linguistics is the description of these rules at the various levels.

A second approach is principle-based, where fundamental principles of grammar account for the well-formedness of linguistic constructions. They hold of the applications of rules and have the effect of constraining the set of possible rule derivations. Principles needn't be learned; they are somehow inherent in the mechanism of language.

Since the mid-50s, following Chomsky's *Syntactic Structures* (1957), transformational generative grammar has been rigorously studied from a formal perspective by many linguists (e.g. Peters and Ritchie (1973), Lasnik and Kupin (1977), Wexler and Culicover (1980)). The evolution in this enterprise has been a shift from rule descriptions to descriptions of general principles, motivated in large part by the desire for a theory of grammar that adequately explains, not just describes, linguistic phenomena. Out of this endeavor arises a theory of universal grammar (UG), i.e. that grammar that represents the linguistic knowledge common to all human languages. This in turn leads to a theory of language acquisition and helps explain why language is learned as fast as it is.

As a consequence, the rule component of grammar diminishes, reducing the number of rules that must be explicitly learned. In particular, all the rules that seem to involve movement, such as Raising-to-Subject, Extraposition, WH-Fronting, etc., reduce to a single rule, Move-\(a\), i.e. "move any category anywhere." Over-applications of this rule are ruled out.
by the principles of grammar.

Chomsky (1981) provides a breakdown of the basic rule systems and systems of principles. The components of the rule systems are the following:

(1) (i) lexicon
(ii) syntax
   (a) categorial component
   (b) transformational component
(iii) PF-component
(iv) LF-component

The systems of principles fall in the domain of the following areas:

(2) (i) X-Bar Theory
(ii) Theta Theory
(iii) Case Theory
(iv) Binding Theory
(v) Bounding Theory
(vi) Government Theory
(vii) Control Theory

We look in some detail below at each of the different components of these systems.

2.1 Rule Systems

The lexicon contains the vocabulary of the language, arranged as a list of lexical entries, with each entry containing syntactic, semantic, phonological, and morphological information.

In addition, the lexicon contains rules for creating new lexical entries, as in Word-Formation Rules (e.g. creating adverbs from adjectives by adding -ly) and Restructuring Rules, as in idioms (e.g. take-advantage-of), the restructuring verbs in Romance (Rizzi 1982, Rouvert and Vergnaud 1980, Burzio 1981), and the amalgamation of prepositions with verbs (e.g. eat-up your granola). The morphological rules for inflections of verbs, nouns and adjectives are also included in the lexicon.

The syntax is responsible for generating structural representations of sentences. It consists of two components: the categorial component and the transformational component.

The categorial component expresses phrase structure. It relies on the principles of X-bar theory to organize phrasal categories and to reduce, if not eliminate, distinct rules of
phrase structure. The phrasal categories themselves are projected directly from the lexicon as properties of lexical entries. Specifiers and modifiers position themselves within the structures according to categorial affinities, represented as features, with ordering being largely idiosyncratic to different languages.

The interaction of the lexicon and categorial component, together referred to as the base component, generate base structures ("D-structures" or "deep structures"), i.e. representations in which the semantic participants are expressed directly in syntactic structures.

The transformational component transforms the base structure into surface-like structures, or S-structures, which obey the principles of well-formed surface representation. Some transformations are language-specific, such as the English rule of Do Support (see 4.4.3). Others are applied generally, such as Move-α and the Move-Affix rule for "attaching" tense information to a verb.

The S-structure is interpreted phonologically by the PF-component and logically by the LF-component, giving rise to phonological form (PF) and logical form (LF), respectively. Each have their own rules, such as the PF "wanna-construction" rule (Chomsky and Lasnik 1978) which converts (3a) to (3b):

(3) a. I want to win.
    b. I wanna win.

and the LF-rule of quantifier-raising (Chomsky 1981) which converts (4a) to (4b):

(4) a. She met a friend of Judy's.
    b. for some x, x a person who likes Judy, she met x

While the rules and principles that make up PF and LF are interesting areas of investigation and share some overlap with those of the syntax, it is not the intention to treat these in this study.

The relationship between these levels of representation is diagrammed below:

---

1 See Drescher's (1981, 1984) work, which emphasizes an approach to explanation in phonology based on general principles.
D-structure is base-generated; S-structure is related to D-structure by the rule Move-a, and S-structure maps to representations of sound (PF) and meaning (LF) by other interpretive subsystems, possibly involving rules like Move-a.

2.2 Systems of Principles

The principles of grammar determine the well-formedness of the various representations. They act as filters to overgeneration by the rule systems. They interact in interesting ways to account for all and only grammatically correct constructions.

The theories listed in (2) represent independent areas that together form a general theory of UG. X-bar theory develops the representation of phrase structures. Theta theory refers to the thematic or semantic relations that hold between lexical items. Case theory concerns the assignment of Case (e.g. nominative, accusative, etc.) to noun phrases. Binding theory specifies the relations that exist between noun phrases. Bounding theory places restrictions on the application of Move-a. Government theory refers to dominance relations that exist between categories. Control theory identifies the reference to PRO, a pronominal element with no phonetic content.

Associated with each theory are one or more principles that hold of grammatical constructions. For example, a principle of Case Theory, the Case Filter, states that all phonetically-realized noun phrases must have Case. This would appear to be a fundamental principle of grammar irrespective of any particular language. The principles may be subject to some degree of variation, and it is this variation that accounts for language differences. With
Case Theory, English seems to have an adjacency requirement, such that nominal elements with objective Case must be adjacent to their Case assigner, as illustrated by the following two sentences:

    b. *John put on the table the book.

This requirement of adjacency may not hold in other languages.

It is postulated, then, that UG is described in terms of the fundamental principles of grammar, and that individual grammars ("core grammars") are described by the particular settings of the available parameters of variation. By exposure to linguistic data from the environment, a child learns the values of the parameters, and a core grammar becomes established. Exceptions to the consequences of core grammar do arise, and these are covered under a theory of markedness.

Each of these theories are described more completely below. However, much investigation is currently going on in all areas, and new analyses are continually being presented. Consequently the theories are in a constant state of flux, and any implementation is bound to reflect out-of-date concepts. By virtue of the modular nature of the principles, it is possible to make some changes in one area without necessarily having to alter substantially other areas, analogous to the situation in computer programming where the concepts of structured programming and modular design are now considered standard practice. By the same token, a small change in one area could have proliferating effects throughout the grammatical system, as Chomsky observes.

---

2 As is customary, ungrammatical sentences are prefixed with "*".
3 Sentence (b) may be derived by a rule of Heavy NP Shift, where the base-generated NP direct object is moved to clause-final position, usually with a comma inserted to indicate an intonation break, as in:

(i) John put $t$ on the table, the book.

The various idiosyncratic properties of sentences like (i) can only be derived by assuming that such cases are derived, not canonical as with (5a).
2.2.1 X-Bar Theory

X-Bar theory arises from the specification of feature types shared naturally among lexical items. Different hypotheses have been proposed as to the most appropriate classification of features; Chomsky’s (1972) classification is that adopted here, although Jackendoff (1977) presents a detailed alternative.

Lexical items can be distinguished as to their substantive qualities, denoted by the feature \([\pm N]\) for nominal or non-nominal, and their predicative qualities, \([\pm V]\), for verbal or nonverbal. The four possible combinations are associated with the four major lexical categories: noun(N)\(=\)\([+N,-V]\), verb(V)\(=\)[-N,+V], adjective(A)\(=\)[+N,+V], and preposition(P)\(=\)[-N,-V]. Thus, generalizations may be made in terms of feature content, such as one that refers to only [-N] categories (i.e. the class of English Case assigners).

The major generalization is that phrase structures are projections of a lexical head X with features \([\pm N,\pm V]\), such that they follow the context-free rule:

\[
X^1+1 \rightarrow C_1 \ldots C_j \ X^1 \ C_k \ldots C_n
\]

where X is one of N, V, A, or P, and the \(C_i\) for \(i=1\) to \(n\), are possible adjuncts. \(X^0\) refers to the lexical head. \(X^1\) dominates \(X^0\), \(X^2\) dominates \(X^1\), etc. \(X^{max}\) is considered to be the maximal projection of X, for some value of \(max\), typically 2. Maximal projections will often be denoted by their more familiar form NP, VP, etc.

The first level of the structure, \(X^1\) (more commonly denoted \(\bar{X}\) in the literature), dominates the lexical head \(X^0\) (or just X) and its complement, if any. This is represented by the phrase structure rule:

\[
X^1 \rightarrow X \text{ Complement}
\]

The complement to the head is determined by the semantic requirements of the head. For example, the verb \textit{put} takes a complement of two arguments: a noun phrase (object) and a prepositional phrase (location). Both are required to satisfy the semantic interpretability of the verb, as illustrated by the following sentences:
b. *John put the book.
c. *John put on the table.

The verb *put, then, is subcategorized as taking an NP and a PP. This subcategorization information, or frame, is contained in its lexical entry.

Other categories besides V can also subcategorize arguments:

(9) Adjectives:
   a. anxious that you might forget (= S)
   b. satisfied with his job (= PP)

(10) Prepositions:
   a. behind every man (= NP)
   b. towards the sun (= NP)
   c. *away the house

(11) Nouns:
   a. pride in his work (= PP)
   b. belief that ghosts are real (= S)
   c. purchase of a book (= NP)

(10c) illustrates that not all prepositions are transitive; an NP following such a preposition will not receive Case and is therefore excluded by the Case Filter. In (11b) and (11c), the nouns are related to transitive verbs (believe, purchase) which subcategorize equivalent phrases. The of is inserted transformationally in (11c) (through an Of-Insertion rule (Chomsky 1981) and analogous rules in other languages) in order to satisfy the Case Filter.

At the next level, X² (= X), are the specifiers to the X¹ constituent. For nouns, these include determiners (e.g. the articles the, a), quantifier phrases (e.g. some, a few), and possessives (e.g. Alan's, my sister-in-law's). For verbs, it includes the auxiliaries (have, be). For adjectives and prepositional phrases, it includes adverbial phrases (e.g. very satisfied, right behind). This level is represented by the rule:

(12) X² → Specifier X¹

In general, specifiers may be optional, depending on semantic and pragmatic considerations.

Modifying phrases may also be included in phrasal structures, and these are typically assumed to adjoin at the X¹ level, at the same time introducing recursivity. Thus the phrase a bright young student of physics with long hair eager to please whomever would be represented by the following phrase structure:
The additional phrase structure rules for recursive expressions are:

(13) a. \(X^1 \rightarrow Y^2 X^1\)
    b. \(X^1 \rightarrow X^1 Y^2\)

where \(Y^2\) represents adjunct(s) of any phrasal category and, as shown, may occur on either side of category \(X^1\).

The basic set of phrase structure rules, then, are summarized below:

(14) a. \(X^2 \rightarrow \text{Specifier} X^1\)
    b. \(X^1 \rightarrow \text{Adjunct} X^1\)
    c. \(X^1 \rightarrow X^1 \text{ (Adjunct)}\)
    d. \(X^1 \rightarrow X \text{ (Complement)}\)

A characteristic of English grammar is that the specifiers of all the lexical categories always occur before the \(X^1\) constituent, and complements always follow the head. This isn't so for all languages. In German, for example, verbal complements generally precede the verb. In the Dravidian language group, verbal specifiers follow the verb, while nominal specifiers precede the noun, as in English; in Kikuyu, a Bantu language of East Africa, nominal
specifiers follow the noun (Lightfoot 1982). Therefore, it would appear that the order of constituents is subject to a degree of parametric variation. But the basic constituent structure follows the pattern given in (14) above. (Consideration of phrase structure syntax for "free word order" languages like Japanese, and how the X-bar analysis might differ, is given in (Chomsky 1981).)

In addition to the X-bar system for lexical categories, a rule is also required for the derivation of the clause. This has traditionally taken the form:

(15) \[ S \rightarrow NP \text{ AUX } VP \]

where NP is the subject, VP the predicate, and AUX contains tense and modality information. Chomsky (1981) gives the revised rule:

(16) \[ S \rightarrow NP \text{ INFL } VP \]

where INFL, suggestive of "inflection", contains a tense feature \[\text{[±TNS]}\] (Chomsky 1980) and, if \[\text{[+TNS]}\], agreement features, abbreviated AGR, consisting of person, number and gender.

That a clause may be finite (\[\text{[+TNS]}\]) or infinitival (\[\text{[-TNS]}\]) is illustrated by the sentences of (17) with the underlying structure in (18):

(17) a. I believe (that) Beatrice is my friend.
    b. I believe Beatrice to be my friend.

(18) \[ \text{[ I believe [ Beatrice [±TNS] be my friend ]]} \]

In (18), the verb believe is subcategorized simply as taking a clause, which may be tensed (17a) or untensed (17b). An alternative analysis would be to state that believe takes either a subordinate clause or a "to-VP" complement. However, this misses the generalization that these two statements appear to express the same proposition, varying only as to tense of the embedded clause. Many verbs are of this type, so that an unnecessarily large lexicon would be required to express the two possible complements. Furthermore, the principles of GB ride heavily on the assumption that verbs taking clausal complements select either a tensed or non-tensed form, subject to other considerations, and that the subject of the non-tensed form normally must be non-lexical PRO, as in (19a) with internal structure (b), unless lexical NPs are allowed, as in (17), by exceptional mechanisms, such as Exceptional Case-Marking (cf. 2.2.3
below).

(19)  
a. They prepared to flee.
b. [ They prepared [ PRO to flee ]]

In addition, it is assumed that another rule exists for introducing clauses with a complementizer, COMP (Chomsky 1977b), such as *that* for tensed clauses and *for* for infinitivals:

(20)  
a. I said (that) we were leaving now.
b. He would prefer (for) you to read it out loud.

This rule is expressed as:

(21)  

\[ \overline{S} \rightarrow \text{COMP } S \]

suggesting that \( S \) is the head of \( \overline{S} \), although this will be altered below. It is reasonable to assume, then, that lexical items that subcategorize for a clause actually specify \( \overline{S} \) as their argument, with a rule introduced to delete the complementizer for those verbs which seem to optionally take it (cf. 4.6).

Rule (16) for \( S \) appears to deviate from the schema presented under the \( X \)-bar system (14). However, it has been suggested that INFL is the head of \( S \) (Chomsky 1981), subcategorizing the predicate phrase, and that the subject NP is the specifier. The modified rule for the clause \( S \) then becomes:

(22)  
a. \[ \text{INFL}^2 \rightarrow \text{N}^2 \text{ INFL}^1 \]
b. \[ \text{INFL}^1 \rightarrow \text{INFL } \text{V}^2 \]

Similarly, an argument is presented (Stowell 1981, Chomsky 1985) that COMP is the head of \( \overline{S} \). We may then consider the specifier of COMP to be a *wh-*phrase and INFL the complement, thus converting rule (21) to rule (23):

(23)  
a. \[ \text{COMP}^1 \rightarrow \text{wh-phrase } \text{COMP}^1 \]
b. \[ \text{COMP}^1 \rightarrow \text{COMP } \text{INFL}^2 \]

Under these new representations of the rules for \( S \) and \( \overline{S} \), the \( X \)-bar system in (14) is now adequate to represent *any* phrasal category \( X \), where \( X \) is one of the set \{ N, A, V, P, COMP, INFL \}.
2.2.2 Theta Theory

Theta Theory concerns the thematic properties of lexical items. It entails such notions as "agent-of-action", "goal", "instrument", etc., as developed in the theories of Gruber (1965), Fillmore (1968), Jackendoff (1972), and others. Lexical items assign thematic roles, or \( \theta \)-roles, to phrasal constituents as part of their semantic description. For example, the verb *put* assigns the \( \theta \)-role "patient" to its direct object NP and "location" to the locative PP. In addition, it assigns the \( \theta \)-role of "agent" to an NP which typically assumes the grammatical function of subject. A fundamental principle of grammar, the Projection Principle, expresses the notion that the \( \theta \)-marking properties of lexical items must be represented categorially at each syntactic level, i.e. at D-structure, S-structure and LF.

\( \theta \)-role assignment and subcategorization are evidently closely linked concepts, as it appears that any phrase which is subcategorized is also assigned a \( \theta \)-role. The reverse is not true. When not assigned a \( \theta \)-role, the subject must still appear (in English) as a pleonastic element, void of semantic content (e.g. *it*, in *It seems that Donald is very happy*). However, subcategorized objects are never pleonastic; they always have a \( \theta \)-role. The definition of the term "argument", introduced in the previous section, is extended to include not only the subcategorized phrases, referred to as "internal arguments," but also any phrase which receives a \( \theta \)-role but is not subcategorized for, referred to as "external arguments" (Williams 1981).

A principle of UG formalizes this relationship, known as the \( \theta \)-Criterion. Stated simply, it is:

\[
\text{(24) } \quad \text{\( \theta \)-Criterion: } \\
\text{Each argument bears one and only one \( \theta \)-role, and each \( \theta \)-role is assigned to one and only one argument.}
\]

The Projection Principle and \( \theta \)-Criterion together state the primary conditions on phrase structure, configured in terms of X-bar syntax. Specific phrase structure rules are eliminated, since lexical items directly determine, through subcategorization and \( \theta \)-marking features, the required constituents. Order of constituents may derive from X-bar parameters and possibly by an adjacency requirement for Case assignment (cf. next section).
Additionally, clauses must have subjects. The subject may be the phonetically null PRO, such as the embedded subject in (19), or if the subject is not θ-marked it must be pleonastic it or there (il in French), as in:

\[(25)\]

a. It is raining.
b. It is likely that no one will leave.
c. There seem to be problems with this analysis.

Spanish and other languages do not show pleonastic subjects but evidence suggests that the subject position is nevertheless present. The Projection Principle and the requirement that clauses have subjects are together referred to as the **Extended Projection Principle**.

Since thematic relations express semantic properties, they are not described in any detail here, as we are primarily concerned with issues of syntax. The relevant issue, though, is the one-to-one correspondence between the θ-roles assigned by a lexical item and the syntactic structures to which they are assigned. Since subcategorization entails θ-marking by the θ-Criterion, it is sufficient to note the subcategorized arguments; the assignment of θ-roles to the arguments follows implicitly. θ-role assignment to the subject must be explicit, however, since the subject position is present (by the Extended Projection Principle) whether or not the position is assigned a θ-role by the predicate. If by default the predicate assigns a θ-role to the subject, then only those that do not assign a θ-role need an explicit feature to that effect, as for the weather verbs (rains, snows, etc., although the subjects of such verbs have "quasi-argument" status (Chomsky 1981)), verbs like seem, appear, and predicate adjectives like likely, probable, apparent, etc. In this system, the feature [−THETA] is associated with predicates that do not θ-mark the subject. The non-θ-marked subjects in English are replaced by pleonastic elements; in Spanish, they remain unfilled.

**2.2.3 Case Theory**

Case Theory refers to the assignment of abstract Case, such as nominative, accusative, etc., to nouns. It is not to be confused with Fillmore’s cases (Fillmore 1968), which are related to thematic roles, covered in the previous section. Morphologically, the English language shows
three Case inflections, exemplified by the pronouns *I, me, my,* for nominative, objective (=accusative), and genitive Case, respectively. Non-pronominals are not distinguished as to Case, except for the affix *'s* on genitives, as in *Noam's book.* Other languages have richer Case inflections: Latin has six Cases, Finnish has fifteen, etc.

Case is assigned to a noun phrase by virtue of the structural position in which it occurs. Thus a transitive verb "governs" its direct object (in the traditional sense of government (Lyons 1968), to be more formalized in Section 2.2.4), assigning it objective Case. Similarly a (transitive) preposition assigns Case to the object it governs. Nominative Case is assigned to the subject of tensed clauses, i.e. in configurations where the subject NP is governed by [+TNS];4 the subject of infinitives is generally not Case–marked, as shown in (26), though the presence of the prepositional *for* complementizer will Case–mark the subject, as in (20b), as will other exceptional means noted below. The influence of [±TNS] on the subject position is illustrated by the following example:

(26) a. It seems [*Heather [+TNS] has left *]
b. *It seems [*Heather [−TNS] to have left *]

In (a), *Heather* is assigned nominative Case by [+TNS] in the embedded clause, whereas in (b) the embedded clause is infinitival so that *Heather* is not Case–marked, and the sentence is ungrammatical.

To account for this and related phenomena, the Case Filter is postulated as a fundamental principle of grammar:

(27) Case Filter:  
•NP if NP is lexical and has no Case

Thus if a lexical NP is in any structural configuration in which Case is not assigned, such as the subject of an infinitival clause, it will be ungrammatical. The transformation Move–α, described later, must be invoked to move the NP to a location where Case is assigned.

---

4 Chomsky (1981) treats AGR, the agreement feature complex, as the nominative Case–assigning agent. Since AGR accompanies [+TNS] but not [−TNS], the choice may not be relevant, at least at the level of detail studied here.
Certain cases arise where an infinitival clause does contain a subject. In these instances the matrix verb, if transitive, may assign Case across the S boundary to the embedded subject. The S node is effectively deleted, allowing the embedded subject to be governed directly by the matrix verb, thus assigning it Case. This is referred to as Exceptional Case-Marking (ECM), an idiosyncratic feature of English grammar. It is a property of certain verbs that take either NP arguments (i.e. are transitive) or clausal arguments. An example is the verb believe:

(28) a. I believe him.
   b. I believe him to be telling the truth.
   c. I believe (that) he is telling the truth.

In (a), an NP direct object is selected and assigned objective Case. In (b), a clausal complement is selected, and Case is assigned by ECM to the subject of the infinitival clause. In (c), nominative Case is assigned in the normal way to the embedded subject since the clause is tensed.

For languages that do not have the ECM property, such as Spanish, only (28a,c) have grammatical equivalents, as in (29a,c):

(29) a. Yo lo creo (a él).
   b. *Yo lo creo decir la verdad.
   c. Yo creo que él dice la verdad.

2.2.4 Government Theory

The theory of government enters into many areas since it relates directly to properties that hold over structural configurations. Thus a lexical head governs the phrases that it subcategorizes, Case is assigned under government (as noted in the previous section), and the

---

5 Š–deletion presents a problem given the formal definition of government (Chomsky 1981:250), since a maximal projection will block government of an element by a higher predicate. Taking S to be a maximal projection whose head is INFL, as assumed here, there is no way in which the specifier of INFL can be governed without crossing S. On the other hand, if S is the head of Š, then S is no longer a maximal projection, and after Š–deletion the predicate may govern without crossing any maximal projection. A third possibility is to allow Š–deleting predicates to subcategorize for either Š or S (i.e. COMP₂ or INFL₂), preserving the formalism that arguments are always maximal projections, but less desirable from the aspect of learnability. As this is still an issue under investigation, the stipulation is simply made that predicates with the Š–deletion property may govern an embedded subject.
binding principles are defined in terms of government.

The following definition of government is taken from Chomsky (1982:19):

\[ (30) \]

\[ a \text{ governs } \beta \text{ if } \]

(i) \[ a = X^o \text{ (i.e. } a \text{ is the head of phrase } X) \]

(ii) \[ a \text{ c-commands } \beta \]

(iii) \[ \beta \text{ is not protected by a maximal projection} \]

The definition of c-command (\textit{=} "constituent command" (Reinhart 1976)), adapted from Chomsky (1981:166), is given as:

\[ (31) \]

\[ a \text{ c-commands } \beta \text{ if } \]

(i) \[ a \text{ does not contain } \beta \]

(ii) \[ all \text{ the maximal projections that contain } a \text{ also contain } \beta \]

(31) allows the head of a phrase to c-command its complement from the X\textsuperscript{1} level, its specifier from the X\textsuperscript{2} level, and an adjunct attached at the level of the maximal projection, as in stylistic inversion of the subject, attaching to VP, in Romance languages. These three configurations are illustrated below:

\[ \begin{array}{c}
X^2 \\
\mid\
\mid \\
W^2 \\
\mid\
\mid \\
Y^2 \\
\mid \\
X^1 \\
\mid \\
X^0 \quad Z^2
\end{array} \]

In the diagram, the head \( X^0 \) c-commands its complement \( Z^2 \), its specifier \( Y^2 \), and the adjunct \( W^2 \) adjoined to \( X^2 \).

As to the notion of protection, \( \beta \) is \textit{protected} from \( a \) by a maximal projection if the maximal projection includes \( \beta \) but not \( a \). The relationship between government and c-command is illustrated below with the clause \textit{They ran towards me}: 
The verb \( V^0 \) c-commands and governs the preposition \( P^0 \), which c-commands and governs \( N^0 \). \( V^0 \) c-commands \( N^0 \) but does not govern it since it is protected by the maximal projection \( P^2 \). \( P^0 \) does not c-command \( V^0 \), and \( N^0 \) does not c-command anything. \( \text{INFL}^0 \) c-commands and governs the subject NP.

The first condition for government (30i) allows any category within the X-bar system to be a potential governor. This originally included only those elements with features \([ \pm N, \pm V ]\), but under the system developed in Section 2.2.1 now also includes \( \text{INFL} \) and
2.2.5 Binding Theory

Binding theory establishes coreference relations among noun phrases. It is assumed, for the purpose of coreferencing, that every NP is assigned an index, and that if two NPs have the same index, they are intended to refer to the same entity. Indices are assigned via some indexing rule which may happen at lexical insertion time or within some process at LF, where interpretation occurs. A natural condition on indexing is that NPs sharing an index must agree in person, number and gender.

The notion of binding is given by the following definition:

(32) \( a \) binds \( \beta \) if
    (i) \( a \) and \( \beta \) have the same index
    (ii) \( a \) c-commands \( \beta \)

If an element is not bound, it is free.

Three classes of NPs are identified: anaphors, pronominals, and names. Anaphors have no independent referential qualities and must take their reference from another NP, its antecedent, within the sentence. They include reflexives (e.g. himself) and reciprocals (e.g. each other). Pronouns (e.g. he, me, etc.) also have antecedents, but the antecedent may either appear within the sentence or outside the sentence. Names include all other (lexical) NPs (e.g. John, dogs, honesty, etc.).

Chomsky (1982) assigns the features \([\pm \text{ANAPHOR}]\) and \([\pm \text{PRONOUN}]\) to each type of noun. Anaphors are marked with \([+\text{ANAPHOR},-\text{PRONOUN}]\), pronouns with \([-\text{ANAPHOR},+\text{PRONOUN}]\), and names are marked \([-\text{ANAPHOR},-\text{PRONOUN}]\). Additionally, non-lexical NPs, both base-generated and traces that arise through movement, also contain these features. The trace left by movement of an NP to an argument position (A-position), as in passives and with raising predicates like seem, likely, pattern with lexical anaphors, while movement to non-argument (= \( \overline{A} \)) positions, such as wh-movement to COMP, leaves behind a

---

\(^6\) Chomsky uses the term R-expression ("referential expression") for name.
variable trace that patterns with lexical names. The null subjects that occur in pro-drop languages pattern after lexical pronouns. The non-lexical PRO contains the features [+ANAPHOR,+PRONOUN] and has no lexical counterpart.

The binding conditions for these three classes are stated as follows:

(33) Condition A: Anaphors must be bound in their governing category.
Condition B: Pronouns must be free in their governing category.
Condition C: Names must be free in all governing categories.

The definition of governing category is:

(34) \( a \) is the governing category for \( \beta \) if and only if \( a \) is the minimal S or NP containing \( \beta \) and a governor of \( \beta \).

Since PRO is both pronominal and anaphoric, it follows from the binding conditions that it cannot have a governing category and therefore must be in an ungoverned position.

The sentences in (35) illustrate the binding conditions for lexical NPs, given the possible indexing combinations:

(35) a. [ Jane\(_i\) said [ Wendy\(_j\) saw her\(_k\) in the mirror ]]

b. [ She\(_i\) said [ Wendy\(_j\) saw herself\(_k\) in the mirror ]]

In (a), if \( j= k \), then her is bound to Wendy, violating Condition B for pronouns. Therefore, \( j \neq k \), i.e. Wendy and her must refer to distinct persons. Her may be coreferential with Jane if \( i = k \); nothing demands nor prohibits it. In (b), she and Wendy cannot refer to each other since Wendy must not be bound anywhere, by Condition C. Wendy and herself must be coreferential, by Condition A, which therefore implies that she and herself cannot be coindexed because \( i \) and \( j \) must be distinct.

2.2.6 Bounding Theory

Bounding theory refers to the bounds placed on movement under the transformation Move-\( a \). Move-\( a \) obeys the condition of subjacency, where a single application of Move-\( a \) cannot cross more than one bounding category. For English, NP and S are considered

\footnote{A more precise definition of governing category includes the notion of accessible SUBJECT (Chomsky 1981:212), although (34) is satisfactory for current purposes.}
bounding categories; for Spanish, NP and $\bar{S}$ are bounding categories. Subjacency appears to hold of movement-to-COMP, movement-to-subject, extraposition, and inversion of subject to post-verbal position, as in *There arrived three men from England* and stylistic inversion in the Romance languages.

The execution of Move-a in relation to subjacency is illustrated in the following example:

(36) Who do you believe (that) Renate saw?

(37) a. $[\bar{S} e [S you believe [\bar{S} e [S Renate see who ]]]]$

b. $[\bar{S} e [S you believe [\bar{S} who_i [S Renate see t_i ]]]]$

c. $[\bar{S} who_i [S you believe [\bar{S} t_i [S Renate see t_i ]]]]$

(37a) is the D-structure of (36). In (b), *who* moves to COMP, leaving a coindexed trace in base position. Only one S node is crossed. In the next cycle (c), *who* moves to the matrix COMP, again crossing only one S node. If *who* had moved directly from its base position to the matrix COMP, it would have crossed both the embedded S and matrix S node, which would be a violation of subjacency.

Clearer evidence for subjacency as a condition on movement is shown by the following sentences in (38) with the D-structure in (39):

(38) a. It seems to be certain that John will win.

b. It seems that John is certain *t* to win.

c. John seems *t* to be certain *t* to win.

d. *John seems (that) it is certain *t* to win.

(39) $[S_1 NP, seem [S_2 [S_3 NP, be certain [S_5 [S_7 NP, win ]]]]]$

Given (39), (38a) results from no movement and NP$_1$ = *John*. (38b) results from movement from NP$_3$ to NP$_2$, and (38c) from a cyclic movement through NP$_2$ to the matrix subject, NP$_1$. (38d) is ungrammatical because *John* has moved from its base position directly to the matrix position, crossing two bounding categories, S$_2$ and S$_3$.

---

1 It may be noted that (38d) seems to be a violation of the binding theory for Condition A for NP-traces, since the trace does not appear to be bound in its governing category. However, given the modifications to the binding theory which include the notion of accessible SUBJECT, the trace in (38d) is bound; therefore, the only factor that accounts for its ungrammaticality is subjacency.
Subjacency is also demonstrated for NP nodes by the following example:

(40) a. \[ \_g \ who_i \ [S \ do \ you \ believe \ [S \ t_i \ [S \ Renate \ saw \ t_i \ ]]] \]
   b. \[^g \ who^- \_g \ do \ you \ believe \ [NP \ the \ claim \ [S \ t_i \ that \ [S \ Renate \ saw \ t_i \ ]]] \]

In (a), \textit{who} moves by successive cyclicity from base to embedded COMP to matrix COMP. However, in (b), \textit{who} moves from base to embedded COMP but then must cross an NP and the matrix S in order to get to the matrix COMP, crossing two bounding categories.

Evidence in favor of \( S \) being a bounding category for Spanish comes from the interaction of \textit{wh}-movement and verb fronting. The details of this interaction are covered in Section 4.4.4, Verb Fronting.

2.2.7 Control Theory

Control theory refers to the identification of PRO, illustrated by the paradigm sentences below:

(41) a. Jim persuaded Gail [ PRO to leave ]
   b. Jim promised Gail [ PRO to leave ]
   c. It is unclear [ what [ PRO to do ]]
   d. John said [ PRO to leave ]
   e. It is unclear to me [ what [ PRO to do ]]

In (a), \textit{Gail} controls \textit{PRO} and identifies it. In (b), \textit{PRO} is controlled by \textit{Jim}. The determination of control is evidently a property of the matrix verb; \textit{persuade} is an object-control verb, while \textit{promise} is subject-control. In (c), \textit{PRO} is uncontrolled and therefore arbitrary in reference. Uncontrolled \textit{PRO} often arises from the unavailability of an antecedent, as in (41c), but in other cases must be stipulated by the main verb, as in (d), where \textit{say} requires that \textit{PRO} be arbitrary and specifically not be identified with the subject, in contrast to (e) where \textit{PRO} may optionally refer to \textit{me}.

The identification of \textit{PRO} is relevant to determining its reference at LF, and thus is not of significance in the syntax. Furthermore, identification should be similar across languages for the equivalent lexical item, so that translation poses no special problems for determining control. Therefore control theory is not implemented in the current model.
Chapter 3
REPRESENTATIONS

This chapter covers the Prolog representations of the linguistic structures. It includes descriptions of the lexicon, phrasal structure, empty categories, and grammar rules. The implementation of the transformational rules for mapping between D- and S-structure is given in Chapter 4, and conditions that determine the well-formedness of structures are covered in Chapter 5.

3.1 Lexicon

The lexicon consists of the dictionary and tables for generating new entries into the dictionary. The tables include the suffix table, contraction table, and collocation table.

3.1.1 Dictionary

The dictionary is a database of unit clauses of the form:

(1) \[ \text{dict}(\text{Lang}, \text{Word}, \text{Cat}, \text{LexData}). \]

where \( \text{Lang} \) = 'english' or 'spanish'

\( \text{Word} \) = the lexical entry

\( \text{Cat} \) = the lexical category, one of:

\begin{align*}
n & = \text{noun} \\
v & = \text{verb} \\
a & = \text{adjective} \\
p & = \text{preposition} \\
art & = \text{article} \\
perf & = \text{perfective} \\
prog & = \text{progressive} \\
comp & = \text{complementizer} \\
infl & = \text{modal} \\
\text{stem#} & = \text{numbered stem for regular orthographic stem-changing verbs (Spanish only)}
\end{align*}

\( \text{LexData} \) = list of lexical data, which includes one or more of the following terms:

\begin{align*}
f & = \text{feature list} \\
\text{subcat} & = \text{subcategorization frame} \\
\text{root} & = \text{root word, for inflected entries}
\end{align*}
The feature list \( f \) may include the following features:

- **per** = person (=per(1), per(2), per(3))
- **pl** = number (pl(−)=singular, pl(+)=plural)
- **fem** = gender (fem(−)=masculine, fem(+)=feminine)
- **tns** = tense (=tns(pres), tns(past), tns(fut); infinitive=tns(−))
- **perf** = perfective aspect (=perf(+))
- **prog** = progressive aspect (=prog(+))
- **wh** = wh feature (=wh(+))
- **theta** = indicates a non-\( \theta \)-marked subject (=theta(−))

The following terms may also be included in LexData for root verbs:

- **irr** = list of irregular inflected forms
- **perf** = irregular perfective form
- **prog** = irregular progressive form
- **stem#** = stem for orthographic stem changes
- **sdel** = S-deletion feature (=sdel(+))

Nouns may include the following terms in LexData:

- **pl** = irregular plural form
- **fem** = irregular feminine form
- **proper** = feature for indicating proper nouns (=proper(+))
- **pro** = pronominal feature (=pro(+))

The following are examples of dictionary entries:

\[
\begin{align*}
dict(english,woman,n,[pl(women),spanish(mu\textcolor{red}{j}er)])
&.dict(english,woman,n,[f(pl(+)),root(woman)])
&.dict(english,i,n,[f(per(1)),pro(+),proper(+),spanish(yo)])
&.dict(english,go,v,[subcat(p),subcat(comp),perf(gone),
    irr([goes,went]),spanish(ir)])
&.dict(english,went,v,[f(tns(past)),root(go)])
&.dict(english,seem,v,[f(theta(−)),sdel(+),subcat(comp),
    spanish(parecer)])
&.dict(english,which,art,[f(wh(+)),spanish(cu\textcolor{red}{ál})])
\end{align*}
\]

Default features pertain to lexical entries in order to reduce the size of the lexicon. If a default feature does not apply, an explicit feature is required to override it. Default features are the following:

---

As mentioned in Section 2.2.2, the 'theta(−)' feature, applicable to \( \theta \)-marking predicates, is included as a convenience in determining the thematic status of subjects. When percolated to INF\textcolor{red}{L}\textsuperscript{3} by \( \theta \)-Extraction (see 6.4.4), the feature will cause movement to or from the subject position by Move-\( \alpha \) or the reverse of Move-\( \alpha \), respectively. This highly simplified mechanism is in lieu of one which indicates \( \theta \)-role assignment by means of some notation such as 'agent=[NP,S]', i.e. the \( \theta \)-role of agent is assigned to the NP immediately dominated by S.
(2) a. Nouns: \( \text{per}(3), \text{pl}(\cdot) \) (i.e. 3rd person singular)  
    \( \text{fem}(\cdot) \) (Spanish nouns default to masculine)  

b. Articles: \( \text{pl}(\cdot), \text{fem}(\cdot) \) (i.e. singular masculine, Spanish only)  

c. Verbs: \( \text{tns}(\cdot) \) (Spanish only)  

English verbs that appear uninflected may actually be the present tense of first or second person, or third person plural, as in:  

(3) a. I win/We win.  
    b. You win.  
    c. They win.  
    d. NP to win.  

When analyzing an apparently uninflected verb, agreement features will be taken from the subject unless the infinitive marker \( \text{to} \) is present, as in (d), in which case no agreement features are associated with the verb. No such action is required with Spanish verbs since its inflection system is generally sufficient to fully identify agreement features.  

Since the translations performed in this model are for equivalent lexical items, it is theoretically possible to eliminate duplicate features by containing the common features in a language-neutral lexicon. The language-specific dictionary would then contain any idiosyncratic features along with pointers to the corresponding common entry. For example, the verb \text{believe} and its Spanish equivalent \text{creer} could be represented somewhat as follows:  

(4) \begin{align*}  
\text{dict}(\text{english}, \text{believe}, \text{v}, [\text{neutral}(\text{xl23}), \text{sdel}(\cdot), \ldots]) . \\
\text{dict}(\text{spanish}, \text{creer}, \text{v}, [\text{neutral}(\text{xl23}), \ldots]) . \\
\text{dict}(\text{neutral}, \text{xl23}, \text{v}, [\text{subcat}(\text{n}), \text{subcat}(\text{comp}), \\
\text{english}(\text{believe}), \text{spanish}(\text{creer}), \ldots]) . 
\end{align*}  

Both \text{believe} and \text{creer} share the same subcategorization frames as shown in the entry for 'xl23', a unique "system-generated" identifier pointed to by the equivalent verbs. Other features in common, such as \( \theta \)-marking properties and selectional restrictions, would also be included in the lexical data for 'xl23.' Only the language-specific information (e.g. 'sdel(\cdot)') for \text{believe} but not for \text{creer}, would be included in the language-specific entries.  

This relational approach to the lexicon, as well as being theoretically attractive, would simplify the addition of other languages to the system, provided the lexical forms have a high
degree of equivalence to the neutral forms in the lexicon. The overall size of the combined lexicons would certainly be reduced. However, access time would approximately double, since at least two entry look-ups would be required to analyze each word.

In this system, a combined approach is taken. Subcategorization information is included only for English lexical items; the Spanish equivalents have a pointer to the English entry for accessing that information. All other common features, e.g. person, are duplicated in the entry for each language. For example, \textit{believe} and \textit{creer} have the following entries:

\begin{verbatim}
(5) dict(english,believe,v,[subcat(n),subcat(comp),
sdel(+),spanish(creer)]).
dict(spanish,creer,v,[prog(creyendo),english(believe)]).
\end{verbatim}

3.1.2 Suffix Table

The suffix table is used in the morphological analysis of inflected words. The structure of the table is a unit clause of the form

\begin{verbatim}
(6) suffixtable(Lang,Suffix,RootEnding,Cat,Features).
\end{verbatim}

where \texttt{Lang} = 'english' or 'spanish'

\texttt{Suffix} = one or more letters in the suffix

\texttt{RootEnding} = zero or more letters to be added to the word after the suffix is removed in order to form the root word

\texttt{Cat} = lexical category

\texttt{Features} = features associated with the suffix.

Below are some examples of the suffix table:

\begin{verbatim}
suffixtable(english,ies,y,n,[pl(+)]).
suffixtable(english,s,"",n,[pl(+)]).
suffixtable(english,ies,y,v,[pl(-),per(3),tns(pres)]).
suffixtable(spanish,a,o,n,[fem(+)]).
suffixtable(spanish,a,ar,v,[pl(-),per(3),tns(pres)])..
suffixtable(spanish,iendo,ir,v,[prog(+)]).
\end{verbatim}
3.1.3 Contraction Table

The contraction table is used to split a contraction into its constituent words during lexical analysis, and to form contractions during lexical generation. The Spanish lexicon contains the following contraction table:

```prolog
contraction(spanish, al, [a, el]).
contraction(spanish, del, [de, el]).
```

English contractions are optional and include entries such as:

```prolog
contraction(english, i'm, [i, am]).
contraction(english, i've, [i, have]).
```

3.1.4 Collocation Table

The collocation table includes groups of words that distributionally behave as a single unit. The collocated unit appears as a single entry in the dictionary. The collocation table for English includes the following:

```prolog
collocation(english, [how, much], how_much).
collocation(english, [how, many], how_many).
```

The corresponding dictionary entries are:

```prolog
dict(english, how_much, n, [f([wh(+)]), spanish(cuánto)]).
dict(english, how_many, n, [f([wh(+), pl(+)]), spanish(cuántos)]).

dict(english, how_much, art, [f([wh(+), pl(-)]), spanish(cuánto)]).
dict(english, how_many, art, [f([wh(+), pl(+)]), spanish(cuántos)]).
```

Some of the entries in the Spanish collocation table for prepositions include:

```prolog
collocation(spanish, [a, el, lado, de], al_lado_de).
collocation(spanish, [acerca, de], acerca_de).
collocation(spanish, [alrededor, de], alrededor_de).
collocation(spanish, [antes, de], antes_de).
collocation(spanish, [después, de], después_de).
```

Note that the dictionary entries could be any uniquely coded item, such as x123 in place of `how_many`. The latter is used only to facilitate human scanning of the dictionary. Coded entries would be reasonable for phrases such as *How do you do, thank you*, and similar invariant expressions. The collocation table, when extended in this way, effectively becomes a phrasal dictionary.
3.2 Phrase Structure

The representation of phrase structure employed here is a modification of the X-bar system detailed in 2.2.1(14). The problem with that representation is that the left-recursive production in (14c) prevents a direct encoding of the rule in Prolog. Therefore the adjuncts in that rule are attached at the X₂ level rather than at the X₁ level. Although this leads to structural misrepresentations, the effect on translation is nil, at least for the range of Spanish and English sentences handled by this system. The resulting phrase structure rules are:

(7)  a. X₂ → (Spec) X₁ (Post-Adjuncts)
b. X₁ → (Pre-Adjunct) X (Complement)

The list representation for the phrase X₂ has the following general form:

(8)  [Node, Specifier, [Pre-Adjunct, X, Argument₁,...], Post-Adjunct₁,...]

The first element of the list, Node, is itself a two-element list which specifies the category of the phrase and the list of features as percolated from its constituents. The second element is the specifier, if it exists; otherwise it is simply the letter "e" (= empty). The third element is the phrasal structure at the X₁ level, which consists of an adjunct, or "e" if none, followed by the lexical head X, followed by zero or more arguments that complement the head. Following the third element are zero or more adjunct structures.

In the following subsections, the list representation is shown for each value of X as an instantiation of the pattern in (8). A complete example of a clause is presented in 3.2.6.

3.2.1 COMP

The clause is represented by a COMP phrase (= 5), which has the following list representation:

(9)  [COMP, WH, [e, Complementizer, INFL], Mode]

where WH is either empty (=e) or a wh-phrase;

Complementizer is either empty or lexical;

INFL is the INFL₂ (=S) complement to COMP;

Mode is the mode of the sentence, i.e. declarative or interrogative.
Mode information is attached as an adjunct to the most deeply-embedded COMP phrase. It is represented in this system as a two-element list of the form:

(10) \[\text{[mode, MODE]}\]

where MODE is either "decl" or "ques".

3.2.2 INFL

The INFL phrase (= S) is represented by the following list structure:

(11) \[\text{[INFL, Subj, [e, Modal, VP]]}\]

where Subj is the NP subject of the clause;

- Modal is a modal verb, the English infinitive marker to, or empty;
- VP is the predicate phrase of the clause.

There are no pre- or post-adjuncts to INFL.

3.2.3 VP

The VP predicate phrase is a verb phrase with the following representation:

(12) \[\text{[VP, P1, [P2, V, Argument\textsubscript{1},...], Adjunct\textsubscript{1},...]}\]

where P1 is either an auxiliary verb (have/be in English, haber/estar in Spanish) or empty;

- P2 is either a second auxiliary or empty;
- V is the lexical verb;
- Argument\textsubscript{1},... are zero or more arguments to the verb;
- Adjunct\textsubscript{1},... are zero or more adjuncts to the verb phrase.

The second auxiliary, which in English is always a form of be, is used in conjunction with the first auxiliary to indicate perfect progressive, perfect passive, or progressive passive aspect. As there are only two auxiliary positions in (12), it is not possible to generate a perfect progressive passive form.\(^\text{10}\) The following sentences illustrate the various combinations:

\(^{10}\) This is probably an invalid restriction, given a sentence such as The Bible has been being translated for thousands of years, which is acceptable to some speakers. The allowance for three auxiliary forms could be provided as a matter of execution by treating the passive be as a main verb, subcategorizing for a "de-verbalized" adjective. This has the
(13) a. Dorothy has bought a book.
b. Dorothy is buying a book.
c. Dorothy has been buying books.
d. A book was bought (by Dorothy).
e. A book has been bought (by Dorothy).
f. *A book has been being bought (by Dorothy).

The arguments to the verb depend on the subcategorization frame indicated in the lexical entry for the verb; the adjuncts consist of non–subcategorized phrases, here limited to prepositional phrases.

3.2.4 NP, PP, AP

Noun phrases, prepositional phrases, and adjective phrases complete the set of phrase structures represented under the X–bar system. They have the same general configuration as in (8). For NP, the specifier may be an article or quantifier word. No pre–adjuncts are provided for NPs, PPs, or APs. NPs may include any prepositional phrase as post–adjunct. Subcategorization is limited to adjectives taking clausal arguments and prepositions taking NP arguments.

3.2.5 S–Adjunction

Adjunction to S (=INFL2) occurs in inversion, such as Subject–Auxiliary Inversion (SAI) in English (cf. 4.4.1) or Verb Fronting in Spanish (cf. 4.4.4). The inverted material is contained in a term, 'Adjunct', where 'Adjunct' is either the English auxiliary or a functor containing the Spanish restructured verb, i.e. a list of one or more verbal elements -- see 4.4. The inverted material is adjoined to INFL2, as shown in the tree diagram below:

10(cont’d) theoretical drawback of treating passive be sometimes as an auxiliary, as in (13d,e), and sometimes as a main verb, as above.
As a list structure, it has the following representation:

(14) \[\text{INFL, Adjunct, [INFL, Subj, [e, Modal, VP]]}\]

3.2.6 Example

The complete S-structure list representation is given in (16) for the sentence in (15). Indentation and spacing are added to improve readability.

(15) The man had seen many things from his window.

(16) \[[\text{comp,FC}], e, e,\\ \text{[[infLFI], [[n,FN1], the, [e, man]], [e, e,\\ \text{[[v,FV]}, had, [e, seen,\\ \text{[[n,FN2], many, [e, things]]],\\ \text{[[p,FP]}, e, [e, from,\\ \text{[[n,FN3], his, [e, window]]]]]],\\ [\text{mode,decl}]]}}\]}

The features, prefixed with F– (e.g. FC), are not shown in detail, but indicate such things as person, number, gender, tense, etc.

3.3 Empty Categories

Empty categories (ECs) are either base–generated, such as PRO and the null subject pronoun \textit{pro} ("little pro"), or result from Move–\textit{a}, where the moved element, \textit{a}, leaves behind a trace \(t\) coindexed with \textit{a}. The ECs PRO, \textit{pro}, NP–trace, NP–variable, and PP–variable have
the following general representation:

(17) $[\text{NODE}, e]$  

where NODE is either NP or PP. The features contained within NODE distinguish between the different types of ECs. PRO contains no features.\(^{11}\) pro contains agreement features and [+PRONOUN]. NP-traces have agreement features, an index, and the features [+ANAPHOR,-PRONOUN]. Variables have the [+WH] feature, an index, and [-ANAPHOR,-PRONOUN]. For NP-variables, agreement features are also included. The representations of the ECs are summarized below:

(18) a. PRO $[n,[n], e]$  
b. pro $[n,[pro( + ),PER,PL,FEM]], e]$  
c. NP-trace $[n,[indx(I),ana( + ),pro(-),PER,PL,FEM]], e]$  
d. NP-variable $[n,[indx(I),wh(+),ana(-),pro(-),PER,PL,FEM]], e]$  
e. PP-variable $[p,[indx(I),wh(+),ana(-),pro(-)]], e]$  

The agreement features PER, PL, and FEM are set appropriately.

3.4 Grammar

In keeping with the conceptualization of a UG and language-specific core grammars, the implementation of the grammar has been divided into two components. The UG component is language-independent and consists of the phrase structure rules utilizing the X-bar system, together with the transformational component for relating surface structures to D-structures. The core grammars, one for Spanish and one for English, contain the language-specific properties that interact with UG in the processing of sentences.

\(^{11}\) This is counter to current GB theory which holds that PRO has agreement features but no phonological content. Values are assigned to the features as a consequence of control theory. As this is not implemented here, and the agreement features serve no other purpose in this model, they are absent from the feature list.
3.4.1 Phrase Structure Rules

The phrase structure rules are written in the grammar rule notation for Prolog (Clocksin and Mellish 1981). They follow the rules in (7) for expanding the $X^2$ and $X^1$ phrasal categories along with their associated specifiers, complements, and adjuncts.

The basic grammar rules for parsing any phrase of type $X$ are the following:


b. $xl(L, C, [Pre, Head|Post])$ —> preadjunct($L, C, Pre$), $x(L, C, Head, Args)$, complements($L, Args, Post$).

c. $x(L, C, W, Args)$ —> $[W], \{\text{extract}(L, W, C, _, \text{subcat}, Args)\}$.

In (a), the $x_2$ clause states that for language $L$ and category $C$, the $X^2$ phrase has the phrase structure indicated by the third parameter. To form the phrase structure, a specifier is parsed, followed by the $X^1$ phrase, followed by any post-adjuncts. The $xl$ clause (b) is similar; a pre-adjunct is parsed, followed by the lexical head, followed by the complements to the head. In (c), the lexical head is indicated by the single token $W$, taken from the input string. The extract clause extracts the subcategorization frame, indicated by 'Args', by referencing the dictionary entry for $W$. The subcategorization frame is used by the complements clause in (b) to continue the parsing.

When $C=\text{COMP}$, 'Head', i.e. the complementizer, may be empty. Similarly when $C=\text{INFL}$, the modal may not be present. Therefore, the following two phrase structure rules are added; the argument of COMP is set to INFL, and the argument of INFL is set to VP:

(20) $x(_, \text{comp}, e, [\text{infl}])$ —> $[]$.

$\text{x}(_, \text{infl}, e, [\text{v}])$ —> $[]$.

The specifiers are represented by the general rule:

(21) $\text{specifier}(L, C, \text{Tree})$ —> Spec.

with $L, C$, and Tree being the language, category, and parse tree, respectively. The particular values of Spec for different Cs are:
(22) a. for C=COMP, Spec = \textit{wh}-phrase (i.e. NP or PP with feature [+WH])
b. for C=INFL, Spec = NP
c. for C=N, Spec = article
d. for C=V, Spec = perfective or progressive auxiliary
e. for C=anything else, Spec is empty.

In (a), the \textit{wh}-NP or \textit{wh}-PP must be lexical during parsing, although in the transition from S-structure to D-structure, or vice versa, it may be interpreted as a \textit{wh}-trace, depending on circumstances. In (b), subjects of clauses are always present, by the Extended Projection Principle; therefore, the NP is lexical or an EC, never just "e" as for the other categories. (c) and (d) are of course optional and may be empty.

The complements of a lexical head are presented as a list of lexical categories. The list may be empty if no arguments are subcategorized for. Or the list may indicate optional categories, such as for the verb \textit{eat}, which may optionally take an NP object:

(23) a. Philip eats.
   b. Philip eats tacos.

Optional arguments are enclosed in braces, e.g. \{n\}. The dictionary entry for \textit{eat} and its Spanish equivalent \textit{comer} would contain the following:

\[
\text{dict(english,eat,v,[subcat\{n\},...,spanish(comer)]).}
\]
\[
\text{dict(spanish,comer,v,[english(eat),...]).}
\]

The Spanish complement is taken from the English entry, as described in 3.1.1.

Since the course of parsing is dependent on the syntactic properties (i.e. subcategorization frames) of the lexical items, the arguments are parsed using the following rules, where the first argument is the language, the second is the list of arguments from the call to \textit{x} (19c), and the third argument is the list of phrase structures built for each argument.

(24)  \text{complements}(_, [], []) \rightarrow [].
\text{complements}(L, [A|As], [Arg|Args]) \rightarrow
\quad \text{parseArg}(L, A, Arg),
\quad \text{complements}(L, As, Args).

The first clause handles the empty list. The second parses the argument 'A', forming the structure 'Arg', and recursively calls \text{complements} on the remainder of the list, 'As'.

The call to \text{parseArg} is simply a call to \textit{x2}, or to \text{optionalArg} which also calls \textit{x2} but succeeds even if the \textit{X2} constituent is not found.
parseArg(L, {Arg}, Tree) --> optionalArg(L, Arg, Tree).
parseArg(L, Arg, Tree) --> x2(L, Arg, Tree).

optionalArg(L, Arg, Tree) --> x2(L, Arg, Tree).
optionalArg(_, _, []) --> [].

Note that optional arguments are not the same as non-lexical arguments. If an argument is required by the subcategorization frame, it must exist either as a lexical constituent or as its trace. However, before assuming that an unfound argument is non-lexical, all subcategorization frames must be tried. If none result in a successful parse, then the possibility exists that the argument is non-lexical. This only applies to subcategorized NPs and PPs, as these are the only constituents that move in this model. For example, the verb *believe* subcategorizes for either an NP or COMP:

(26)  
\[
\text{dict(english,believe,v,[subcat(n),subcat(comp),...])}.
\]

During parsing, an NP argument will be searched for first. If not found, then a clause will be searched. If that is not found, then because NP is a possible argument of believe (determined by re-scanning its list of subcategorization frames), an empty NP will be inserted into the parse tree, and the parse continues.

To accommodate empty arguments, the additional *x1* clause is added:

(27)  
\[
\text{x1(L, C, [Pre,Head|Post]) --> preadjunct(L, C, Pre), x(L, C, Head, Args), emptyArgs(L, Args, Post).}
\]

The *emptyArgs* clause will replace either an NP or PP argument with an EC, if subcategorized for.

The only pre-adjunct considered in this model is the second auxiliary to the verb. The grammar rules for this are simply:

(28)  
\[
\text{preadjunct(L, v, w) --> [w], \{\text{dict(L,w,prog,_)}\}.}
\]
\[
\text{preadjunct(_, _, e) --> [\].}
\]

Post-adjuncts may be lexical prepositional phrases (not ECs) for either NPs or VPs, or else null. Also, sentence punctuation is considered a post-adjunct to the COMP.
The remaining grammar rule is the rule for inversion. Since inversion for SAI and Verb Fronting both involve adjunction to S, an additional rule at the X' level for INFL is added:

\[(29) \quad x2(L, \text{infl}, [[\text{infl},[]],\text{Adjunct},\text{INFL}]) \rightarrow\]
\[\text{adjunct}(L, \text{Adjunct}),\]
\[\text{parse\_inversion}(L, \text{Adjunct}, \text{INFL}).\]

The rules for \text{adjunct} and \text{parse\_inversion} are both language-specific, so they are included in the core grammar section. For English, SAI involves adjunction of a modal or auxiliary. (Do is included as a modal verb, as well as a main verb. Also, the infinitive marker to is included as a modal, since modal verbs may not co-occur with infinitives.)

Spanish inversion is considerably more complicated, since inversion involves verb restructuring. The modal, both auxiliaries, and main verb, or some restricted combination of them, may be reanalyzed as a single verbal element which inverts. Thirteen combinations are allowed that maintain the ordering of elements; that is, if the main verb inverts, for example, then any and all preceding verbal elements must invert. Similarly for the auxiliaries and any elements preceding them. Also, following Torrego (1984), if perfective haber or one of its inflected forms is found inverted, then its participle must also be inverted. To allow for the possibility of all four elements inverting, an adjunct structure is created as a four-element list, where each element may either be lexically filled or empty, with the proviso that the adjunct must not be completely empty (i.e. not = [e,e,e,e]). That is, if an adjunct is formed, then at least one element has inverted.
The transformations considered in this model fall into two general categories. One is the set of language-independent rules, i.e. Move-\( \alpha \) and Move Affix. These are represented in the UG component as general movement rules, although parameter settings within the core grammars will influence their application. The other category includes idiosyncratic rules such as Do Support for English and pro-drop for Romance languages.

Each transformation is described in the following sections, presenting first the motivation for the rule, and second its implementation. The last section deals with the order of transformations within the transformational cycle.

4.1 Move Alpha

The fundamental transformation under current theories of transformational grammar is Move-\( \alpha \), i.e. "move any category anywhere". The principles of UG effectively determine all and only those applications of Move-\( \alpha \) that result in grammatical sentences, subject to parametric variation.

When a phrase moves, it forms an association chain with its base position. Each position to which the phrase moves becomes a member of the chain and inherits the index that was assigned to the base element at lexical insertion time. A \( \theta \)-role is assigned to the base position in the chain, and for lexical NPs Case will be assigned to the initial position. The movement obeys the bounding condition of subadjacency, and all non-initial positions must be properly governed by the Empty Category Principle, a well-formedness condition covered in Section 5.5.

A basic result that falls out of the Projection Principle and the \( \theta \)-Criterion is that movement is always to a non-\( \theta \)-marked (= \( \bar{\theta} \)) position (Borer 1979, 1981). If movement
were possible to another $\theta$-marked position it would receive two $\theta$-roles, in violation of the $\theta$-Criterion. $\theta$ positions include COMP, the subject of INFL if not $\theta$-marked by the predicate phrase, and adjuncts.

In this model, the landing sites for movement are taken to be empty non-$\theta$-marked subjects and COMP. Only NPs may move to subject position, by the structure-preserving hypothesis (Emonds 1976, but see Stowell 1981), and only categories with the feature $+[WH]$ may move to COMP (Chomsky 1981:115).

The process of moving an element out of base position and the reverse operation of restoring a moved element are logically just inverse operations of each other. However they are treated separately here, since the goals of the two operations are quite different; one, the latter, is concerned with relating a phrase at surface structure with its base $\theta$-marked position, while the other relates a base position with possibly many valid $\theta$ surface positions. The details of these two operations are covered more fully in 6.4.8 and 6.6.2.

4.2 Move Affix

Move Affix, or Affix-Hopping (Chomsky 1957, Baker 1978), affixes inflectional markers to verbal elements. At D-structure, tense and agreement make up the set of features in INFL (Chomsky 1981). Considering $+[TNS]$ as an affix, it "moves" from INFL to the first verbal element, which may be a modal, auxiliary, or main verb. AGR is assumed to be associated with $+[TNS]$ (but not with $-[TNS]$) so it too moves with $+[TNS]$, thereby fully determining the morphology of the verbal to which it attaches.

The perfective auxiliary (have/haber) is associated with a perfect participle, which affixes to the following verb form. Similarly, the progressive auxiliary (be/estar) supplies a progressive participle, and the passive be/ser a passive participle (although this form of passive is not often used in Spanish, and not implemented in this model).

The effects of these rules are exemplified below:
(1) a. Dana [+TNS(past)] sit at home.
b. Dana sit+TNS(past) at home.
c. Dana sat at home.

(2) a. Michael [+TNS(pres)] have+en leave.
b. Michael have+TNS(pres) leave+en.
c. Michael has left.

(3) a. They [+TNS(pres)] be+ing visit friends.
b. They be+TNS(pres) visit+ing friends.
c. They are visiting friends.

(4) a. He [+TNS(past)] can+en be+en see from the window.
b. He can+TNS(past) have be+en see+en from the window.
c. He could have been seen from the window.

The sentences of (a) are in base form, with participles associated with their auxiliaries. In (b), all affixes have moved to right–adjacent verbal elements. The resulting surface forms are shown in (c).

In the reverse process, the inflectional features of the lexical items are combined and merged with the features in INFL. The inflected forms are then replaced by their root forms. Validity tests on the combined features ensure that the appropriate form of the item has been taken. For example, the word bought may be past tense or a perfect participle, depending on the context. The reverse transformation will select the correct choice.

4.3 Null Subject

A characteristic of many languages (e.g. Spanish, Italian, Hungarian, Chinese, Japanese) that falls under the rubric of "pro–drop" is the possibility of "missing" subjects of tensed clauses, as in the following Spanish examples:

(5) a. (Yo) voy a la playa. ('I'm going to the beach.')
b. (Tú) vas a la playa. ('You [informal] are going to the beach.')
c. (El/Ella/Ud.) va a la playa. ('He/She/You are going to the beach.')
d. (Nosotros) vamos a la playa. ('We're going to the beach.')
e. (Ellos/Ellas/Uds.) van a la playa. ('They/You [plural] are going to the beach.')

In all cases the parenthesized subject pronoun is optional. It appears that, following Taraldsen (1978), when the inflectional system is sufficiently rich to identify the subject, the subject may delete. This connects the presence of AGR in INFL (obligatory with [+TNS] but absent from
with the possibility of a null subject if AGR is sufficiently specified.\textsuperscript{12} English AGR is not sufficiently rich, so empty subjects of finite declarative clauses do not occur.

The empty subject in these cases is pro, an NP with $[+\text{PRONOUN}]$ and agreement features. It is governed by AGR, as is the subject of any finite clause, and thereby may receive nominative Case.

In analyzing Spanish sentences, a null subject may indicate any one of the ECs. If the verb is inflected, then the subject is either a wh-trace or pro; it cannot be PRO because it is governed by $[+\text{TNS}]$, and it cannot be an NP-trace because, as subject, it could not be bound in its governing category, in violation of binding condition A. The distinction between pro and wh-trace is readily apparent if the verbal inflection indicates first or second person, in which case the appropriate lexical pronoun can replace the EC under recoverability of deletion. Otherwise the status of the EC will depend on whether there exists a wh-element to bind the variable.

In generating Spanish sentences, a subject pronoun may be replaced by an EC, retaining its features which include person, number, possibly gender, and $[+\text{PRONOUN}]$.

English, not being a pro-drop language, is barred from the possibility of null subjects.

\textbf{4.4 Inversion}

The inversion transformation for English is called Subject–Auxiliary Inversion (SAI), or Subject Helping–Verb Inversion (Baker 1978). To fully account for the empirical facts of SAI in English, some additional transformations are required, namely Do Support and Have/Be Raising. These are described below, followed by the counterpart inversion rule in Spanish, Verb Fronting.

\textsuperscript{12} Huang (1982) notes that Chinese has no AGR feature set, yet is a pro-drop language; he relates the identification of the empty subject not with AGR but with a higher lexical subject.
4.4.1 Subject–Auxiliary Inversion

SAI is an obligatory rule for English yes/no and wh questions, as demonstrated below:

(6)  a. Can we still be friends?
    b. Has it ever happened before?
    c. Are you lonesome tonight?
    d. Do you really want to hurt me?

(7)  a. How can I be sure?
    b. Where have all the flowers gone?
    c. Who are you kidding?
    d. Why do fools fall in love?

In the (a) examples, the modal verb inverts with the subject. In (b), the perfective auxiliary inverts; in (c) the progressive auxiliary inverts, and in (d) the "helping verb" do is inserted, then inverted.

In structural terms, the contents of the INFL head (="AUX") become an adjunct structure that attaches at the INFL^2 level:

(8)    [INFL, Adjunct, [INFL, Subj, [e, e, VP]]]

This is diagrammed below:

(a) base

(b) surface
The implementation of this rule is straightforward and reversible. From a surface structure (b) in which an adjunct was parsed, the adjunct is restored to the head of the INFL, as in (a), which must, of course, have been empty, given the ungrammaticality of sentences like *Should John can go?*. At generation, inversion of INFL occurs (at the matrix level only) merely by virtue of the sentence being a question.

4.4.2 Have/Be Raising

When no modal is present in INFL to undergo inversion, the first auxiliary inverts, as in the (6b,c) and (7b,c). It "raises" into INFL, where SAI then applies. The structures are reversible:

```
(a) base

(b) surface
```
4.4.3 Do Support

If SAI inverts a null AUX, i.e. void of a modal or auxiliary verb, the rule of Do Support is activated. The grammatical formative *do*, distinct from main verb *do* although with similar morphology, is inserted into the inverted AUX position. However, it does not apply in *wh*-questions with empty subjects: *Who saw David?*, although SAI still takes place, even if vacuously.

The current implementation departs from the traditional order by scheduling Do Support before SAI. This allows Move Affix to attach tense onto *do* before inversion takes place. The Do Support transformation is provided with a "look-ahead" capability to detect if Do Support will be required.

Do Support is illustrated below:

![Diagram](a) base (b) surface
4.4.4 Verb Fronting

Verb Fronting, or V-Preposing (Torrego 1984), is a characteristic of Spanish questions that fronts one or more [+V] elements to the head of the clause, as in the examples below:

(9) a. Debió Juan haber estado cantando tan fuerte?
   ('Should John have been singing so loudly?')

   b. Debió haber estado Juan cantando tan fuerte?

   c. Debió haber estado cantando Juan tan fuerte?

In (a), only the modal verb undergoes fronting. In (b), the modal and perfect auxiliary with its participle inverts (perfectives may not be separated from their participle – cf. *Debió haber Juan estado cantando tan fuerte?). In (c), the modal, both auxiliaries, and main verb are fronted.

As noted by Torrego, verb fronting is optional for yes/no questions but obligatory in clauses containing certain wh-phrases or their trace in COMP:

(10) a. Qué querían esos dos?
   ('What did those two want?')

   b. *Qué esos dos querían?

(11) a. Con quién sabía Juan que había admitido Ana que había hablado Pedro?
   ('Who did John know that Ana had admitted that Peter had spoken with?')

   b. Con quién sabía Juan que había admitido Ana que Pedro había hablado?

   c. *Con quién sabía Juan que Ana había admitido que Pedro había hablado?

In (11), the wh-phrase moves from the lowest embedded clause to the matrix COMP. In (11a), the wh-phrase passes through each COMP position, forcing inversion. Example (11b) is grammatical and provides evidence for $S$ as a bounding category for Move-$_a$. The wh-phrase moves directly to the second COMP position, bypassing the lowest COMP. Since the lowest COMP does not contain the wh-trace, no inversion is required. Example (11c) is a subadjacency violation as the wh-phrase crosses two $S$ nodes, evident from the fact that inversion has only occurred in the highest clause containing the wh-phrase.¹³

Several combinations of inversion are thus possible, so the V-preposing clause simply lists all patterns of base structure, surface structure and adjunct in the format shown:

¹³ Not all linguists agree with Torrego's grammaticality judgments of the sentences in (11) nor with the conclusion that $S$ is a bounding node for Spanish. Her analysis is adopted here, although other analyses may prove to be preferable.
(12) \( v\_\text{prepose}(\text{Base, Surface, Adjunct}). \)

Each argument to \( v\_\text{prepose} \) consists of a four-element list representing the contents of the modal, first auxiliary, second auxiliary and main verb. When mapping from surface to base structure, the surface-adjunct pair directly unifies with one of the combinations to derive the base structure. On generation, the base structure will, through backtracking, unify with one or more possible surface structures, giving rise to multiple transformations. For example, if a modal and main verb are found inverted during parsing, it will match the first entry below:

(13)  
\[
\begin{align*}
&v\_\text{prepose}( [I,e,e,V], [e,e,e,e], [I,e,e,V]) . \\
&v\_\text{prepose}( [I,e,e,V], [e,e,e,V], [I,e,e,e]) :- !.
\end{align*}
\]

During generation, the first entry will cause the modal and main verb to invert; subsequent backtracking to \( v\_\text{prepose} \) will select the second entry, inverting only the modal.

4.5 It Insertion

In the pro-drop languages, null subjects of clauses that are not \( \theta \)-marked remain null. In English, they must be lexicalized. The non-referential pronoun \textit{it} serves this function. \textit{(There} is the other non-referential element, but its characteristics are more complex (Chomsky 1981, Safir 1982, Williams 1984).) It Insertion is diagrammed below:

\[
\begin{align*}
\text{INFL}^2 &\quad \text{INFL}^2 [\text{PER(3)}, -\text{PL}] \\
\text{N}^2 &\quad \text{INFL}^1 \\
e &\quad \text{INFL}^0 \\
\text{V}^2 [-\text{THETA}] &\quad \text{INFL}^0 \text{V}^2 [-\text{THETA}] \\
\end{align*}
\]

(a) base (b) surface

In going from base to surface form, an empty subject of a non-\( \theta \)-marked clause has \textit{it}
inserted, with features [PER(3),-PL] (= 3rd person singular) merged into the INFL features for determining verbal inflection. Going from surface to base form, if it appears as subject of a non-\(\theta\)-marked clause, it deletes. (This is not always correct if it is in fact referential, as in the sentence *It seems to behave intelligently*. Some amount of backtracking or lookahead, ultimately relying on satisfaction of the \(\theta\)-Criterion, would be required in order to correctly handle this situation.)

### 4.6 Complementizer Insertion

It is assumed that complementizers are optionally selected, depending on specific lexical characteristics and other conditions such as embedded tense. When introducing tensed embedded clauses in English, the complementizer seems to be freely selected (or deleted, depending on the direction from which the problem is approached):

\begin{equation}
(14) \quad \text{I know (that) you think (that) I'm crazy.}
\end{equation}

Similarly, *for* may be optional (in some dialects) in infinitival clauses:

\begin{equation}
(15) \quad \text{She would prefer (for) you to leave immediately.}
\end{equation}

The complementizer *for* takes on the same Case-assigning properties as the preposition *for*, so its presence in COMP is closely connected with the Case Filter. It may not delete, for example, in the sentence:

\begin{equation}
(16) \quad \text{I'm eager for you to take part.}
\end{equation}

since *you* would not be assigned Case (although note the counterexample for *want* below).

Choice of complementizer is determined in part by lexical properties, presumably as part of the subcategorization frame that specifies clausal arguments. For example, *wonder* may take *whether* as a complementizer (or other [+WH] phrase) but not *that*:

\begin{equation}
(17) \quad \begin{align*}
\text{a. } & \text{I wonder whether Jennifer knows yet.} \\
\text{b. } & \text{I wonder who Jennifer knows.} \\
\text{c. } & \text{*I wonder that Jennifer knows yet.}
\end{align*}
\end{equation}

The *for*-complementizer is not permitted by *wonder*, although other verbs obligatorily require it:
(18) a. They waited for him to leave.
   b. *They waited that he would leave.
   c. I want (for) David to come with me.
   d. *I want (that) David come with me.

(The for in (18c) is assumed to delete when immediately adjacent to the verb after assigning Case to the embedded subject, another marked property of want -- cf. I want very much for David to come with me.) However, it is assumed that in the unmarked case, clauses may be tensed or infinitival, with choice of complementizer determined accordingly.

In this model, these peculiarities are avoided by allowing the that-complementizer to optionally appear in any tensed embedded clause. In a more complete system, some form of selectional restrictions on the subcategorized clause is necessary, as discussed in Chapter 8.

4.7 Modal Insertion

In English, future tense may be indicated by the modal will, while Spanish has a regular inflection for future tense:

(19) a. George will arrive tomorrow at seven.
   b. Jorge llegará mañana a las siete.

Similarly for conditional tense, where English sometimes uses the modal would and Spanish inflects:

(20) a. If I could, I would buy a house.
   b. Si pudiera, me compraría una casa.

In this model, only future tense is analyzed, since conditional tense generally involves a subordinate clause in the subjunctive mood (cf. (20b) above), which is not considered here.

When will is encountered in analyzing an English sentence, the modal is deleted, leaving behind the tense feature in the feature set for INFL. In generating English sentences, will is inserted into the head position of INFL if future tense is specified.

In the case of untensed clauses in English, the infinitive marker to is used. Since to indicates [−TNS], it may not co-occur with a modal, which always carries the [+TNS] feature. Therefore, the position occupied by a modal in tensed clauses is filled with to in infinitival clauses. As with will, it deletes during analysis of sentences, leaving behind [−TNS]
as a feature, and is inserted on generation.

4.8 Successive Cyclicity and Order of Transformations

The cyclic nature of transformations is relevant to the application of Move-$a$, where long-distance movement can only be explained by assuming a series of successive movements, given the bounding condition of subjacency. This was illustrated in Section 4.1, Move Alpha, and in Section 4.4.4, Verb Fronting, for $wh$-movement, and applies as well to NP-movement, as in the sentence:

(21) Thatcher seems to be likely to win.

with the corresponding S-structure representation:

(22) $[S_i, \text{Thatcher}_i$ seems $[S_j, t_i$ to be likely $[S_k, t_i$ to win $]]]]$]

The subject, Thatcher, moves from the lowest embedded clause $S_3$, where it receives its $\theta$-role, to the matrix subject via the non-$\theta$-marked subject of the intermediate clause $S_2$. Direct movement from $S_3$ to $S_1$ would violate subjacency.

Successive cyclicity does not necessarily enter into the operation of the other transformations, which are all local to the clause anyway. However, since it is required for Move-$a$, all transformations are defined in terms of the cycle, that is, as having their effect within the boundaries of a single clause, i.e. COMP phrase. This facilitates the execution of a transformational cycle by recursively calling the set of transformations on each COMP phrase. In analyzing sentences, the cycle begins at the matrix clause; in generating sentences it begins at the lowest clause.

Within one cycle, the transformations are performed sequentially. The results of a transformation may have a bearing on the functioning of subsequent ones, so that ordering becomes significant. As a consequence, the transformational cycle may take only limited

---

14 An alternative would be to introduce filters (Chomsky and Lasnik 1977) to eliminate mis-applied transformations. In that case, ordering would be irrelevant but a large number of filters might be required, possibly incurring excessive computational overhead in their execution. The result effectively imposes an implicit rule ordering, which is made explicit here and minimizes the reliance on filters.
advantage of concurrent processing in mapping from one level of representation to another. However, the conditions on well-formedness of representations could be applied simultaneously.

Because of the similarity between Spanish and English, the transformational cycle for the two languages share much in common. Ordering is the same for related transformations, and some, such as Move-\( \alpha \) and Move Affix, are general enough to be described independently of the language. The overall effect is to reduce the size and complexity of the transformational component, while maintaining relatively high cross-linguistic generality.

For English, the transformational cycle includes the following:

(23) a. Move-\( \alpha \)
b. It Insertion
c. Do Support
d. Move Affix
e. COMP Insertion
f. modal Insertion
g. Have/Be Raising
h. SAI

Move-\( \alpha \) occurs first, if at all, to move phrases into \( \bar{\theta} \) positions. If a non-\( \theta \)-marked subject position remains empty, It Insertion applies to lexicalize the subject. COMP Insertion and modal Insertion may occur at any time. Before Move Affix applies to inflect verbs, Do Support must occur, in case inversion (or "Not" Insertion, if negation were implemented) is called for. After the tense has been affixed, SAI may move the inflected auxiliary. Have/Be Raising is a convenience in the specification of SAI, so it precedes SAI.

The reverse-transformational cycle is exactly the reverse of the above, with the exception that COMP Insertion and Modal Insertion are not performed; that is, COMP "deletion" and Modal "deletion" are not necessary in the reconstruction of D-structure from surface structure. They are effectively deleted during the translation stage. Further, some auxiliary actions are performed, such as \( \theta \)-extraction (Section 6.4.4), that do not have counterparts in the transformational cycle.

For Spanish, the transformational cycle includes:

(24) a. Move-\( \alpha \)
b. Null Subject
c. Move Affix
d. COMP Insertion
e. Verb Fronting

Move-a applies first, after which the Null Subject option may apply. Verb Fronting is applied to the matrix clause of yes/no questions and to any clause in which Move-a has moved a wh-phrase to COMP. However, before Verb Fronting occurs, Move Affix generates the verbal inflections. As in English, COMP Insertion may occur at any time. The reverse transformational cycle performs these transformations in reverse order, again bypassing COMP Insertion.
Representations of the different levels of linguistic structures (i.e. D-structure, S-structure, LF, etc.) must satisfy certain well-formedness conditions to meet standards of grammaticality. In the previous chapters we've covered the general representations of D-structure and S-structure and the rules for mapping between the two. We saw that D-structure was unique to a sentence and that multiple S-structures were related to the D-structure through the basic transformation Move-α, the other transformations being highly idiosyncratic and relatively uninteresting. Move-α was seen to obey the principle of subjacency, a condition on movement rather than on representation. See Huang (1984) and Chomsky (1982) for argumentation supporting this viewpoint.

In this chapter we look at the conditions that determine the well-formedness of D- and S-structures, and how these conditions are implemented in Prolog. Certain conditions are expressed as filters on over-generation, specifically designed to filter out ill-formed S-structure constructions. Only two such filters are modeled here, the Doubly-filled COMP Filter and the Case Filter. Other well-formedness conditions are expressions of the principles of grammaticality, also implemented in the form of filters. These include the Binding Conditions and the Empty Category Principle. The Extended Projection Principle and θ-Criterion, discussed first, are satisfied by virtue of the structural design of the model; no explicit filtering mechanisms are used.

5.1 Extended Projection Principle and Theta-Criterion

The Extended Projection Principle and θ-Criterion interact to constrain both S-structure and D-structure. The two principles guarantee conservation of θ-roles among the different levels of representation within the X-bar system of phrase structure. This correspondence is
maintained by the grammar rules which are driven by subcategorization frames within the lexicon, and by the fact that lexical categories in $\theta$ positions may be related to one and only one empty $\theta$-position. Elements $\theta$-marked at D-structure move within the sentence structure leaving coindexed traces in each occupied location. The indexed chain so formed contains at most one lexical phrase ("at most" since it may consist of only non-lexical PRO), and each structural position is represented in only one chain. This ensures that a definable path, not necessarily unique, exists from S-structure back to D-structure, and that the thematic properties expressed at D-structure are present in any S-structure configuration. Given that only grammatically correct sentences are analyzed by this system, the well-formedness of the representations are maintained as to the Extended Projection Principle and the $\theta$-Criterion.

5.2 Doubly-Filled COMP Filter

In the course of $wh$-movement, the COMP node may contain a $wh$-phrase in specifier position and a lexical complementizer in head position. Such a configuration is ruled out at S-structure by the Doubly-Filled COMP Filter (DFCF) (Chomsky and Lasnik 1977):

$[^{\text{COMP}}\alpha\beta]$  

where $\alpha$ is assumed to contain a $wh$-phrase and $\beta$ a complementizer, as in:

$[^{\ldots[\text{the man}^{\text{S}}[\text{COMP who that}]}^{\text{S}}[\text{you met}^{\text{S}}]]]$  

Either $\alpha$ or $\beta$ (or both) must delete in order for the clause to be grammatical.

We might have achieved the same result by stipulating that a $wh$-phrase moves to the head of COMP, thereby preventing a lexical complementizer from simultaneously appearing in head position. However, a couple of problems result from this analysis. One is that there exist languages for which the DFCF does not hold (Pesetsky 1982). Another is that $wh$-movement through COMP is assumed to leave a trace in COMP, even though a complementizer may be present in certain circumstances:

$[^{\ldots [\text{Who}_i \text{ do you think}^{\text{S}}[\text{COMP} t_i\text{ that}]}^{\text{S}}[\text{Bill saw} t_i]]]$  

Chomsky (1981) proposes that this is also ruled out by the DFCF, and that either the trace
or *that* must delete. While it might be agreed that the trace is deletable, especially if the DFCF is a filter in the PF component, the analogue to *that* in Spanish, for example, is not deletable:

(4)  

a. A quién piensas [\(\mathcal{S} [\text{COMP } t_i \text{ que }] [\mathcal{S} \text{ vio Guillermo } t_i ]\)]  
b. *A quién piensas [\(\mathcal{S} [\text{COMP } t_i ] [\mathcal{S} \text{ vio Guillermo } t_i ]\)]

Therefore the head position of COMP must be reserved for a complementizer for those languages that require it, and the specifier position of COMP may still be used to contain a *wh*-phrase, preserving the derivational history of movement in the syntax.

The DFCF, then, becomes a condition on the representation of S-structure, where the COMP may not contain \(\alpha\) and \(\beta\), both lexical.

As a Prolog clause, this is stated simply as:

(5)  

\[
\text{dbl_comp_filter}([[\text{comp, }], \text{WH}, [\_, \text{COMP, }]]) :- \\
\text{lexical(WH),} \\
\text{lexical(COMP),} \\
!,\text{fail}.
\]

The argument to *dbl_comp_filter* is the S-structure of the clause, a COMP\(^2\) phrase. The specifier is WH and the head is COMP, all other constituents being immaterial. If both WH and COMP are lexical, where lexical is defined as:

(6)  

\[
\text{lexical([X,e]) :- !,fail.}
\]

i.e. not empty, then *dbl_comp_filter* fails and the S-structure is rejected.

5.3 Case Filter

The Case Filter basically states that all lexical NPs must have Case. This is formulated in terms of the chain \(C=(a_1, a_2, \ldots, a_n)\), where each \(a_i\) is an NP in an argument position (A-position) and \(a_i\) binds \(a_{i+1}\), thus sharing an index. Furthermore, for all \(i>1\), \(a_i\) is an EC. The head of the chain, \(a_1\), is either in a Case-marked position or PRO, and no other \(a_i\) in the chain may be Case-marked. The head of the chain, then, must either be lexical or PRO. The chain is termed an A-chain since it involves only A-positions.
The implementation of the Case Filter involves inspecting every A-position and verifying that lexical NPs in these positions are Case-marked. The A-positions of interest are the subjects of clauses and objects of verbs. Objects of prepositions are always assigned Case by the preposition, and may not move to another Case-assigned position; therefore, they do not need explicit verification.

The subject of a clause may receive Case in one of two ways: through the mechanism of Exceptional Case-Marking (ECM) or through [+TNS] in INFL, the former assigning objective Case, the latter nominative. If the subject is lexical, it must be in one of these two environments. If it is not lexical, the Case Filter does not apply; other principles will determine the status of the EC.

Exceptional Case-Marking is an idiosyncratic feature of English that allows for sentences such as:

(7) a. I believe (that) John is my friend.
    b. I believe John to be my friend.

In (a), John receives nominative Case from the tensed embedded INFL. In (b), nominative Case is not assigned since the embedded clause is infinitival, which normally prevents lexical NPs from appearing in subject position, as in:

(8) *I think John to be my friend.

The transitive verb believe "deletes" the subcategorized S node and thus is able to directly govern the embedded subject, assigning it Case by virtue of transitivity.

The Case Filter is implemented by separating the analysis of Case into two stages, one for analyzing the subject position, the other for object position:

(9) case_filter(L, Sstructure) :-
    subj_position(L, Sstructure),
    obj_position(L, Sstructure).

The two configurations in which Case is assigned to the subject position are represented by the following Prolog clauses:

15 Restructuring of a preposition with the verb may occur in certain cases, such as This task should not have been taken on. The restructured verb take on assigns Case, not the preposition. However, this model does not contain cases of such restructuring.
In the first subj_pos statement, the configuration for ECM is presented, where a VP dominates a clause COMP which contains no wh-phrase, trace, or complementizer. Then, if the subject is lexical and INFL contains the feature [-TNS], ECM must obligatorily apply. (The cut symbol (!) prevents backtracking to the next subj_pos statement.) The ecm statement is defined as:

(11) ecm(english,V) :-
    is_sdeleter(english,V,v),
    is_transitive(english,V).

where is_sdeleter scans the lexical entry of the verb for the feature 'sdel(+)', and is_transitive checks verbs for subcategorization of NP. ecm is present only in the core grammar for English; Spanish has no ECM properties.

In the second subj_pos statement, a lexical subject not in an ECM environment must be in a tensed clause, i.e. the features of INFL must not contain [-TNS].

Appropriate Prolog clauses are added to recursively search through the phrase structure for the two subject Case-marking configurations.

Objective Case is always assigned by transitive verbs, as for prepositions, except in the passive construction, where the transitive verb is "de-transitivized" and fails to assign Case to its object. Therefore if the object of a transitive verb is lexical but the phrase is marked for passive, the configuration will be ruled out. The Prolog clause is defined as:

(12) obj_pos( [[v,FV], _, [_, _, [n,_, _,_]_]) :-
    not member(pass(+),FV).

This situation occurs if, for example, Move-a fails to move the NP out of object position, as in:

(13) *(It) was taken a book from the shelf.

The NP a book must obligatorily move to subject position, where it will receive Case from the
tensed INFL.

5.4 Binding Conditions

The relation between an NP and a possible antecedent is expressed by the binding conditions (and by the theory of control for PRO). As stated in Section 2.2.5, the binding conditions hold over lexical NPs and non-lexical NPs, with NP-traces conforming to Condition A for anaphors, pro following Condition B for pronouns, and variables conforming to Condition C for names.

Although in principle all NPs, whether lexical or non-lexical, receive an index whose value is determined by the binding conditions, only NPs that participate in movement are explicitly checked for valid binding, since only they are given indices in this system. The binding of all other (static) NPs determines their reference for interpretation purposes, a semantic operation at LF, and therefore not of interest here. Consequently, the implementation of the binding conditions is limited to Condition A for NP-traces and Condition C for variable traces, expressed by the following top-level Prolog statement:

(14) binding_conditions(L, Sstructure) :-
    condition_A(L, Sstructure),
    condition_C(L, Sstructure).

For Condition A, an NP-trace may occur either in subject position or in object position. If the subject is an NP-trace, then it must be governed by a predicate with the S-deletion property. It may not be governed by [+TNS], thereby preventing a sentence such as:

(15) John seems [ t has left ]

(Note that (15) is more properly a Case Filter violation since the chain of John and its trace is receiving Case twice. However, the implementation of the Case Filter only applies to lexical NPs and variables, not NP-traces.)

The Prolog statement of Condition A for subject position is given in (16):
(16)  condition_A(L, [[X, _], _, [_, XW, 
    [[comp, _], WH, [e, COMP, 
    [[infl, FI], NP, _]]]])) :-
    is_nptrace(NP),
    !,
    WH=e, COMP=e,
    member(tns(-), FI),
    is_sdeleter(L, XW, X).

No explicit index matching is performed, since in this model if an NP-trace is in
subject position, it can only be bound to an antecedent in the next higher subject position, the
only possible available A-position. Empty object positions as landing sites for an antecedent are
excluded by the Projection Principle, and more distant subject positions are excluded by
subjacency.

For an NP-trace in object position, its antecedent must be the subject, a result of
passive formation. Here, explicit index matching is required. Otherwise, the D-structure in
(17a) could generate the sentence in (17b), in which movement obeys subjacency and the trace
is properly governed, but the trace is not bound in its governing category, violating Condition
A:

(17)  a.  [ e be likely [ Mary [-TNS,+PERF] see John ]]

b.  John_i is likely Mary to have seen t_j

The statement for this condition is given below, where INFL represents the governing
category for the trace in object position:

(18)  condition_A(L, [[infl, _], NP1, [_, _],
    [[v, _], _, [_, _ArgS]]]) :-
    member(NP2, ArgS),
    is_nptrace(NP2),
    !,
    index(NP1, I), index(NP2, J), I == J.

In (18), each argument of the verb is checked for the presence of an NP-trace. If none is
found, Condition A succeeds by default. Otherwise the index of the trace must match the
index of the clausal subject.

As these are the only two possibilities for an NP-trace to occur, a further check is
made to verify that no other EC in the structure has the features of an NP-trace, i.e.
As for Condition C, a violation will occur if a variable is bound to a c-commanding phrase in an A-position. As with Condition A, object positions are excluded as possible antecedents by the Projection Principle, leaving subjects as the only possible candidates. A violation could arise in the following instance, where (19a) gives the D-structure and (19b) a possible derivation:

(19) a. 
\[ e \text{ seem } [ e \left[ \text{ who like Madonna } \right] ] \]

b. *[ Who\text{ who} [ t_i \left[ t_i \text{ likes Madonna } \right] ]

c. [ Who\text{ who} [ does it seem \left[ t_i \left[ t_i \text{ likes Madonna } \right] \right] ]]

In (b), who moves to COMP by wh-movement, and in the next cycle to the Ā-subject of the matrix clause. The variable \( t_i \) in the embedded subject position is now bound to the matrix subject, violating Condition C. Compare this with the grammatical sentence in (c), where the embedded subject trace is bound to who in COMP, an Ā-position, and therefore not in violation of the binding conditions.

The relevant statements that constitute the basic formulation of Condition C are given below. As with the implementation of the other well-formedness conditions, other statements not shown are also involved in recursively searching clausal constituents for binding condition violations.

(20) condition_C(L, [[comp,],_,,[_,_]], [[infl,],NP,[_,_|vp]]) :-
var_free(NP, VP).

var_free(NP, Args) :- index(NP,I), !, free(I, VP).
var_free(_,_).

free(I, [H|T]) :- free_arg(I, H),
free_arg2(I, H),
free(I, T).
free(_,[]).

free_arg(I, X) :- not (is_variable(X),
index(X, J),
I == J).

The var_free statement determines if the subject has an index; if so, then any embedded variable must not be bound with the same index. The free statement makes sure that each
argument of the verb does not share the index of the subject (checked by `free_arg`), and that no embedded phrase is bound to the subject (checked by `free_arg2`). The `free_arg` statement will succeed unless the phrase under inspection is a variable whose index matches that of the subject.

5.5 Empty Category Principle

The ECP (or Extended ECP (Chomsky 1982)) states the conditions on empty categories (ECs):

(21) Extended Empty Category Principle
   (i) An EC is a trace if and only if it is properly governed
   (ii) An EC is PRO if and only if it is ungoverned

NP-traces and `wh`-traces must be properly governed by (i), where proper government is a more restricted notion of government than that presented in Section 2.2.4. The EC `pro` is not subject to the ECP but rather depends on Condition B of the binding conditions, being a pronominal, and on its identifiability; that is, if deleted its identification must be recoverable. In Spanish, the verbal inflection is generally sufficient to identify the missing pronoun. (Note that this applies to `pro` in subject position, not object `pro` which identifies with a clitic.) The distribution of the remaining EC, PRO, is described by (ii) of the ECP, which restricts it to the subject of infinitival clauses.

Certain configurations are classed as proper governors, possibly subject to parametric variation. Thus, for English, `[+TNS]` governs the subject but does not properly govern it. For Spanish, `[+TNS]` is included in the set of proper governors but prepositions are not. The "core grammar" components of the system contain the following Prolog unit clauses for defining the proper governors:

---

16 The inclusion of `[+TNS]` as a proper governor for languages such as Spanish was proposed by Chomsky (1981) but later rejected. It is included here as a proper governor since it simplifies the implementation for the effects we wish to achieve.
In addition to the above categories, a *wh*-phrase or its trace in COMP is implicitly included as a proper governor.

With these descriptions of proper government, the ECP is implemented by enumerating all the structural configurations in which an EC may occur and verifying that the position meets the ECP. These are listed below, and include (1) proper government by an S-deleting predicate, (2) proper government by a *wh*-element (overt or trace) in COMP, (3) proper government by [+TNS] (for Spanish), (4) proper government by V or P of its arguments, and (5) proper government by a lexical head of any adjuncts. The latter case arises, for example, in *wh*-movement of a non-argument, as in (23):

(23) At which dance did Kevin meet Marlene t?

The *wh*-phrase is not subcategorized by *meet*, but the trace is nevertheless properly governed.

The principal statements of the ECP are the following:

(24) ecp(L, [[X,_,],[_,XW, [[comp,_,]e,[e,e, [[infl,_,]NP,[_]]]]]]):-
    not lexical(NP),
    is_sdeleter(L, XW, X),
    !, not is_PRO(NP).

ecp(L, [[comp,_,],WH,[e,e, [[infl,_,]NP,[_]]]]):-
    index(WH,_),
    !, not is_PRO(NP).

ecp(L, [[infl,FI],NP,_]):-
    not lexical(NP),
    not member(tns(-), FI),
    !,
    proper_governor(L, infl).

---

17 This is perhaps better described as antecedent government, not lexical government (Lasnik and Saito 1984).
ecp(L, [[X, _], _, [_, _|Args]]) :-
  member(XP, Args),
  is_trace(XP),
  !,
  proper_governor(L, X).
ecp(L, [[X, _], _, [_, _|PostAdj]]) :-
  member(XP, PostAdj),
  is_trace(XP),
  !,
  proper_governor(L, X).
The overall purpose of the system is to read grammatical sentences in a source language, convert them to D-structures, translate the D-structures into a target language, and then generate grammatical sentences in the target language, relying on the principles of transformational grammar. Since many transformations are optional, the system will perform all possible transformations on a single D-structure, producing an output sentence for each option. Likewise, a sentence in the source language may have multiple translations, and for each translation the complete set of transformational options are applied.

The high-level operation of the system is defined by the following Prolog program:

(1) \[
\text{model} \leftarrow \text{readinput}(SL, TL, PF1), \\
\text{gb}(SL, PF1, TL, PF2), \\
\text{printout}(PF2). \\
\]

\[
\text{gb}(SL, PF1, TL, PF2) \leftarrow \\
\text{morph}(SL, PF1, PF1a), \\
\text{parse}(SL, PF1a, Sstructure1), \\
\text{rtransformation}(SL, Sstructure1, Dstructure1), \\
\text{translation}(SL, Dstructure1, TL, Dstructure2), \\
\text{transformation}(TL, Dstructure2, Sstructure2), \\
\text{pf\_generation}(TL, Sstructure2, PF2), \\
\]

Following input of the source language, target language, and source sentence, the gb predicate derives the target sentence, which is then printed. The six clauses of gb have the following interpretation:

1. Perform a morphological analysis on the source sentence PF1, converting it to a list of lexical tokens PF1a suitable for parsing.
2. Parse the list of tokens, creating an S-structure representation.
3. Perform reverse-transformations on the S-structure, converting it to D-structure representation.
4. Translate the source language D-structure into a D-structure in the target language.
5. Perform transformations on the D-structure to produce an S-structure. It is in this stage that the well-formedness conditions apply to filter out ungrammatical S-structures.

6. Generate a list of tokens PF2 from the S-structure.

The implementation uses the built-in Prolog predicate `bagof` in step 4 to collect all possible translations, which are passed one at a time to the transformation routine. The `bagof` predicate is again used in step 5 to produce multiple transformations on each of the translations.

The eight clauses of the model are described below.

6.1 Sentence Input

The three-argument `readinput` clause prompts for the source language, target language, and sentence in the source language. The sentence is represented as a list of words, or tokens, separated by commas. Final punctuation is also represented as a token. All upper-case characters are converted to lower-case.

The procedure for reading each character, combining characters into words, and stringing words into a list, is taken directly from Clocksin and Mellish (1981:87–88).

The following illustrates a typical dialogue:

Enter source language: Spanish.
Enter target language: English.
Enter sentence in Spanish: Quien dijiste que salió?

As a result of the above dialogue, the `readinput` clause would have its three arguments instantiated as follows:

(2) \[ \text{readinput(spanish,english,[quien,dijiste,que,salió,?])}. \]

(Note that diacritical marks appear over the letters in the examples here and elsewhere to improve readability. In practice, given the English keyboards available, diacritics immediately follow the letters over which they are to appear, as can be seen in the examples in Appendix A.)
6.2 Morphological Analysis

The morph clause analyzes the list provided by readinput and updates the dictionary if necessary. It also analyzes contractions and collocations, forming a new list of tokens to be parsed.

The operation of morph is defined by the following three clauses:

\[(3)\]
\[
\text{morph(SL, Surface, Newsurface) :-}
\]
\[
\text{contract(SL, Surface, Surfacel),}
\]
\[
\text{collocate(SL, Surfacel, Newsurface),}
\]
\[
\text{morphwords(SL, Newsurface).}
\]

The first clause, contract, uses the contraction table (cf. 3.1.3) to split a contraction into its constituent words. It constructs an output list of tokens identical to the input list except that any contractions are expanded, or "de-contracted". For example, the Spanish sentence *Yo pongo los libros al lado del radio* 'I put the books beside the radio' would evaluate to the following:

\[(4)\]
\[
\text{contract(spanish,}
\]
\[
[yo,pongo,los,libros,al,lado,del,radio,.],
\]
\[
[yo,pongo,los,libros,a,el,lado,de,el,radio,.]).
\]

The second clause, collocate, forms collocations on the words supplied by the contract routine. It uses the collocation table (cf. 3.1.4) to generate a new list of tokens that include collocated entries. Using the same example as above, the collocate clause would evaluate to:

\[(5)\]
\[
\text{collocate(spanish,}
\]
\[
[yo,pongo,los,libros,a,el,lado,de,el,radio,.],
\]
\[
[yo,pongo,los,libros,al_lado_de,el,radio,.]).
\]

The third clause, morphwords, performs a morphological analysis on each token in the list. Three options apply: (1) the token is already in the dictionary, in which case processing continues with the next token; (2) a suffix analysis is undertaken on the token to see if it is an inflected form of an existing lexical entry; or (3) a prompt is issued on the computer terminal to allow the addition of the word to the dictionary.

The suffix analysis uses the suffix table (cf. 3.1.2) to determine if the word is a regular inflection of a root word. For each entry in the suffix table, an attempt is made to split the word into a stem segment followed by the suffix, with letters appended to the stem
if present in the suffix table. If the (appended) stem is in the dictionary, then the current
token is added with features specified for in the table. If not, the next entry in the suffix
table is selected, continuing in this manner until a match is found or the table is exhausted.
Once an inflected form is added to the dictionary it remains until the computer session is
terminated.

An example of suffix analysis is given below for the Spanish word *viviendo* 'living'. A
segment of the suffix table includes the following entries:

(6)

```
suffixtable(spanish,ando,ar,v,[prog(+)]).
suffixtable(spanish,iendo,er,v,[prog(+)]).
suffixtable(spanish,iendo,ir,v,[prog(+)]).
```

The root entry in the dictionary contains:

(7)

```
dict(spanish,vivir,v,[english(live),...]).
```

The first entry in the suffix table, *-ando*, fails to match with the ending of *viviendo*. The
second entry succeeds, but when the stem *viv* is appended with *-er*, the result *viver* is not
listed in the dictionary, so the next entry in the table is chosen. This also succeeds, and since
the result *vivir* does exist, the dictionary is updated with:

(8)

```
dict(spanish,viviendo,v,[f(prog(+)),root(vivir)]).
```

If the suffix analysis fails, an opportunity is presented for including the word in the
dictionary, either permanently or just for the duration of the current session. The lexical
category and relevant features are requested, and the dictionary is appropriately updated.

6.3 Parsing

The parse routine converts the string of tokens into a phrase marker representing the
S-structure of a clause. The clause is represented by a COMP phrase, and parsed by calling
the x2 routine with X=comp:

(9)

```
parse(L, PF, Sstructure) :- x2(L, comp, Sstructure, PF, []).
```

COMP subcategorizes an INFL phrase, as indicated in the lexical entry for complementizers, or
by default, and INFL subcategorizes a VP. So parsing continues from one phrase to another as directed by the subcategorization information for each lexical head. Parsing will succeed as long as the subcategorization requirements are met, with ECs inserted if necessary. The main purpose of the parsing component in this model, then, is simply to satisfy the Extended Projection Principle without regard to other rules or principles of grammar, such as subject–verb agreement.

The following is the instantiation of the parse statement after parsing the sentence: *Who did you say you saw?*

(10)    \[
    \text{parse(english,}
    \begin{align*}
    \text{[who,did,you,say,you,saw,?],} \\
    \text{[[comp,[]], [[n,[]],e,[e,who]], [e,e,} \\
    \text{[[infl,[]], did,} \\
    \text{[[infl,[]], [[n,[]],e,[e,you]], [e,e,} \\
    \text{[[v,[]],e,[e,say,} \\
    \text{[[comp,[]],e,[e,e,} \\
    \text{[[infl,[]], [[n,[]],e,[e,you]], [e,e,} \\
    \text{[[v,[]],e,[e,saw,} \\
    \text{[[n,[]],e]]], [mode,ques]]])]]])].}
    \end{align*}
\]

6.4 Reverse Transformations

The function of rtransformation is to convert the S-structure to D-structure. As D-structure is a pure representation of the \(\theta\)-marking properties of the clause, a function of rtransformation is to return all lexical phrases in \(\bar{\theta}\) positions to their base-generated \(\theta\) positions. A second function is to determine the feature specifications of each phrase by percolating and merging features from lexical constituents, validating feature agreement in the process. The other functions reverse the effects of the various other transformations that are required in forming well-formed surface structures.

For the most part, reverse-transformations are, in fact, transformations performed in reverse. That is, if a transformation inserts a lexical item, such as pleonastic *it*, the reverse-transformation will remove it. If Do Support is required before inversion takes place, the reverse-transformation sequence is to undo the inversion, then remove the do-formative.
Consequently, the components that perform transformations and reverse-transformations call upon many of the same routines but with interchanged parameters. The transformational cycle is also reversed in rtransformation, so that all reverse-transformations are performed on the matrix clause first, then on each embedded clause in descending order.

As a recursive function, rtransformation calls all the reverse-transformations for one cycle. It operates functionally on various stages of S-structure, with each subroutine converting the previous S-structure to a modified S-structure, passing it to the next subroutine, until the D-structure representation is obtained for that cycle. Then the process repeats on the embedded clause. The operation of rtransformation is given by the following:

\[
\text{rtransformation}(L, \text{COMP1}, \text{COMP11}) : - \\
\text{rinvversion}(L, \text{COMP1}, \text{COMP2}), \\
\text{percolate}(L, \text{COMP2}, \text{COMP3}), \\
\text{rmove_affix}(L, \text{COMP3}, \text{COMP4}), \\
\text{theta_extraction}(\text{COMP4}, \text{COMP5}), \\
\text{do_support}(L, \ldots, \text{COMP6}, \text{COMP5}), \\
\text{it_insertion}(L, \text{COMP7}, \text{COMP6}), \\
\text{null_subject}(L, \text{COMP8}, \text{COMP7}), \\
\text{rmove_alpha}(\text{COMP8}, \text{COMP9}), \\
\text{wh_feature_extraction}(\text{COMP9}, \text{COMP10}), \\
\text{rtransform_lower}(L, \text{COMP10}, \text{COMP11}).
\]

Each function is described below, although some, such as Do Support, It Insertion, and Null Subject, are simply the reverse of the transformations described in Chapter 4. If the structural configuration for these transformations is present in the surface structure, the corresponding structural change occurs. Otherwise, the structure remains unchanged. Notice that the parameters are reversed in these three transformations.

6.4.1 Rinversion

The rinversion clause for English reverses the effect of SAI, then reverses Have/Be Raising. If an adjunct has been parsed, SAI will return the contents of the adjunct to the base INFL head position. Then if the head of INFL is a perfective or progressive auxiliary, it bumps the current auxiliary to second auxiliary position, and the auxiliary in INFL is moved into the primary auxiliary position, leaving the head of INFL empty.
The rinversion statement for English is defined as:

\[(12) \quad \text{rinversion(english, COMP1, COMP3)} :-
\quad \text{sai(\_\_, COMP2, COMP1)},
\quad \text{havebe_raising(COMP3, COMP2)}.\]

The parameters are reversed in each clause, with the input of the reverse transformation being the same as the output of the "forward" transformation.

For Spanish, rinversion is defined in terms of the verb-preposing transformation. If an adjunct has been parsed, its contents are restored to their base position according to the pattern defined by the surface structure and adjunct pair, as described in Section 4.4.4. Otherwise the structure remains unchanged. The clauses that define rinversion for Spanish are the following:

\[(13) \quad \text{rinversion(spanish, COMP1, COMP2)} :-
\quad \text{v_preposing(COMP2, COMP1)},
\quad !.\]

The v_preposing clause simply forms the four-element lists from the base structure, surface structure and adjunct, for use by the pattern-matching unit clauses in v_prepose (cf. 4.4.4). The arguments to v_preposing are given in reverse order.

### 6.4.2 Percolate

The node position of each phrasal category contains the name of the phrase and a list of features for that phrase. Initially, the list is null. The function of percolate is to assign to the list the merged set of features of the head and specifier, which must of course agree. Percolation is carried out on all phrasal categories except VP, which has its own "percolation" routine, rmove_affix.

Percolation for COMP, INFL, PP, and AP is trivial, since in the latter two cases there are no specifiers with features to percolate (in this model), and in the former two the head contributes no features that would clash with the specifier. The primary function of percolate, then, is to establish agreement between head nouns and articles with respect to person, number, and, for Spanish, gender. Subject-verb agreement occurs within rmove_affix.
Another function of percolate is to replace inflected words (other than verbs) with their root form in preparation for translation.

An illustration of percolate is shown below for the sentence: *I saw the ships.*

(14) \[
\text{percolate(english,} \\
\quad \text{[[comp,[]],e,[e,e,} \\
\quad \text{[[infl,[]], [[n,[]],e,[e,i]], [e,e,} \\
\quad \text{[[v,[]],e,[e,saw,} \\
\quad \text{[[n,[]],the,[e,ships]]]]],} \\
\quad \text{[mode,decl]]],} \\
\quad \text{[[comp,[]],e,[e,e,} \\
\quad \text{[[infl,[per(l),pl(-)]]],} \\
\quad \text{[[n,[per(l),pl(-)],[e,i]], [e,e,} \\
\quad \text{[[v,[]],e,[e,saw,} \\
\quad \text{[[n,[per(3),pl(+)]],the,[e,ship]]]]]],} \\
\quad \text{[mode,decl]])}.\]

6.4.3 Rmove-Affix

The rmove_affix clause moves tense information from the VP to INFL, undoing the affect of affix-hopping. After the first stage of rmove_affix has applied, the verb will be in its uninflected form, and the tense and agreement features of the verb will have percolated to the V\textsuperscript{2} level. A final step guarantees subject–verb agreement by merging the features of V\textsuperscript{2} with those in INFL\textsuperscript{2}.

Tense information includes the tense itself (past, present or future), aspect (perfective or progressive), and voice (passive or, by default, active). These features are determined from the morphology of the verbal elements. A clause within rmove_affix lists all the combinations of auxiliaries with main verb and extracts the relevant features. It also ensures that the appropriate affixes were present for each aspect. Thus a perfective auxiliary requires the next verbal element to have the feature [+PERF]; a progressive auxiliary requires a following verbal with [+PROG]; and passive requires perfective morphology, i.e. [+PERF]. One consequence of the passive voice is that the subject is not θ-marked. Therefore if passive morphology is detected, the feature [−THETA] is added to the feature list.

An additional function of rmove_affix is to assign a default tense if none is indicated. Since Spanish exhibits inflection for every tensed form, the absence of an explicit feature
means that the first verbal element is infinitival. In English, absence of tense indicates present tense, with person and number features taken from the subject.

6.4.4 Theta-Extraction

Non-\(\theta\)-marked subjects of clauses will be returned to their base position by the \texttt{rmove_alpha} function. If a predicate contains the \([-\text{THETA}]\) feature, the reverse movement will be performed. In anticipation of this action, the \([-\text{THETA}]\) feature is transferred up to the INFL feature set. The reverse movement will then be triggered simply by inspecting the features in INFL.

For verbs with this feature (e.g. \textit{seem}), the transfer will be simultaneous with the transfer of tense and agreement features to INFL during \texttt{rmove_affix}. However for adjectives (e.g. \textit{probable}), the feature must be explicitly searched for and if found transferred to INFL. This is because the AP structure is embedded within the VP as a subcategorized phrase, and \texttt{rmove_affix} only moves features from the VP.

The \texttt{theta_extraction} routine is defined below. The first parameter is the current S-structure and the second is the resulting base structure:

\begin{equation}
\texttt{theta_extraction([[comp,FC],WH,[e,COMP,}
\[\texttt{[infl,FI],NP,[e,INFL,}
\[\texttt{[[v,FV],P1,[P2,V,}
\[\texttt{[[a,FA],e,AP]]|PostV]]]|PostC}],
\[\texttt{[[comp,FC],WH,[e,COMP,}
\[\texttt{[infl,[theta(-)|FI]],NP,[e,INFL,}
\[\texttt{[[v,FV],P1,[P2,V,}
\[\texttt{[[a,FA],e,AP]]|PostV]]]|PostC]) :-}
\texttt{member(theta(-), FA), !.}}
\end{equation}

6.4.5 Do-Support

The \texttt{do_support} function deletes the "quasi-modal" \textit{do} from the head of INFL, as described in Section 4.4.3. As a reverse-transformation, the parameters are listed in reverse order.
6.4.6 It-Insertion

The *it_insertion* function replaces *it* with an empty NP if the [-THETA] feature is present in INFL, as described in Section 4.5.

6.4.7 Null-Subject

The *null_subject* function has no effect in English as a reverse-transformation. For Spanish, an attempt will be made to lexicalize a null subject. If the verbal inflection indicates first or second person, the subject is unambiguously determined, and the EC is appropriately replaced. A third person inflection may indicate a variable, so a null subject in this case remains an EC, pending later analysis. See Section 4.3.

6.4.8 Rmove-Alpha

By the \(\theta\)-Criterion and the Projection Principle, movement may only be to a \(\overline{\theta}\) position. For the cases of Move-\(\alpha\) under consideration in this model, the only \(\overline{\theta}\) positions are COMP and the subject of non-\(\theta\)-marking predicates. The *rmove_alpha* clause consists of (1) movement from COMP, followed by (2) movement from subject:

\[
\begin{align*}
\text{rmove}_\alpha(\text{COMPl}, \text{COMP3}) & : - \\
& \text{rmove}_\text{comp}(\text{COMPl}, \text{COMP2}), \\
& \text{rmove}_\text{subj}(\text{COMP2}, \text{COMP3}).
\end{align*}
\]

For movement from COMP, the phrase must be either an NP or PP with the feature \([+WH]\). Possible destinations are (1) an empty subcategorized argument, (2) an empty NP object of a preposition, (3) an empty subject, or (4) an embedded COMP. The options are taken in that order. Options (1) and (2), and (3) for English, are required in order to satisfy the \(\theta\)-Criterion. ECs in these positions must be filled. For option (2), the PP may be a subcategorized argument or an adjunct. Option (3) poses difficulty for Spanish because of the null subject parameter. Substituting a *w*/NP for an EC subject is premature if an embedded obligatory position remains empty. Unfortunately the current system does not have adequate backtracking ability to recover from this situation, so input sentences with this characteristic are
avoided. In the case that a sentence has two null subjects (e.g. *Quién piensa que salió?* 'Who thinks (he/she) left?'), the first will be replaced by the NP in COMP. For both English and Spanish, the *wh*-NP must agree in features with the verb, which for English represents a grammaticality check, while for Spanish it helps to further distinguish between a null subject and a variable subject. If none of options (1–3) are available, then for NPs, option (4) is obligatory (assuming that NPs must always receive a θ-role). For PPs, option (4) will be selected unless no embedded COMP is available, in which case an adjunct position to the VP will be created to contain the *wh*-PP. (NPs may also have PP adjuncts, but this possibility is ignored here.)

Movement from subject takes place if a lexical NP occupies a non-θ-marked subject, indicated by the [−THETA] feature. If the clause is marked for passive, then movement must be to an empty object. Otherwise an empty lower subject must be available as a destination.

6.4.9 WH-Feature-Extraction

The purpose of this function is to merge a [+WH] feature into the feature list for a PP if its NP object contains the [+WH] feature. The need for this arises in sentences such as:

(17) a. To whom did they give glass beads?
    b. Who(m) did they give glass beads to?

In both cases, the NP object *who(m)* has the feature [+WH] but the preposition *to* does not. In order for the prepositional phrase *to whom* to move to COMP, it must have the [+WH] feature. The `wh_feature_extraction` routine finds and extracts the features of the NP objects of all PPs. If the features of NP contain [+WH], then [+WH] is merged with the features of PP. The part of the function that performs this action is shown below:

(18) `wh_feature_extraction([[p,F],e,[e,P,NP]],
    [[p,[wh(+)|F]],e,[e,P,NP]]) :-
    features(NP, FNP),
    member(wh(+), FNP),
    !.`
6.5 Translation

Translation is a complex process that involves many factors, including problems of syntax, semantics, intensionality, causality, etc. Accurate high-quality translation of general text by machine is not yet available, and is an advanced skill even at the human level.

Our interest in this system is in studying the applicability of a model of UG, augmented by the effects of parametric variation, to achieve grammatically correct translations. Subtleties of meaning are not of interest, but rather the comparative analyses of sentences that express essentially identical propositions. Therefore we restrict attention to achieving sentence translations based on the semantic equivalence between lexical elements. For example, the English verb *put* is semantically equivalent to the Spanish *poner*; both subcategorize and assign \( \theta \)-roles to identical elements. We disregard other possible translations (e.g. *meter*, *echar*, *colocar*) as well as the myriad other expressions which use *put* (e.g. *hard put to do something*, *put up with something*, *put to sleep*, etc.). Similarly for nouns and adjectives where a single translation suffices; an English *car* translates to a Spanish *coche*, ignoring *carro*, *auto*, etc., and the adjective *certain* simply translates to *cierto*, ignoring *seguro*, etc.

Multiple translations are provided for some lexical items, especially prepositions, although in general only one will be correct in a given context. As it is often very difficult to distinguish by automatic means the correct choice, this system indiscriminately uses all the translations provided. Typical examples are the Spanish preposition *de* which means *of* or *from*, the English *for* which translates to *por* or *para*, and the Spanish possessive pronoun *su* which means *his*, *her*, *its*, *their*, and *your* (formal form). Consequently, some anomalous translations result from input sentences such as the following: *Juan salió con su amigo de España.*

(19)  
\begin{align*}
\text{a. } & \text{John left with his friend from Spain.} \\
\text{b. } & \text{John left with her friend from Spain.} \\
\text{c. } & \text{John left with its friend from Spain.} \\
\text{d. } & \text{John left with their friend from Spain.} \\
\text{e. } & \text{John left with your friend from Spain.} \\
\text{f. } & \text{John left with his friend of Spain.} \\
\text{g. } & \text{John left with her friend of Spain.} \\
\text{h. } & \text{John left with its friend of Spain.} \\
\text{i. } & \text{John left with their friend of Spain.} \\
\text{j. } & \text{John left with your friend of Spain.}
\end{align*}
Two methods are provided for translating phrases and idioms, since literal translations are almost always totally inappropriate. One is by using the collocation table, as mentioned in Section 3.1.4. The other is by storing the idiomatic expression in the lexical entry of the leading word. A typical example is *kick the bucket*, where *kick* contains as part of its lexical definition the expression *the bucket*. Whenever *kick* is encountered in the input sentence, a routine check is made first to see if an idiomatic expression can be matched with the subsequent wording.

Examples of dictionary entries with multiple translations and idiomatic expressions are shown below:

\[(20)\]  
\[
dict(\text{spanish, de, p, [english([of,from])]}). \\
dict(\text{english, kick, v, [expr([[the,bucket],spanish([morir, expr([estirar,la,pata])]),subcat(n),spanish(patear)])].} \\
\]  
In the entry for *kick* above, the first 'expr' term contains a list of two items; the first is a list of words that form the idiomatic usage, and the second is the Spanish equivalent, shown here as a list of two possible translations. One is the simple word *morir* 'die', the second is itself an idiomatic expression, *estirar la pata*. If *kick* is not being used in its idiomatic form, then the Spanish *patear* is the translation, subcategorizing for an NP.

Translation takes place between D-structures, since D-structures directly represent the thematic relations in a sentence. In fact, the primary motivation for recreating D-structure from S-structure is just to simplify the translation stage, which would otherwise involve an analysis of movement chains.

The strategy is to translate the sentence phrase by phrase, independently of each other, starting at the matrix COMP² level. The head of each phrase is translated first and then, for nouns, inflected according to the feature specifications of the phrase. (Verbal inflections are applied later by the Move Affix transformation.) Specifiers are translated next and inflected to match the head. All subcategorized phrases and adjunct phrases are translated similarly.

The translation function, recursively called for each maximal phrase structure, is shown below. The first argument to *translate* is the source language, the second argument is the list
structure of the sentence, the third argument is the target language, and the fourth is the translated list structure:

(21) \text{translate} (SL, [[C,F1],Spec1,HD1|Post1],
    TL, [[C,F2],Spec2,HD2|Post2]) :-
    transl\_x1(SL,C,F1,HD1, TL,F2,HD2),
    transl\_spec(SL,C,Spec1, TL,F2,Spec2),
    transl\_comp(SL,Post1, TL,Post2).

    transl\_x1(SL,C,F1, [Pre,HD1|Complement1],
    TL, F2, [Pre,HD2|Complement2]) :-
    transl\_head(SL,C,F1,HD1, TL,F2,HD2),
    transl\_comp(SL,Complement1, TL,Complement2).

The example in (23) illustrates how the list structures would appear in translating sentence (22a) to (22b). Unessential detail has been omitted, and phrases are indented to improve clarity:

(22) a. How many books had Mary put beside her bed?
    b. Cuántos libros había puesto María al lado de su cama?
(23)  
\[COMP\]
\[INFL:TNS(past), + PERF, PER(3), - PL e \]
\[NP:PER(3), - PL maria \]
\[VP:TNS(past), + PERF put \]
\[NP:PER(3), + PL how many book \]
\[PP beside \]
\[NP:PER(3), - PL her bed ]]
\[MODE:ques ]

\[COMP\]
\[INFL:TNS(past), + PERF, PER(3), - PL e \]
\[NP:PER(3), - PL maria \]
\[VP:TNS(past), + PERF poner \]
\[NP:PER(3), + PL cuantos libros \]
\[PP al lado de \]
\[NP:PER(3), - PL su cama ]]
\[MODE:ques ]

6.6 Transformations

The transformational stage transforms each translated D-structure into a set (a "bag" in Prolog terminology) of S-structures, using the transformations described in Chapter 4. Each S-structure is passed through the set of well-formedness conditions described in Chapter 5. Only the S-structures that satisfy all the conditions are displayed as output.

The high level transformational function is shown below:
(24) \texttt{transformation(L, Dstructure, Sstructure) :-}
    transform\texttt{(L, matrix(+), Dstructure, Sstructure),}
    dbl\_comp\_filter(Sstructure),
    case\_filter(L, Sstructure),
    binding\_conditions(L, Structure),
    ecp(L, Sstructure).}

The D-structure is transformed into S-structure by the \texttt{transform} function. The 'matrix(+)' parameter indicates that a matrix clause is being passed initially, since \texttt{transform} is recursively called on each embedded clause. Following the transformation, the well-formedness of the S-structure is validated against the Doubly-filled COMP Filter, the Case Filter, the Binding Conditions, and the ECP. Failure to pass any of the conditions results in automatic backtracking into \texttt{transform} to try another transformational option.

The \texttt{transform} function, shown below, performs the transformational cycle on each clause. The first statement, \texttt{transform\_lower}, recursively descends the list structure looking for an embedded COMP\textsuperscript{2} phrase. If it finds one it calls \texttt{transform} on that clause, which itself calls \texttt{transform\_lower} as its first step. In this way the structure is completely transformed beginning with the most deeply embedded clause and ending with the matrix clause.

(25) \texttt{transform(L, Matrix, Mode, COMPl, COMPl1) :-}
    transform\_lower(L, matrix(-), Mode, COMPl, COMPl2),
    set\_tns(Matrix, COMPl2, COMPl3),
    move\_alpha(L, COMPl3, COMPl4),
    it\_insertion(L, COMPl4, COMPl5),
    null\_subject(L, COMPl5, COMPl6),
    do\_support(L, Matrix, Mode, COMPl6, COMPl7),
    move\_affix(L, COMPl7, COMPl8),
    comp\_insertion(L, Matrix, Mode, COMPl8, COMPl9),
    modal\_insertion(L, Matrix, COMPl9, COMPl10),
    inversion(L, Matrix, Mode, COMPl10, COMPl11).

The 'Mode' variable unifies with the mode of the clause, as either declarative or question. It is returned by \texttt{transform\_lower} since the mode is attached as an adjunct to the lowest COMP\textsuperscript{2} phrase. The mode influences Do Support and Inversion, in that a question mode will trigger these transformations.

Each of the remaining functions of \texttt{transform} are described below.
6.6.1 Set-TNS

Each clause within the D-structure retains the tense specification of the original source sentence. Assuming that all tenses other than the matrix clause may be either [+TNS] or [-TNS] (ignoring marked lexical exceptions), the purpose of the set_tns function is to alternate each tense specification. During the transformational phase the original tense of each clause is maintained. Then when backtracking occurs, the tense will alternate. If the tense specification is [+TNS] (i.e. one of [TNS(past)], [TNS(pres)], or [TNS(fut)]), it changes to [-TNS], i.e. infinitive. If the original tense is infinitive, then, for the sake of making a reasonable choice blindly, [TNS(pres)] is selected, along with [PER(3)] and [-PL] for agreement features.

The matrix clause remains [+TNS] and does not alternate.

6.6.2 Move-Alpha

While technically Move-a moves "any category anywhere", its operation is limited here to movement to COMP or movement to subject, the latter restricted to $\bar{\theta}$ subjects. Additionally, a is restricted to either NP or PP with the feature [+WH] in the case of movement to COMP, and only NP in the case of movement to subject.

Five move_alpha statements are included to handle the various possibilities. They are summarized below:

(26) a. No movement.
b. NP movement to subject
   - subject must be empty and $\bar{\theta}$
c. NP movement to COMP
   - NP must have the feature [+WH]
d. PP movement to COMP
   - PP must have the feature [+WH]
e. NP movement to subject, then to COMP
   - same conditions as (b) and (c)

A general-purpose move routine is called by move_alpha to do the actual move. The layout of move is as follows:

(27) move(L, Alpha, Level, X, F, BINFL, INFL).

move is provided with the category 'Alpha' (= NP or PP) and the features 'F' to be assigned
to the trace (i.e. [±ANAPHOR,±PRONOUN]), depending on its destination. The routine recursively searches top-down through each phrase within the base INFL structure 'BINFL' until it finds a phrase 'X' of category 'Alpha'. It extracts 'X' from 'BINFL', replacing it with an EC to which are given the features 'F'. The modified structure is returned in 'INFL'. On backtracking, move will eventually find and pass back all c-commanded phrases of category 'Alpha'.

An index is also generated as a Prolog logical variable and added to the features of 'X' and the EC. The variable is never instantiated but retains its Prolog-generated unique value. Only the trace and the antecedent will share the same value.

To implement subjacency, the move function keeps track of the number of bounding nodes encountered. The core grammars each have a definition of the bounding categories for that language, as given in (28) below:

(28)    bounding_cat(english, infl).
        bounding_cat(english, n).

        bounding_cat(spanish, comp).
        bounding_cat(spanish, n).

This explains why the language 'L' is passed as a parameter to move. The 'Level' parameter, initially set to zero by move_alpha, is incremented by one for each bounding category crossed. The function fails when 'Level' reaches two.

The examples given in Section 6.9 illustrate the processing of the move_alpha transformation.

6.6.3 It-Insertion

As described in Section 4.5, an empty θ subject is obligatorily replaced with it, including features [PER(3),-PL]. If the subject position is already filled and/or is θ-marked no action is taken.
6.6.4 Null−Subject

The null_subject transformation for Spanish optionally deletes a lexical subject if it has the feature [+PRONOUN]. English has no null subject option. However in the cases where a Spanish sentence is input with a null subject that does not become lexicalized, the generation of the sentence in English must lexicalize it (assuming it is not PRO, i.e. not in the environment of [−TNS]). Third person English pronouns are selected one at a time as valid options, re-instantiating during backtracking.

6.6.5 Do−Support

Following the description in Section 4.4.3, the matrix clause of a question in English will have do inserted into modal position as long as the subject is lexical, and no other modal or auxiliary is specified. That is, the matrix clause must not be in future tense, have perfect or progressive aspect, or be in the passive voice.

6.6.6 Move−Affix

The function of move_affix is to implement affix−hopping, at the same time generating auxiliaries as indicated by the presence of any aspectual features. The function is defined at the top level as follows:

(29) \[
\text{move\_affix}(L, [[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{BINFL},\text{BVP}]]]|\text{Post}],
[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{INFL},\text{VP}]]]|\text{Post}]) :-
\text{affix\_infl}(L, \text{FI}, \text{BINFL}, \text{FV}, \text{INFL}),
\text{affix\_vp}(L, \text{FI}, \text{BVP}, \text{VP}),
].
\]

\[
\text{affix\_infl}(\_,F,e, F,e).
\]

\[
\text{affix\_infl}(L, F, \text{SourceW}, [\text{tns}(\−)|F], \text{TargetW}) :-
dict(L, \text{SourceW}, \text{infl}, \text{Lex}),
not member(root(_), \text{Lex}),
gen\_verb(L, \text{infl}, F, \text{SourceW}, \text{Lex}, \text{TargetW}).
\]

\[
\text{affix\_vp}(L, \text{FI}, [[[\text{v},\text{FV}],\_,[\_,\text{BV}|\text{Args}]|\text{Post}],
[[\text{v},\text{FV}],\text{PERF},[\text{PROG},V|\text{Args}]|\text{Post}) :-
\text{affix\_to\_perf}(L, \text{FI}, \text{PERF}),
\text{affix\_to\_prog}(L, \text{FI}, \text{PROG}),
\text{affix\_to\_verb}(L, \text{FI}, \text{BV}, V).
\]
The first action is to affix the tense to a modal. The lexical entry of the uninflected modal form is retrieved from the dictionary and then appropriately inflected (in the gen_verb routine, not shown) according to the tense and agreement features in INFL. English modals show minimal inflection, but Spanish verbs acting as modals (poder for can, deber for should, etc.) show normal verbal inflections.

If a modal ends up taking the tense, then the feature [-TNS] is passed to the affix_vp routine to suppress tense inflection of the following verbal elements.

The affix_vp routine generates auxiliary verbs and selects the appropriately inflected main verb. If the INFL features specify [+PERF], then a perfective auxiliary is created and inflected depending on the prevailing tense feature. Similarly with the progressive auxiliary if [+PROG] is specified. Finally the main verb root form is taken from the dictionary and inflected. If the verb specifies irregular forms, then these are inspected first to see if a match can be found. If not then if the verb specifies stem forms, such as the preterite stem tuv- for the Spanish verb tener 'to have', these will be scanned for a match. Otherwise the verb will be automatically inflected by running through the suffix table looking for a matching entry. For English verbs, if the feature specifications include [TNS(pres)] and either [+PL], [PER(2)], or [PER(1)], then the infinitive form is selected, if an irregular form has not already been found.

6.6.7 COMP–Insertion

As described in Section 4.6, the comp_insertion routine for English optionally inserts the that–complementizer into non-infinitival embedded clauses. For Spanish, the insertion of que is obligatory; the "cut" prevents backtracking. The comp_insertion routine is defined as follows:

(30) comp_insertion(english,
     matrix(-),
     [[[comp,PC],WH,[e,e,INFL]|Post],
      [[[comp,PC],WH,[e,that,INFL]|Post]]) :-
    features(INFL, FI),
    not member(tns(-), FI).
comp_insertion(spanish, matrix(-),
   [[comp,FC],WH,[e,e,INFL]|Post],
   [[comp,FC],WH,[e,que,INFL]|Post]) :-
features(INFL, FI),
not member(tns(-), FI), !.

comp_insertion(_,_,COMP, COMP).

6.6.8 Modal-Insertion

In English, the infinitive marker to is inserted into modal position if the INFL features contain [-TNS]. If future tense is indicated, then will is inserted.

No modal insertion occurs in Spanish.

6.6.9 Inversion

Inversion in English involves the transformations of Have/Be Raising (if relevant) followed by SAI. The code for inversion is the following:

(31) inversion(english, Matrix, Mode, COMPl, COMP3) :-
    havebe_raising(COMPl, COMP2),
    sai(Matrix, Mode, COMP2, COMP3).

Have/Be Raising occurs only if an auxiliary is available for raising, and SAI occurs at the matrix level for Mode=ques.

For Spanish, inversion involves the Verb Fronting transformation, obligatory for wh-phrases in any COMP node, and also enforced at the matrix level for Mode=ques. The code for Spanish inversion is the following:

(32) inversion(spanish,_,_, BaseCOMP, SurfaceCOMP) :-
    member([comp,FC], BaseCOMP),
    member(wh(+), FC), !,
    v_preposing(BaseCOMP, SurfaceCOMP).

inversion(spanish, matrix(+), ques, BaseCOMP, SurfaceCOMP) :-
    !,
    v_preposing(BaseCOMP, SurfaceCOMP).

inversion(spanish,_,_,COMP, COMP).
6.7 PF-Generation

As shown by the diagram in Section 2.1, S-structure maps directly to PF and LF. Here, we consider the mapping to PF, which is for the most part merely a stringing out of the terminal nodes of the S-structure tree into a linear list form. However, in a speech synthesis system this component would take on major proportions.

The PF-generation process is defined by the following Prolog statement:

(33) \( pf\_generate(L, S\text{-}structure, Surface) :- \)

\( \text{readleaves}(S\text{-}structure, \text{List}1), \)
\( \text{delete}_e(\text{List}1, \text{List}2), \)
\( \text{collocate}(L, \text{List}3, \text{List}2), \)
\( \text{contract}(L, \text{Surface}, \text{List}3). \)

The head of the clause states that for language L, 'Sstructure' converts to 'Surface' form as a result of passing 'Sstructure' through the four clauses. The first, \text{readleaves}, simply traverses the tree in a depth-first left-to-right fashion, putting each terminal symbol into a list, 'List1'. The second clause, \text{delete}_e, removes all occurrences of the empty marker "e" from the list. This is included as a separate clause, since the adjunct term created during Verb Fronting may contain "e"s; the contents of the adjunct are not analyzed within \text{readleaves}. The \text{collocate} clause is identical to that used for forming collocations during morphological analysis (cf. 6.2), but with the parameters reversed. The same applies to the fourth clause, \text{contract}, which forms contractions. The result is a list of tokens in surface form, suitable for display.

6.8 Sentence Output

The final stage of processing is displaying the list of tokens on the output device. Each token in the list is printed, separated by a blank character. The first character of the list is capitalized, and final punctuation is printed depending on the value of the last token in the list: 'decl' results in a period being displayed; 'ques' converts to the question mark.
6.9 Examples

Two examples are presented in this section. The first illustrates the transformational phase from D-structure to one possible S-structure. The other illustrates how the well-formedness conditions restrict the set of generated S-structures to only grammatically correct ones.

6.9.1 Transformational Phase

In this example, a D-structure passes through the various transformations on its way to S-structure. For each applicable transformation, the resulting phrase structure is indicated. The Move-α transformations shown are only one of its many optional applications, including no movement. The COMP-Insertion step shows that-insertion, an optional transformation.

Example: \( \textit{What did John say that Mary saw?} \)

1. D-structure
\[
[e \left[ S_1 \text{ John } [ \text{ say } e \left[ S_2 \text{ Mary } [ \text{ see } [ \text{ what } ] ] ] ] \right] ] ]
\]

2. \( S_2 \) Cycle - Move-α
\[
[e \left[ S_1 \text{ John } [ \text{ say } e \left[ S_2 \text{ what } [ S_2 \text{ Mary } [ \text{ see } [ t ] ] ] ] \right] ] ]
\]

3. \( S_1 \) Cycle - Move Affix
\[
[e \left[ S_1 \text{ John } [ \text{ say } e \left[ S_2 \text{ what } [ S_2 \text{ Mary } [ \text{ saw } [ t ] ] ] ] \right] ] ]
\]

4. \( S_1 \) Cycle - Move-α
\[
[e \left[ S_1 \text{ what } [ S_1 \text{ John } [ \text{ say } e \left[ S_2 \text{ t } [ S_2 \text{ Mary } [ \text{ saw } [ t ] ] ] ] \right] ] ] ]
\]

5. \( S_1 \) Cycle - Do Support
\[
[e \left[ S_1 \text{ what } [ S_1 \text{ John } [ \text{INFL do } ] [ \text{ say } e \left[ S_2 \text{ t } [ S_2 \text{ Mary } [ \text{ saw } [ t ] ] ] ] ] \right] ] ]
\]
6. S₁ Cycle - Move Affix

\[
[S₁ \text{ what } [S₁ \text{ John } [\text{INFL did } ] [ \text{ say } [S₂ \text{ t } [S₂ \text{ Mary } [ \text{ saw } [ t ]]]]]]]
\]

7. S₁ Cycle - Inversion

\[
[S₁ \text{ what } [S₁ [\text{INFL did } ] [ S₁ \text{ John } [ \text{ say } [S₂ \text{ t } [S₂ \text{ Mary } [ \text{ saw } [ t ]]]]]]]
\]

6.9.2 Well-formedness Phase

In this example, a D-structure is given which then undergoes the series of transformations resulting in an S-structure. Backtracking occurs, deriving alternate S-structures depending on the transformational options taken. For each S-structure generated, its status with respect to the well-formedness conditions is given. In some cases more than one condition disqualifies the structure, although in practice the first failure will cause backtracking. Conceptually the conditions apply in parallel, illustrating the modular nature of the theory; failure in any one component is sufficient to cause termination of the remaining components. Subjacency violations are included among the list of S-structures for illustrative purposes only; they are actually filtered out within the application of Move-\(a\), not as a condition on S-structure.

The sentence in (34) below represents one of the many possible valid S-structures for the D-structure in (35), including those that represent echo questions and yes/no questions. Also, it will be observed that a \(wh\)-phrase in subject position will appear identical to (i.e. have the same PF representation as) a \(wh\)-phrase in COMP binding a variable in subject position, as in items 10 and 18 below.

The S-structures are listed in the order in which they are generated. Only the clause boundaries are shown. Traces are indicated by \(t\), and \(e\) represents the empty complementizer.

(34) Who seems to have left?

(35) \[
[S₁, e [S₁ [\text{TNS} \text{ seem } [S₂, e [S₁ \text{ who } [\text{±TNS, +PERF} \text{ leave } ]]]]]]]
\]
Select [-TNS] in embedded clause:

1. \[ e \ [ \text{does it seem} \ [ e \ [ \text{who to have left} ]] ]\]  
   Fails: Case Filter

2. \[ e \ [ \text{does who}_{i} \text{ seem} \ [ e \ [ t_{i} \text{ to have left} ] ] ]\]

3. \[ \text{who}_{i} \ [ t_{i} \text{ seems} \ [ e \ [ t_{i} \text{ to have left} ] ] ]\]

4. \[ \text{who}_{i} \ [ \text{does it seem} \ [ e \ [ t_{i} \text{ to have left} ] ] ]\]  
   Fails: Subjacency

5. \[ e \ [ \text{does it seem} \ [ \text{who}_{i} \ [ t_{i} \text{ to have left} ] ] ]\]  
   Fails: Case Filter

6. \[ e \ [ \text{does who}_{i} \text{ seem} \ [ t_{i} \ [ t_{i} \text{ to have left} ] ] ]\]  
   Fails: Case Filter, Binding Condition C

7. \[ \text{who}_{i} \ [ t_{i} \text{ seems} \ [ t_{i} \ [ t_{i} \text{ to have left} ] ] ]\]  
   Fails: Case Filter, Binding Condition C

8. \[ \text{who}_{i} \ [ \text{does it seem} \ [ t_{i} \ [ t_{i} \text{ to have left} ] ] ]\]  
   Fails: Case Filter

Select [+TNS] in embedded clause:

9. \[ e \ [ \text{does it seem} \ [ \text{that} \ [ \text{who}_{i} \text{ has left} ] ] ]\]

10. \[ e \ [ \text{does it seem} \ [ e \ [ \text{who}_{i} \text{ has left} ] ] ]\]

11. \[ e \ [ \text{does who}_{i} \text{ seem} \ [ \text{that} \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Case Filter, Binding Condition A, ECP

12. \[ e \ [ \text{does who}_{i} \text{ seem} \ [ e \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Case Filter, Binding Condition A

13. \[ \text{who}_{i} \ [ t_{i} \text{ seems} \ [ \text{that} \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Case Filter, Binding Condition A, ECP

14. \[ \text{who}_{i} \ [ t_{i} \text{ seems} \ [ e \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Case Filter, Binding Condition A

15. \[ \text{who}_{i} \ [ \text{does it seem} \ [ \text{that} \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Subjacency

16. \[ \text{who}_{i} \ [ \text{does it seem} \ [ e \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Subjacency

17. \[ e \ [ \text{does it seem} \ [ \text{who}_{i} \text{ that} \ [ t_{i} \text{ has left} ] ] ]\]  
    Fails: Doubly–filled COMP Filter, ECP
18. \[ e \[ \text{does it seem} \[ \text{who}_i \ e \[ t_i \ \text{has left} \] \] \]

19. \[ e \[ \text{does who}_i \ \text{seem} \[ t_i \ \text{that} \[ t_i \ \text{has left} \] \] \] \]
   
   Fails: Case Filter, Binding Condition C, ECP

20. \[ e \[ \text{does who}_i \ \text{seem} \[ t_i \ e \[ t_i \ \text{has left} \] \] \]
   
   Fails: Case Filter, Binding Condition C

21. \[ \text{who}_i \[ t_i \ \text{seems} \[ t_i \ \text{that} \[ t_i \ \text{has left} \] \]
   
   Fails: Case Filter, Binding Condition C, ECP

22. \[ \text{who}_i \[ t_i \ \text{seems} \[ t_i \ e \[ t_i \ \text{has left} \] \]
   
   Fails: Case Filter, Binding Condition C

23. \[ \text{who}_i \[ \text{does it seem} \[ t_i \ \text{that} \[ t_i \ \text{has left} \] \]
   
   Fails: ECP

24. \[ \text{who}_i \[ \text{does it seem} \[ t_i \ e \[ t_i \ \text{has left} \] \]

The foregoing chapters have been descriptive of the model in terms of its capabilities. This chapter reviews the capabilities in light of its limitations. The concerns center around the representations that have been chosen and the procedures used to manipulate and transform the representations. Little attention will be paid to analyzing the theoretical constructs on which the model is based.

7.1 Representations

As pointed out in Section 2.1, the description of a grammatical system relies on the lexicon and the rules and principles involved in combining lexical items. The role of specific phrase structure rules is minimized, generalizing to a single phrase structure rule system based on X-bar theory, with lexical entries directly determining the course of a derivation. This is in accord with attempts to reduce if not eliminate phrase structure rules (Chomsky 1981; Stowell 1981), in contrast to alternative approaches to extend the phrase structure component, as in Generalized Phrase Structure Grammar (Gazdar and Pullum 1982). The power of the system, then, is a function primarily of the complexity of the lexicon and the principles that determine well-formed constructions in accordance with the X-bar system of phrase structure.

7.1.1 Lexicon

The dictionary component of the lexicon consists of entries keyed by lexical item and consisting of feature descriptions. The definition of a lexical entry, given in 3.1.1(1), indicates a lexical category, but this is merely an abbreviation for some combination of the features [±N,±V] which could just as well be included among the entry's other features. One advantage to representing categories as features is that subcategorization information would not
be restricted to the set of categories but rather could specify from one to any number of features. A couple of examples will illustrate this.

Chomsky’s (1981) conception of the clause is defined as in (1) (=2.2.1(16)):

\[ S \rightarrow NP \text{ INFL } VP \]

where VP is actually a general predicate phrase, i.e. any [+V] constituent, thus allowing for either VP or AP, as in She is upset. (The copula is added as an adjunct, perhaps transformationally, in order to provide an attachment site for tense during Affix Movement.) This overcomes the necessity of the θ-Extraction routine (cf. 6.4.4) and provides a more general representation for the clause.\(^{11}\) The INFL category could then be defined as subcategorizing for [+V] rather than VP.

In the other direction, subcategorization by feature specification can be used to restrict the allowable constituents. For example, the verb put requires its PP to be of type [+LOC] (i.e. locative), so that the subcategorization frame for put might specify [+N,-V], [-N,-V,+LOC] for the NP and PP, respectively. Similar techniques could be used in selecting complementizers, embedded tense of clausal arguments, and selectional restrictions on subcategorized arguments in general.

Nevertheless, the present system makes use of category names to simplify the retrieval of lexical items. No internal search is required to match features, so access time is reduced. It also facilitates the description and access of specifiers and adjuncts by associating them with category names, not features.

\(^{11}\) Even more generality can be obtained by adopting the rule in (i) for the clause as proposed by Williams (1984):

\[ S \rightarrow NP \text{ INFL } X^2 \]

where \( X \) may be any of the major lexical categories, as in the following examples:

\[ (ii) \]

a. Mary is a woman. \( (X^2=NP) \)

b. Mary is hungry. \( (X^2=AP) \)

c. Mary is at home. \( (X^2=PP) \)

d. Mary is eating. \( (X^2=VP) \)
Furthermore, it is not clear as to how the minor categories should be represented in terms of features. Does an article include the features [+N,-V] to indicate affinity with pure pronominals? Evidence suggests that it might, since they do not modify [+V] categories and they show morphological Case in some languages (e.g. German) by virtue of having the [+N] feature. But they are prohibited from occupying the head of an NP phrase, so some additional features are required to distinguish the two. Without creating the necessary features to characterize this distinction, they are simply given the name ART.

The number of features used in this system is in fact limited to that required to achieve a minimum level of descriptive accuracy, principally those of person, number, and gender. The inadequacy of this set is abundantly demonstrated by the range of ungrammatical and anomalous sentences accepted and generated. It seems clear that, for example, some feature regarding animacy ([±ANIMATE]) is required to distinguish between those entities that possess volition and can initiate action, and therefore serve as (thematic) agents of an action, and those that cannot. For example, consider the sentences in (2):

(2)  a. John opened the door.
    b. The key opened the door.

Structurally identical, they differ thematically; John acts as agent in (2a), whereas the key in (2b) has the role of instrument, although both are represented grammatically as the subject.

The problem becomes apparent in the grammar of Spanish. Spanish makes use of the personal a to distinguish direct objects with human(like) or animate qualities, as in (3):

(3)  a. Juan vio el libro.
    ('John saw the book.')
    b. Juan vio a María.
    ('John saw Mary.')
    c. Juan vio el perro.
    ('John saw the dog.')
    d. Juan vio al perro.
    ('John saw the dog.')

The quasi-preposition a is obligatory in (3b) and obligatorily absent in (3a); (3c) expresses an objective event, while (3d) imputes a sentiment of, say, affection or familiarity. Since the current system lacks a feature of animacy and therefore cannot accommodate the use of the
personal a, this leads to incorrect analyses, such as the sentence (4a) which in this model translates to (4b):

(4)  

a. Quién vio la película? ('Who saw the film?')

b. Who did the film see?

The analysis of (4a) involves verb fronting, such that the subject is taken to be la película, inverted with the verb vio. Quién then assumes the role of direct object.

Other features commonly associated with lexical items but not implemented include the count/mass distinction for nouns, locative/directional for prepositions, predicate/non-predicative adjectives, etc. At this time there appears to be no generally accepted theory as to the nature and number of semantic classes required, nor whether strictly binary features are sufficient as opposed to n-ary features or gradations along a continuum.

7.1.2 Phrase Structure

The representation of a phrase according to the metarules of X-bar theory is convenient and theoretically attractive. It provides the framework in which to express linguistic configurations as generalized tree structures (or as a linearization of structures in free phrase order languages such as Latin and Japanese). The choice of trees follows from a long tradition in linguistics, and computer science provides a large store of knowledge in manipulating trees as lists, efficiently traversing them through recursive programming.

The characteristics of a phrase, its features, are naturally associated with its structural representation, insofar as phrase descriptions are independent of each other. Gender, for example, is attached to a noun phrase independent of the predicating verb phrase, although semantic notions such as selectional restrictions may impose further constraints on what may be predicated, i.e. which features are selected and which prohibited.

Some cross-phrase dependencies may also be expressed structurally, such as the agreement of a subject and verb. In this case the abstract but motivated creation of an INFL phrase containing both subject and verb provides the unifying mechanism.
In other cases, the association is not as direct. The mode of a sentence is variously represented orthographically (punctuation), lexically (particles), phonetically (intonation), or pragmatically (intention). In the present model, the mode, attached as a structural adjunct to the lowest COMP node, is passed up as an external flag to affect the initial structure of the sentence in the form of inversion. (That is, a question mark forces inversion.) This mechanism is expedient for present purposes but almost certainly inadequate as a general representation of mode. (Note that this is very similar to the abstract Q 'morpheme as an adjunct to S (Baker 1978).) Alternatively, features of mode may be clausally related and reside in INFL, partaking in some form of affix movement (and reverse affix movement) along with features of tense, voice, aspect, and mood. Presence or absence of the feature will then control inversion.

Some structural possibilities not implemented include relative clauses, coordination, adverbial phrases, and adjectival modifiers. The present system could be extended without a great deal of effort to include some aspect of these, although they are not trivial extensions. Furthermore, translation imposes increased effort in having to work out idiosyncracies in each language. However, the general approach is fairly clear. Relative clauses, as a post-adjunct to an NP, could be defined as an S in which some form of wh-movement has applied to leave a gap in the clause. Some adjustment to the definition of the principles would be necessary, particularly affecting the usage of governing categories, which are not explicitly defined in this system. An approach to coordination is presented in (Fong and Berwick 1985), based on a technique of linearizing the conjoined phrases and finding equivalences within the linearization. Adverbial phrases of manner (e.g. lentamente 'slowly') could be defined as pre- or post-adjuncts to INFL. Adverbials of degree (e.g. muy 'very') may be specifiers of adverbials or adjectival phrases. Other adverbials of time and place (e.g. ahora 'now', aquí 'here') can attach as post-adjuncts to the verb.

Adjectival modifiers may appear to be relatively simple to implement, but in fact present special problems. Predicate adjectives such as small, red, sincere may be positioned prenominally in English, postnominally in Spanish, or in copular phrases in either language:
(5) a. a small apple; the apple is small
   una manzana pequeña; la manzana es pequeña
b. the red houses; the houses are red
   las casas rojas; las casas son rojas
c. a sincere man; the man is sincere
   un hombre sincero; el hombre es sincero

A single lexical entry would appear to suffice for each of these adjectives, and its position may be defined in terms of X–bar as an NP pre–adjunct, post–adjunct, or as a complement of the copula verb, agreeing in number and gender with the noun. However, not all predicate adjectives may be nominal modifiers, as illustrated by the following examples:

(6) a. The boy is afraid.
   b. *He is an afraid boy.
   c. *He is a boy afraid.
   d. He is a boy afraid of the dark.

(7) a. The woman is fond of her nephew.
   b. *She is a fond of her nephew woman.
   c. She is a woman fond of her nephew.

The simple adjective afraid may not be positioned prenominally as in (6b), nor postnominally as in (6c) unless it is part of an adjectival phrase as in (6d). Similarly, the phrase fond of her nephew may not appear prenominally in (7b) but because of its transitivity, like (6d), must appear postnominally, as in (7c). Moreover, descriptive adjectives in Spanish may often appear on either side of the noun with differing semantic connotations:

(8) a. un hombre pobre
   ('a poor man', i.e. without money)
b. un pobre hombre
   ('a poor [unfortunate, wretched] man')

But adjectives modified by adverbials must always follow the noun:

(9) a. un hombre muy pobre
   ('a very poor man')
b. *un muy pobre hombre

Some (ad hoc) extension to feature types is required to resolve these lexical differences. On account of considerations such as these, the implementation of adjective phrases has not been attempted, other than those that introduce clausal arguments (e.g. likely, probable) and whose distribution is therefore invariant.
Additional extensions to phrase structure representation include the variety of Move-a operations, e.g. Extrapolation with It Insertion (e.g. *It amazed me that Fred won a prize*), Topicalization (e.g. *This picture, I like*), Heavy NP Shift (e.g. *We have with us this evening a very special guest*), Stylistic Inversion in Spanish (e.g. *Canta bien la chica* 'The girl sings well'), and Passivization. Some machinery has been included in the model to handle the latter, but the major problem here lies more in translation. True Spanish passives are rendered with the auxiliary *ser* 'be' and the past participle which agrees in number and gender with the subject:

(10) a. El juguete fue roto ayer.
    ('The toy was broken yesterday.')
    b. Las reuniones fueron organizadas por los obreros.
    ('The meetings were organized by the workers.')

The *ser* auxiliary has not been added, nor have participles that, like adjectives, require agreement. More commonly, passive voice is expressed with the reflexive pronoun *se*, as in:

(11) a. Se prohíbe fumar.
    ('Smoking is prohibited.')
    b. Esas casas se vieron por los agentes.
    ('Those houses were seen by the agents.')
    c. Se venden tamales aquí.
    ('Tamales are sold here.')

Reflexives, including all other object pronoun clitics, have not been implemented here. Relevant theoretical discussion of cliticization within the GB framework may be found in Jaeggli (1982) and Borer (1984).

7.1.3 Configurationality

The primary design goal of the system is to construct a model of grammatical knowledge that achieves cross-linguistic generality. It emphasizes the representation of a universal grammar as an embodiment of general principles, and individual core grammars as parametrized variations to the principles. For a not insignificant subset of the grammars of English and Spanish, this goal has been achieved.
An obvious question that arises is to what extent this system will accommodate the grammars of other languages, especially non-Indo-European languages. To the extent that a language is configurational in terms of embedded phrase structure (Chomsky 1981), e.g. English, the model would ideally require only minor, if any revision, apart from establishing the lexicon and the parameters. Unfortunately, the analysis component of the system, i.e. the parser, is designed for strictly top-down left-to-right processing, imposed by the Prolog execution of the grammar rules. This presupposes head-initial grammars, such as English and Spanish, and other so-called SVO languages. This strategy falls apart with German, for example, an SOV language. Since the object is an argument of the verb and the verb is in final position, an indeterminate amount of lexical material must be passed before the verb is recognized and its complement determined. This suggests that ultimately a bottom-up approach may prove to be more general and preferable, using some technique such as "chart" parsing (Kaplan 1973), as in the Q-system (Colmerauer 1970), Prolog's predecessor.

As to so-called non-configurational languages (e.g. Japanese), a bottom-up approach to parsing may again be preferable. The verb will indicate the argument structure that must be satisfied, with order of arguments relatively freely distributed in pre-verbal position. A mechanism, possibly incorporated as a reverse-transformation, could map a non-configurational structure into a configurational D-structure in which thematic relations are structurally represented as in configurational languages. The same alterations to the parsing strategy required for German may accordingly hold for Japanese, and by extension English, Spanish, etc., at least as far as the satisfaction of the θ-Criterion and Extended Projection Principle.

Since the present model is configurationally based, the statement of the ECP, binding conditions, and Case Filter are all expressed in terms of structural configurations, and a language so represented should require minimal change in the model. At most, new structural configurations would be necessary to express the new environment(s) where the conditions may hold. Once again, if non-configurational languages can be equivalently represented configurationally, then no substantial changes or additions would be required in the statement
of the principles, apart from those motivated by ongoing empirical investigations.

### 7.1.4 Well-Formedness Conditions

The well-formedness conditions as implemented here act as output filters on S-structures. An obvious improvement to the time efficiency is to execute the conditions in parallel, given parallel machine architecture. If any one condition fails it halts all ongoing tasks, rather than the current situation where a structure may sequentially pass several conditions, possibly incurring substantial computation, before being rejected. This also motivates the use of filters to reduce the rule component, since filters may operate in parallel, but rules are necessarily serial operations.

Ideally, though, the model would be constructed in such a way that only valid S-structures are generated in the first place. This suggests a closer integration of the principles within the rules of grammar, as subjacency is (necessarily) implemented. As an example, take Move-α, which includes the following preconditions (with subjacency being a condition on execution):

(12) **Move-α Preconditions**

(i) α agrees with its destination
(ii) the destination is an empty position
(iii) α and its trace satisfy the binding conditions
(iv) the trace would be properly governed

Condition (i) requires a correspondence between the lexical category of α and its destination, and further that if the destination is COMP then α includes [+WH]. The latter is implemented as a filter after Move-α has applied (which leads to some unnecessary computation, as phrases are brought one at a time to COMP only to be rejected by the filter).

Condition (ii) is trivial. Conditions (iii,iv) implement the principles of binding and government as output filters. One advantage to the separation is that it localizes the definition of the principles to a single component rather than scattering them among different rules, which would result in, for example, an ECP condition on NP-traces in one rule, an ECP condition on wh-traces in another, and a disjoint ECP condition on PRO. The disadvantage is in the
additional computation that takes place before an invalid movement is rejected, as in condition (i) for *wh*-movement.

In a sense, this integration has been achieved in the expression of c-command, a recurring notion in GB theory. The phrase structure design along with top-down recursion effectively implements the concept of c-command, obviating the necessity of a special predicate such as "c-command(X,Y)". Any material contained in the specifier, complement, or adjunct, and no other, is inherently c-commanded by the head. The definitions of government and binding, as implemented, require no reference to c-command as a precondition; it is inherent in the structure itself.

But top-down recursion is not always satisfactory, at least not for the binding relation. In a recursive descent of phrase structure, government of an argument, including ECs, may be locally determined by isolating the governor and its c-commanded phrases. Thus the operations that rely on government, i.e. assignment of Case and θ-roles and validation of the ECP, can be performed within the recursive descent. However, binding is expressed as a passive operation. That is, rather than determining if a binds b, the conditions depend on whether b is bound/not bound by a, suggesting an upward search rather than a downward search. For example, a name (or variable) must not be bound by a c-commanded phrase (= Condition C). This is implemented in one of two ways. Either a search is made upward from a name, halting when a co-indexed antecedent is found or the top node is reached, or a search is made from every NP downward to see if by chance it binds a name. The latter is the approach taken in the present model, which suffers from the inefficiency of having to search the entire length of every downward path from every NP just in case it binds a name (or variable, as it were, in this model).

An interference in integrating principles with rules arises from exceptional government phenomena such as S-deletion. S-deletion affects the bounding category for the application of the binding conditions. But since S is a cyclic node in terms of the transformational cycle, a procedure within the cycle has no access to structural information above S to determine if
S-deletion is in effect. Therefore the satisfaction of the binding conditions is sometimes relevant at the current cycle and other times at the next higher cycle. The same applies in all cases where government of the subject is crucial. Consequently S-deletion is a major contributor to the decision to implement the principles as well-formedness conditions on output.

7.2 Execution

7.2.1 Relation to Logic Grammars

The use of Prolog in natural language processing stems from the introduction of logic grammars in 1975 (Colmerauer 1975). Since the logic grammar formalism is a part of the Prolog language, grammars can be written in concise, familiar notation and be interpreted directly by Prolog. Several notable logic grammar formalisms include Metamorphosis Grammars (Colmerauer 1975), Definite Clause Grammars (Pereira and Warren 1980), Extraposition Grammars (Pereira 1981), Modifier Structure Grammars (Dahl and McCord 1983), and Definite Clause Translation Grammars (Abramson 1984).

Prolog is designed primarily for top-down backtracking analysis of sentences, although a bottom-up Prolog parser has been designed (Stabler 1983) to implement the deterministic parser of Marcus (1980).

In contrast to the grammars mentioned above, this implementation has a very small grammar rule component. All phrase structures are parsed using the X-bar metarule system, guided by the lexicon directly with minimal backtracking. Arguments are not used in the rules except to name the language and category of the phrase undergoing the parse, and to return its parse tree. Validation only occurs during the reverse-transformation stage. This is not to say that validation should not occur during parsing, but only that this system maintains a separation of function for the sake of modular design.

As the system is designed to accept and process only grammatical sentences, the validations that are performed occur primarily as a side effect of establishing the sentence's
D-structure. For example, in restoring the tense feature to INFL, a check is made to ensure that tense is specified once and only once, preventing sentences such as:

(13) a. Paul can leaves for Costa Rica tomorrow.
    b. She drink mineral water.

where (a) is overspecified for tense and (b) is underspecified. Similarly, the percolation routine contains consistency checks for subject-verb agreement and article-noun agreements as a side-effect of feature identification. Sentences containing errors of these sorts, such as (13), will parse successfully but fail in the reverse-transformation component.

Apart from the logic grammar formalism, the derivation of D-structure from S-structure (and vice versa) is largely a functional process, for which a functional programming language (such as LISP) is entirely adequate. The output of one function (a transformation) is input to a following function, with often no non-determinism involved. The unification provided by Prolog is mostly used as a pattern-matching device for phrasal descriptions, not for instantiation of logical variables, except of course for output variables from function application. As such, the power of Prolog may be under-utilized, but its relative perspicuity, especially in grammar notation, over LISP-like languages (as noted by Pereira and Warren (1980)) makes it an attractive programming language for natural language research.

### 7.2.2 Parsing

The areas where parsing shows difficulty is in relation to empty and optional arguments. Consider the two sentences below:

(14) a. What did you eat?
    b. Where did you eat?

The verb *eat* is transitive in (a) and intransitive in (b), but the parser is unable to determine which is applicable at the time the verb is being retrieved from the lexicon. The part of the parse tree that contains the wh-phrase is not accessible at this point. If it assumes an intransitive use then (a) will fail the $\theta$-Criterion when reverse-Move-$\alpha$ attempts to relocate the $wh$-NP to a Case-marked, hence $\theta$, position. What is required is a strategy similar to
that taken in (b) where a position is actually created to contain the \textit{wh}-element, only instead of a \(\theta\) adjunct, a \(\theta\) argument is created, provided the verb subcategorizes for a phrase of that type and that its subcategorization frame is not already satisfied.

The verb \textit{be} has particular difficulty with ECs, since as a main verb it may take NP, PP, or AP, as in:

(15) a. Who are you \(t\)?  
b. Where are you \(t\)?  
c. How are you \(t\)?

The EC defaults to an NP by the parser for the sake of creating an argument position, so that (b,c) cannot be recognized. A further problem arises when the argument is incorrectly instantiated, as in:

(16) What is that behind you?

The phrase \textit{behind you} is parsed as a PP argument of \textit{is}, so no EC is created. The \textit{wh}-NP \textit{what} will then fail to find a \(\theta\) position to occupy.

(In passing, an oversight in design adds to \textit{be}'s problems as a main verb. It fails to participate in Have/Be Raising, resulting in the following anomalous translation:

(17) a. Dónde estás?  
     ("Where are you?")  
b. Where do you \textit{be}?

It also causes the following sentence to be unrecognizable since the parser fails to find a main verb in canonical position:

(18) Is he likely to be late?

This is easily corrected by modifying the Have/Be Raising rule appropriately.)

The pro-drop parameter as implemented fails to adequately distinguish \textit{pro} from PRO in cases where \textit{pro} represents a third person pronoun. When the parser reads sentence (19a) it converts it to the D-structure in (b) with an EC in the position labelled NP:

(19) a. Es posible que \textit{e} sale hoy.  
     ("It is possible that he/she/it is leaving today.")  
b. \(\text{[ e ser posible [ NP [±TNS] salir hoy ]]}\)  
c. Es posible \textit{e} salir hoy.  
     ("It is possible to leave today.")
Since the null subject routine cannot distinguish the correct pronoun for the EC in (a), it leaves its features empty. When translated to English, the NP will be lexicalized to each of the possible pronouns. However, when given sentence (c), it will also create D-structure (b). Since the tense feature freely alternates (in the set_tns routine), [+TNS] will cause the NP to appear as pro, as in (a), resulting in lexicalized pronouns being generated. But the EC in (c) is PRO, not pro, so that the English sentences of (a) cannot be valid translations of (c). The proper way to resolve this is to provide the ECs with the appropriate [±ANAPHOR,±PRONOUN] features, either at the time of parsing when the EC is created or as a subfunction of percolation, when other features are determined.

7.2.3 Translation

The translation component is primitive in comparison to developed machine translation systems (see Slocum (1985) for a review of such). This is largely because no semantic analysis occurs in the translation. The system is probably best characterized as an interlingual system (Slocum 1985) in which a source language sentence is converted into an essentially language-independent representation, i.e. the D-structure. The language-independent character derives from the equivalence of lexical items in the dictionary. For example, the verb eat has an equivalent word in Spanish, comer, in which subcategorization frames and thematic roles are identical. Therefore the D-structure containing eat will be identical to that for the Spanish comer and (presumably) for any language with an equivalent concept. Hence, the language-independent nature of the interlingual representation. The translation of lexical items itself, then, is simply a lexical transfer process based on the correspondences established in the (bilingual) lexicon.

Repeating what was said in Section 6.5, no attempt is made to provide for the range of translations for each lexical item. This avoids the task of handling the "17 words for snow in Eskimo" problem, which otherwise would require some circumlocutory translations.
Lexical equivalence needn't be restricted to single lexical items. Thus, the English verb to feed is translated in Spanish as dar de comer ('give to eat'); typewriter is translated as máquina de escribir ('machine to write'). The mechanisms exist to effect this form of translation.

An area that does not lend itself well to translation in this system involves verbs in which the grammatical function of the thematic roles are reversed. A typical example is the verb like in English, generally translated using the Spanish gustar, as in:

(20) a. Mary does not like pastries.
    b. A María no le gustan los pasteles.

The direct object in English becomes the subject in Spanish. Subcategorizations are identical, but the \( \theta \)-role of "agent" in Spanish is assigned to the NP dominated by the VP, not S as in English. As \( \theta \)-theory is not developed in this system to the point of specifying actual \( \theta \)-roles, the distinction cannot be made between like and gustar, nor between verbs of similar nature, e.g. olvidar 'forget', perder 'lose', etc.

Two possible solutions are (1) to retranslate to a word that does have equivalent functional and thematic structure, and (2) to implement \( \theta \)-role assignment. Solution 1 for like/gustar is to retranslate gustar to please ("The pastries are not pleasing to Mary."). but like has no equivalent. Besides being a poor translation in terms of normal usage, some means are required to deal with items which simply have no strict translation.

Complications immediately arise when the structural and thematic properties diverge, as in:

(21) a. I am hungry.
    b. Tengo hambre.
    ('I have hunger.')

The predicate adjectival usage in (a) is substituted for a transitive verb in (b). The translation component must have the ability to make structural alterations with corresponding \( \theta \)-role changes, as required.

Solution 2 is obviously preferable, since it imparts a degree of semantic information into an existing syntactic structure. The necessity of explicit subcategorization information
becomes doubtful given a well-defined theory and notation of θ-roles.

A major problem in translation is ambiguity in all its various forms. No attempt has been made in this system at disambiguation since it requires contextual information not available in the surface sentence. For example, the sentence:

(22) Time flies like an arrow.

has many possible syntactic structures depending on the specific meanings of each item and the context (sometimes pragmatically absurd) in which they occur.

As mentioned earlier, some assumptions are made regarding the ambiguity of PP-attachments, exemplified by the classical sentence:

(23) a. I saw a man in the park with the telescope.
   b. Yo vi a un hombre en el parque con telescopio.

Apart from the lexical ambiguity of the verb saw (the further ambiguity of saw as verb or noun is resolved by the Extended Projection Principle), the correct attachment of the two PPs is many-wise ambiguous. The PP in the park may modify man or saw, and the PP with the telescope may modify man, saw, or park. However, disambiguation in (23) turns out to be unnecessary. Sentence (a), ambiguous in English, remains ambiguous in Spanish (b).

In other instances, disambiguation becomes essential, as for (24a) below which has either (b) or (c) as its translation:

(24) a. The man told the woman that he loved the story.
    b. El hombre le dijo a la mujer que le gustaba el cuento.
    c. El hombre contó el cuento a la mujer a quien quería.

Sentence (b) has the sense of "The story that the man loved was told to the woman (by the man)", whereas (c) has the sense "The story was told (by the man) to the woman whom he loved." In this example, not only are the structural configurations radically different, but they lead to quite distinct lexical translations; tell, when subcategorizing for a clause, translates to decir, (b), but when used in the sense of recounting, translates to contar, (c). Further, the translation of the verb love depends on the characteristics of the direct object. When animate, it translates to querer; otherwise gustar. Without the benefit of context (or intonation patterns) there is no way of determining which of (b) or (c) is the correct translation of (a).
Another area which receives minimal treatment is the specification of time. The system recognizes tense, aspect, and voice, clearly, especially in Spanish, but makes no correspondence of the usage patterns between English and Spanish. Whereas English has one simple past tense, Spanish has two, the imperfect and the preterit. The imperfect is often translated into past progressive. Present progressive in English is commonly expressed by simple present in Spanish. Other constructions will use the perfective aspect in one language but not in the other. An adequate theory of temporal representation is required (see, e.g., Hornstein (1981) and Yip (1985)) and an understanding of how different languages express equivalent time references.

7.2.4 Generation

Little more will be stated here on generation than what has already been covered under parsing, since the two operations are ideally inverse operations of each other. In fact, it is interesting to consider the possibility of using a single predicate function to achieve both, i.e.:

(25) parse_gen(L, Surface, Deep).

where either 'Surface' is supplied, resulting in the D-structure 'Deep', or 'Deep' is supplied giving 'Surface' or a set of 'Surface's. In general, such reversible predicates are not feasible for complex relations such as this, although certain sub-processes may be handled as such. In the transformational cycle, for example, many transformations are reversible, such as Do Support and It Insertion. Others are nearly so, such as the Inversion transformations, where Have/Be Raising occurs before SAI in one direction, and the opposite order is performed in the reverse direction. However, other processes are specifically designed to be unidirectional, and it is difficult to see how they may generalize to bidirectionality. For example, the Move-a transformation assigns indices and features, but the reverse movement ignores such assignments; it has been optimized, in a sense, to remove them from consideration, and in fact they are not even present at the time of the reverse-Move-a operation.
The use of reversible predicates would most certainly reduce the overall size of the transformational component, but possibly at the expense of increased processing time. This remains an area for further investigation.
Chapter 8

Conclusion

This thesis has described the implementation of a model of natural language that reflects current theories of transformational grammar. The fundamental motivation behind the theory is to develop a description of natural language grammars that achieves a high degree of explanatory capacity. A number of basic principles are formulated to this end which provide for a concise, modular, cross-linguistic account of grammatical structure. The principles in effect represent the embodiment of a universal grammar. Specific language grammars are instantiations of this universal grammar, with language variations attributed to parametrizations of the principles.

As there exists little published research on systems whose design is based on principles of GB, it is difficult to do a comparative analysis. Principal among what is available include the work of Berwick and Weinberg (1984) regarding the computational complexity of GB-oriented systems, that of Wehrli (1983, 1984) on a GB parser of French, and current developments at MIT (Barton 1984; Dorr 1985) in parsing and translation in the context of GB theory. The present study represents a further contribution to a growing body of practical GB systems.

A subset of the grammars of English and Spanish is presented, including language-specific rule applications along with the parameters that affect general rules. The conditions on well-formedness are represented as general conditions, again with simple parametric values determining the effect.

Translation is simplified by relating equivalent D-structures, allowing for some degree of structural differences in the representations. The equivalence reduces the size of the lexicons, since identical features between lexical items need be represented only once.
The lack of semantics is intentional, since the principles modeled here relate primarily to syntax. The analysis of logical form (LF) becomes the next major area of research, incorporating new rules and principles of LF and borrowing from existing ones where motivated. At the same time, further extensions to the syntactic component will expand the range of analyzable surface constructions (PF).

With adequate representations of PF and LF, and sufficient memory to record their discourse history, it becomes possible to analyze discourse in terms of knowledge networks and inference rules. This of course leads to a more refined sense of the term "natural language understanding", in which a higher degree of intelligence is demonstrated.

This is one goal among many in the combined study of linguistics, artificial intelligence, and cognitive science.
Bibliography


109


Taraldsen, T. (1978) "On the NIC, Vacuous Application and the That–Trace Filter," mimeographed ms., MIT; distributed by The Indiana University Linguistics Club, Bloomington, IN.

15:103–129.


Appendix A. EXAMPLES

The examples shown here illustrate the properties of the model. Sentences are input in one language and converted to D-structure representation. A translation of D-structures is made, followed by the generation of all allowable S-structures. The working vocabulary is fairly small, but the intention is to model representative lexical items, rather than handle a broad coverage of the languages.

The examples were produced on a VAX 11/750 running under UNIX 4.2 BSD at the University of British Columbia. The program is written in C-Prolog 1.5 (Pereira 1984).

Some minor amount of post-editing has been done to facilitate inclusion of these examples in the appendix. Diacritics have been eliminated except where ambiguity might result (e.g. habló vs. hablo). In the examples, a diacritic follows the vowel over which it applies.

As noted in the text, the sentences generated from questions in the source language include echo questions, yes/no questions, and wh questions, as illustrated in some of the examples below, such as Example 3. No attempt is made to limit the form of the output to that of the input sentence.

Example 1.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: John said he thought his brother had seen the film.
Juan dijo que e'l pensaba que su hermano había visto la película.
Juan dijo que pensaba que su hermano había visto la película.

Example 2.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: creo que eres mi amigo.
I believe that you are my friend.
I believe you are my friend.
I believe you to be my friend.

Example 3.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: quie'n parece haber salido?

Does who seem to have left?
Who seems to have left?
Does it seem that who has left?
Does it seem who has left?
Does it seem who has left?
Who does it seem has left?

Example 4.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: es probable que la mujer hablara' antes de la reunion.

It is likely that the woman will speak before the meeting.
It is likely the woman will speak before the meeting.
The woman is likely to speak before the meeting.

It is probable that the woman will speak before the meeting.
It is probable the woman will speak before the meeting.

Example 5.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: should mary know the truth about her father?

Debe saber Maria la verdad acerca de su padre?
Debe Maria saber la verdad acerca de su padre?

Example 6.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: parece que juan ha estirado la pata.

It seems that John has died.
It seems John has died.
John seems to have died.

It seems that John has kicked the bucket.
It seems John has kicked the bucket.
John seems to have kicked the bucket.

Example 7.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: where did you put my book?

Pusiste tu' mi libro do'nde?
Pusiste mi libro do'nde?
Do'nde pusiste tu' mi libro?
Do'nde pusiste mi libro?

Example 8.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: when did my sister put this book on the table?
Puso mi hermana este libro en la mesa cuándo?
Cua'ndo puso mi hermana este libro en la mesa?
Puso mi hermana este libro sobre la mesa cuándo?
Cua'ndo puso mi hermana este libro sobre la mesa?

Example 9.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: mary is likely to come to the meeting after the film.
Es probable que Maria viene a la reunion después de la pelicula.

Example 10.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: después de cuál pelicula salieron?
Did they leave after which film?
Which film did they leave after?
After which film did they leave?

Example 11.
Enter source language: english.
Enter target language: spanish.
Enter sentence in english: which film did they leave after?
Salieron ellos después de cuál pelicula?
Salieron después de cuál pelicula?
Después de cuál pelicula salieron ellos?
Después de cuál pelicula salieron?

Example 12.
Enter source language: spanish.
Enter target language: english.
Enter sentence in spanish: quiero su libro.
I want his book.
I want her book.
I want its book.
I want their book.
I want your book.

Example 13.
Enter source language: Spanish.
Enter target language: English.
Enter sentence in Spanish: salió?

Did he leave?
Did she leave?
Did it leave?

Example 14.
Enter source language: English.
Enter target language: Spanish.
Enter sentence in English: how much time do we have?

Tenemos nosotros cuánto tiempo?
Tenemos cuánto tiempo?
Cuánto tiempo tenemos nosotros?
Cuánto tiempo tenemos?

Example 15.
Enter source language: English.
Enter target language: Spanish.
Enter sentence in English: who did john know mary had said some man had spoken with?

Sabía Juan que María había dicho que algún hombre había hablado con quién?
Con quién sabía Juan que había dicho María que algún hombre había hablado?
Con quién sabía Juan que había dicho María que había hablado algún hombre?

Example 16.
Enter source language: Spanish.
Enter target language: English.
Enter sentence in Spanish: quién vio la película?

Did the film see who?
Who did the film see?
Appendix B. Prolog Code – English Lexicon

/* Lexicon: English */

/* Morphology */

suffixtable(english, ies, y, n, [pl(+)]).
suffixtable(english, ves, f, n, [pl(+)]).
suffixtable(english, ves, fe, n, [pl(+)]).
suffixtable(english, es, "", n, [pl(+)]).
suffixtable(english, s, "", n, [pl(+)]).

suffixtable(english, ies, y, v, [pl(-), per(3), tns(pres)]).
suffixtable(english, ied, y, v, [tns(past)]).
suffixtable(english, ed, e, v, [tns(past)]).
suffixtable(english, ed, "", v, [tns(past)]).
suffixtable(english, en, e, v, [perf(+)]).
suffixtable(english, en, "", v, [perf(+)]).
suffixtable(english, ing, e, v, [prog(+)]).
suffixtable(english, ing, "", v, [prog(+)]).

/* Function Words */

dict(english, a, art, [f(pl(-)), spanish(un)]).
dict(english, an, art, [f(pl(-)), spanish(un)]).
dict(english, another, art, [f(pl(-)), spanish(otro)]).
dict(english, each, art, [f(pl(-)), spanish(cada)]).
dict(english, every, art, [f(pl(-)), spanish(cada)]).
dict(english, her, art, [spanish(su)]).
dict(english, his, art, [spanish(su)]).
dict(english, how_many, art, f([wh(+), pl(+)]),
    spanish('cua''ntos')).
dict(english, how_much, art, f([wh(+), pl(-)]),
    spanish('cua''nto')).
dict(english, its, art, [spanish(su)]).
dict(english, many, art, [f(pl(+)), spanish(muchos)]).
dict(english, my, art, [spanish(mi)]).
dict(english, no, art, [spanish(ningun)]).
dict(english, other, art, [f(pl(+)), spanish(otros)]).
dict(english, our, art, [spanish(nuestro)]).
dict(english, several, art, [f(pl(+)), spanish(varios)]).
dict(english, some, art, [spanish(algun)]).
dict(english, that, art, [f(pl(-)), spanish(ese)]).
dict(english, the, art, [spanish(el)]).
dict(english, their, art, [spanish(su)]).
dict(english, these, art, [f(pl(+)), spanish(estos)]).
dict(english, this, art, [f(pl(-)), spanish(este)]).
dict(english, those, art, [f(pl(+)), spanish(esos)]).
dict(english, what, art, [f(wh(+))],
    spanish(['que'' ','cua''1'])].
dict(english, which, art, [f(wh(+)), spanish('cua''1')]).
dict(english, your, art, [spanish(tu)]).
dict(english, had, perf, [f(tns(past)), root(have)]).
dict(english, has, perf, [f([tns(pres), per(3), pl(-)]),
    root(have))]
dict(english, have, perf, [spanish(haber)]).
dict(english, be, prog, [spanish(estar)]).
dict(english, been, prog, [f(perf(+)), root(be)]).
dict(english, is, prog, [f([tns(pres), per(3), pl(-)]), root(be)]).
dict(english, am, prog, [f([tns(pres), per(1), pl(-)]), root(be)]).
dict(english, are, prog, [f([tns(pres), pl(+)]), root(be)]).
dict(english, was, prog, [f([tns(past), per(2)]), root(be)]).
dict(english, were, prog, [f([tns(past), per(2)]), root(be)]).
dict(english, is, prog, [f([tns(past), per(3), pl(-)]), root(be)]).
dict(english, were, prog, [f([tns(past), per(2)]), root(be)]).
dict(english, that, comp, [subcat(infl), spanish(e)]).
dict(english, for, comp, [f(tns(-)), subcat(infl), spanish(e)]).
dict(english, he, n, [f(pro(+)), spanish('e' '1')]).
dict(english, how much, n, [f(wh(+)), spanish('cua' 'nto')]).
dict(english, how many, n, [f([wh(+), pl(+)]),
    spanish('cua' 'ntos')]).
dict(english, i, n, [f([per(1), pro(+)]), proper(+), spanish(yo)]).
dict(english, it, n, [f(pro(+)), spanish('e' '1')]).
dict(english, she, n, [f(pro(+)), spanish(ella)]).
dict(english, that, n, [spanish(eso)]).
dict(english, these, n, [f(pl(+)), spanish('e' 'stos')]).
dict(english, they, n, [f([pl(+), pro(+)]), spanish(ellos)]).
dict(english, this, n, [spanish(esto)]).
dict(english, those, n, [f(pl(+)), spanish('e' 'sos')]).
dict(english, we, n, [f([per(1), pl(+), pro(+)]),
    spanish(nosotros)]).
dict(english, what, n, [f([wh(+), per(3), pl(-)]),
    spanish('que' '')]).
dict(english, which, n, [f([wh(+), per(3), pl(-)]),
    spanish('cua' 'l')]).
dict(english, who, n, [f([wh(+), per(3), pl(-)]),
    spanish('quie' 'n')]).
dict(english, whom, n, [f([wh(+), per(3), pl(-)]),
    spanish('quie' 'n')]).
dict(english, you, n, [f([per(2), pl(-), pro(+)]),
    spanish('tu' '')]).
dict(english, can, infl, [f(tns(pres)), subcat(v), irr(could),
    spanish(poder)]).
dict(english, could, infl, [f(tns(past)), root(can)]).
dict(english, should, infl, [f(tns(pres)), subcat(v),
    spanish(deber)]).
dict(english, did, infl, [f(tns(past)), root(do)]).
dict(english, do, infl, [f([tns(pres), irr([does, did]), subcat(v)])].
dict(english, does, infl, [f([tns(pres), per(3), pl(-)]), root(do)]).
dict(english, to, infl, [f(tns(-)), subcat(v), spanish(e)]).
dict(english, will, infl, [f(tns(fut)), subcat(v), spanish(e)])
dict(english, about, p, [subcat(n), spanish(acerca_de)]).
dict(english, above, p, [subcat(n), spanish([encima_de, sobre]])).
dict(english, after, p, [subcat(n), spanish(después_de)]).
dict(english, against, p, [subcat(n), spanish(contrá)])..
dict(english, among, p, [subcat(n), spanish(entre)]).
dict(english, around, p, [subcat(n), spanish(alrededor_de)]).
dict(english, at, p, [subcat(n), spanis h(a)]]).
dict(english, before, p, [subcat(n), spanish(antes_de)]).
dict(english, behind, p, [subcat(n), spanish(detrás_de)]).
dict(english, below, p, [subcat(n), spanish(debajo_de)]).
dict(english, beneath, p, [subcat(n), spanis h([bajo, debajo_de])])

/* CONTRACTIONS */
contraction(english, 'i''m', [i, am]).
contraction(english, 'I''m', ['I', am]).
contraction(english, 'i''ve', [i, have]).
contraction(english, 'I''ve', ['I', have]).

/* COLLOCATIONS */
collocation(english, [how, many], how_many).
collocation(english, [how, much], how_much).

/* DICTIONARY */
dict(english, am, v, [f([tns(pres), per(1), pl(-)]), root(be)])
dict(english, are, v, [f([tns(pres), per(2)]), root(be)])
dict(english, are, v, [f([tns(pres), pl(+)]), root(be)])
dict(english, be, v, [subcat(a), subcat(n), subcat(p), perf(been),
                         prog(being), irr([is, am, are, was, were]),
                         spanish([estar, ser])]).
dict(english, known, v, [f(perf(+)), root(know)]).
dict(english, leave, v, [irr(left), perf(left), spanish(salir)]).
dict(english, left, v, [f([perf(+), tns(past)]), root(leave)]).
dict(english, likely, a, [f(theta(-)), subcat(comp), sdel(+),
            spanish(probable)]).
dict(english, made, v, [f([perf(+), tns(past)]), root(make)]).
dict(english, make, v, [perf(made), irr(made), subcat(n),
            spanish(hacer)]).
dict(english, man, n, [pl(men), spanish(hombre)]).
dict(english, mary, n, [proper(+), spanish(maria)]).
dict(english, meeting, n, [spanish(reunion)]).
dict(english, men, n, [f(pl(+)), root(man)]).
dict(english, mother, n, [spanish(madre)]).
dict(english, movie, n, [spanish(cine)]).
dict(english, paw, n, [spanish(pata)]).
dict(english, possible, a, [f(theta(-)), subcat(comp),
            spanish(possible)]).
dict(english, probable, a, [f(theta(-)), subcat(comp),
            spanish(probable)]).
dict(english, put, v, [f([perf(+), tns(past)]), root(put)])..
dict(english, put, v, [subcat([n,p]), irr(put), perf(put),
            prog(putting), spanish(poner)]).
dict(english, putting, v, [f(prog(+)), root(put)])..
dict(english, said, v, [f([perf(+), tns(past)]), root(say)])..
dict(english, saw, v, [f(tns(past)), root(see)])..
dict(english, say, v, [subcat(comp), irr(said), perf(said),
            spanish(decir)]).
dict(english, see, v, [subcat(n), prog(seeing), perf(seen),
            irr(saw), spanish(ver)])..
dict(english, seeing, v, [f(prog(+)), root(see)])..
dict(english, seem, v, [f(theta(-)), sdel(+), subcat(comp),
            spanish(parecer)]).
dict(english, seen, v, [f(perf(+)), root(see)])..
dict(english, sell, v, [irr(sold), perf(sold), subcat(n),
            spanish(vender)]).
dict(english, side, n, [spanish(lado)]).
dict(english, sister, n, [spanish(hermana)]).
dict(english, sleep, v, [irr([slept]), perf(slept),
            spanish(dormir)]).
dict(english, slept, v, [f([perf(+), tns(past)]), root(sleep)]).
dict(english, sold, v, [f([perf(+), tns(past)]), root(sell)]).
dict(english, spain, n, [proper(+), spanish(espana)]).
dict(english, speak, v, [irr(spoke), perf(spoken), subcat(p),
            spanish(hablar)]).
dict(english, spoke, v, [f(tns(past)), root(speak)])..
dict(english, spoken, v, [f(perf(+)), root(speak)])..
dict(english, stretch, v, [subcat(n), spanish(estirar)])..
dict(english, table, n, [spanish(mesa)]).
dict(english, thing, n, [spanish(cosa)]).
dict(english, think, v, [subcat(comp), irr([thought]),
            perf(thought), spanish(pensar)]).
dict(english, thought, v, [f([perf(+), tns(past)]), root(think)])..
dict(english, time, n, [spanish(tiempo)]).
dict(english, truth, n, [spanish(verdad)]).
dict(english, want, v, [perf(wanted), subcat(comp), subcat(n),
            sdel(+), spanish(querer)]).
dict(english, wanted, v, [f([perf(+), tns(past)]), root(want)]).
dict(english, was, v, [f([tns(past), per(1), pl(-)]), root(be)]).
dict(english, was, v, [f([tns(past), per(3), pl(-)]), root(be)]).
dict(english, went, v, [f(tns(past)), root(go)]).
dict(english, were, v, [f([tns(past), per(2)]), root(be)]).
dict(english, were, v, [f([tns(past), pl(+)]), root(be)]).
dict(english, woman, n, [pl(women), spanish(mujer)]).
dict(english, women, n, [f(pl(+)), root(woman)]).
dict(english, written, v, [f(perf(+)), root(write)]).
dict(english, wrote, v, [f(tns(past)), root(write)]).
Appendix C. Prolog Code - Spanish Lexicon

/*************************************************************/
/* LEXICON: Spanish */
/* ***********************************************************/

/** MORPHOLOGY **/

suffixtable(Spanish,es,"",n,[pl(+)]).
suffixtable(Spanish,s,"",n,[pl(+)]).
suffixtable(Spanish,a,o,n,[fem(+)]).
suffixtable(Spanish,as,o,n,[fem(+),pi(+)]).
suffixtable(Spanish,o,ar,v,[per(l),pl(-),tns(pres)]).
 suffixtable(Spanish,as,ar,v,[per(2),pl(-),tns(pres)]).
suffixtable(Spanish,a,ar,v,[per(3),pl(-),tns(pres)]).
suffixtable(Spanish,amos,ar,v,[per(l),pl(+),tns(pres)]).
suffixtable(Spanish,an,ar,v,[per(3),pl(+),tns(pres)]).
suffixtable(Spanish,abas,ar,v,[per(2),pl(-),tns(past)]).
suffixtable(Spanish,aba,ar,v,[per(3),pl(-),tns(past)]).
suffixtable(Spanish,abamos,ar,v,[per(l),pl(+),tns(past)]).
suffixtable(Spanish,aban,ar,v,[per(3),pl(+),tns(past)]).
suffixtable(Spanish,'e''',ar,v,[per(l),pl(-),tns(past)]).
suffixtable(Spanish,aste,ar,v,[per(2),pl(-),tns(past)]).
suffixtable(Spanish,'o''',ar,v,[per(3),pl(-),tns(past)]).
suffixtable(Spanish,amos,ar,v,[per(l),pl(+),tns(past)]).
suffixtable(Spanish,aron,ar,v,[per(3),pl(+),tns(past)]).
suffixtable(Spanish,o,er,v,[per(l),pl(-),tns(pres)]).
suffixtable(Spanish,es,er,v,[per(2),pl(-),tns(pres)]).
suffixtable(Spanish,e,er,v,[per(3),pl(-),tns(pres)]).
suffixtable(Spanish,emos,er,v,[per(l),pl(+),tns(pres)]).
suffixtable(Spanish,en,er,v,[per(3),pl(+),tns(pres)]).
suffixtable(Spanish,ias,er,v,[per(2),pl(-),tns(past)]).
suffixtable(Spanish,ia,er,v,[per(3),pl(-),tns(past)]).
suffixtable(Spanish,iamos,er,v,[per(l),pl(+),tns(past)]).
suffixtable(Spanish,ian,er,v,[per(3),pl(+),tns(past)]).
suffixtable(Spanish,'i''',er,v,[per(l),pl(-),tns(past)]).
suffixtable(Spanish,iste,er,v,[per(2),pl(-),tns(past)]).
suffixtable(Spanish,'io''',er,v,[per(3),pl(-),tns(past)]).
suffixtable(Spanish,imos,er,v,[per(l),pl(+),tns(past)]).
suffixtable(Spanish,ieron,er,v,[per(3),pl(+),tns(past)]).
suffixtable(Spanish,o,ir,v,[per(l),pl(-),tns(pres)]).
suffixtable(Spanish,es,ir,v,[per(2),pl(-),tns(pres)]).
suffixtable(Spanish,e,ir,v,[per(3),pl(-),tns(pres)]).

123
suffixtable (spanish, imos, ir, v, [per(1), pl(+), tns(pres)]).
suffixtable (spanish, en, ir, v, [per(3), pl(+), tns(pres)]).

suffixtable (spanish, ias, ir, v, [per(2), pl(-), tns(past)]).
suffixtable (spanish, ia, ir, v, [per(3), pl(-), tns(past)]).
suffixtable (spanish, ia, ir, v, [per(1), pl(-), tns(past)]).
suffixtable (spanish, iamos, ir, v, [per(1), pl(+), tns(past)]).
suffixtable (spanish, ian, ir, v, [per(3), pl(+), tns(past)]).

suffixtable (spanish, 'i''', ir, v, [per(1), pl(-), tns(past)]).
suffixtable (spanish, iste, ir, v, [per(2), pl(-), tns(past)]).
suffixtable (spanish, 'io''', ir, v, [per(3), pl(-), tns(past)]).
suffixtable (spanish, imos, ir, v, [per(1), pl(+), tns(past)]).
suffixtable (spanish, ieron, ir, v, [per(3), pl(+), tns(past)]).

suffixtable (spanish, 'e''', '', v, [per(1), pl(-), tns(fut)]).
suffixtable (spanish, 'a''s', '', v, [per(2), pl(-), tns(fut)]).
suffixtable (spanish, 'a''', '', v, [per(3), pl(-), tns(fut)]).
suffixtable (spanish, 'ara'mos', ar, v, [per(1), pl(+), tns(fut)]).
suffixtable (spanish, 'e'mos', '', v, [per(1), pl(+), tns(fut)]).
suffixtable (spanish, 'a''n', '', v, [per(3), pl(+), tns(fut)]).

suffixtable (spanish, ado, ar, v, [perf(+)]).
suffixtable (spanish, ido, er, v, [perf(+)]).
suffixtable (spanish, ido, ir, v, [perf(+)]).

suffixtable (spanish, ando, ar, v, [prog(+)]).
suffixtable (spanish, iendo, er, v, [prog(+)]).
suffixtable (spanish, iendo, ir, v, [prog(+)]).

suffixtable (spanish, o, er, infl1, [per(1), pl(-), tns(pres)]).
suffixtable (spanish, es, er, infl1, [per(2), pl(-), tns(pres)]).
suffixtable (spanish, e, er, infl1, [per(3), pl(-), tns(pres)]).
suffixtable (spanish, emos, er, infl1, [per(1), pl(+), tns(pres)]).
suffixtable (spanish, en, er, infl1, [per(3), pl(+), tns(pres)]).

suffixtable (spanish, o, '', stem1, [per(1), pl(-), tns(pres)]).
suffixtable (spanish, as, '', stem1, [per(2), pl(-), tns(pres)]).
suffixtable (spanish, a, '', stem1, [per(3), pl(-), tns(pres)]).
suffixtable (spanish, an, '', stem1, [per(3), pl(+), tns(pres)]).

suffixtable (spanish, o, '', stem2, [per(1), pl(-), tns(pres)]).
suffixtable (spanish, es, '', stem2, [per(2), pl(-), tns(pres)]).
suffixtable (spanish, e, '', stem2, [per(3), pl(-), tns(pres)]).
suffixtable (spanish, en, '', stem2, [per(3), pl(+), tns(pres)]).

suffixtable (spanish, e, '', stem3, [per(1), pl(-), tns(past)]).
suffixtable (spanish,iste, '', stem3, [per(2), pl(-), tns(past)]).
suffixtable (spanish, o, '', stem3, [per(3), pl(-), tns(past)]).
suffixtable (spanish, imos, '', stem3, [per(1), pl(+), tns(past)]).
suffixtable (spanish, eron, '', stem3, [per(3), pl(+), tns(past)]).
suffixtable(Spanish,'i'' ','stem4,[per(1),pl(-),tns(past)]).
suffixtable(Spanish,iste, '',stem4,[per(2),pl(-),tns(past)]).
suffixtable(Spanish,'io'' ','stem4,[per(3),pl(-),tns(past)]).
suffixtable(Spanish,imos, '' ,stem4,[per(1),pl(+),tns(past)]).
suffixtable(Spanish,ieron, '' ,stem4,[per(3),pl(+),tns(past)]).

suffixtable(Spanish,'e'' ','stem5,[per(1),pl(-),tns(fut)]).
suffixtable(Spanish,'a'' ','stem5,[per(2),pl(-),tns(fut)]).
suffixtable(Spanish,'a'' ','stem5,[per(3),pl(-),tns(fut)]).
suffixtable(Spanish,'e''mos'' ','stem5,[per(1),pl(+),tns(fut)]).
suffixtable(Spanish,'a''n'' ','stem5,[per(3),pl(+),tns(fut)]).

suffixtable(Spanish,a, '' ,art,[fem(+)]).
suffixtable(Spanish,a,o,art,[fem(+)]).
suffixtable(Spanish,as, '' ,art,[fem(+),pl(+)]).
suffixtable(Spanish,as,os,art,[fem(+),pl(+)]).
suffixtable(Spanish,es, '' ,art,[pl(+)]).
suffixtable(Spanish,os, '' ,art,[pl(+)]).
suffixtable(Spanish,s, '' ,art,[pl(+)]).

FUNCTION WORDS

/* */
dict(Spanish,algun,art,[pl(algunos),english(some)]).
dict(Spanish,algunos,art,[f(pl(+)),root(algun)]).
dict(Spanish,cada,art,[f(fem(_)),english(each,every)])).
dict(Spanish,'cua''1',art,[f([wh(+),fem(_),pl(-)]),
    pl('cua''1'),english(wh(1))])).
dict(Spanish,'cua''les',art,[f([wh(+),fem(_),pl(+)]),
    root('cua''1')]).
dict(Spanish,'cua''nto',art,[f([wh(+),pl(-)]),
    english(how_much)]).
dict(Spanish,'cua''ntos',art,[f([wh(+),pl(+)]),
    english(how_many)]).
dict(Spanish,el,art,[fem(la),pl(los),english(the)]).
dict(Spanish,esa,art,[f(fem(+)),root(ese)]).
dict(Spanish,ese,art,[fem(esa),english(that)]).
dict(Spanish,esos,art,[f(pl(+)),english(those)]).
dict(Spanish,esta,art,[f(fem(+)),root(este)]).
dict(Spanish,este,art,[fem(esta),pl(estos),english(this)]).
dict(Spanish,estos,art,[f(pl(+)),english(thes[e])]).
dict(Spanish,la,art,[f(fem(+)),root(el)]).
dict(Spanish,las,art,[f([fem(+),pl(+)]),root(el)]).
dict(Spanish,los,art,[f(pl(+)),root(el)]).
dict(Spanish,mi,art,[f(fem(_)),english(my)]).
dict(Spanish,muchos,art,[f(pl(+)),english(many)]).
dict(Spanish,ningun,art,[english(no)]).
dict(Spanish,nuestro,art,[english(our)]).
dict(Spanish,otro,art,[english(another)]).
dict(Spanish,otros,art,[f(pl(+)),english(other)]).
dict(Spanish,'que'',art,[f(wh(+)),english(what)]).
dict(Spanish,su,art,[f(fem(_)),english([his,her,its,their,
    your])]).
dict(Spanish,tu,art,[f(fem(_)),english(your)]).
dict(Spanish,un,art,[english(a)]).
dict(Spanish,unos,art,[f(pl(+)),english(some)]).
dict(spanish,varios,art,[f(pl(+)),english(several))].

dict(spanish,que,comp,[english(that)])].

dict(spanish,haber,perf,[english(have)])].

dict(spanish,he,perf,[f([tns(pres),per(l),pl(-)]),
        root(haber)])].

dict(spanish,has,perf,[f([tns(pres),per(2),pl(-)]),
        root(haber)])].

dict(spanish,ha,perf,[f([tns(pres),per(3),pl(-)]),
        root(haber)])].

dict(spanish,hemos,perf,[f([tns(pres),per(1),pl(+)]),
        root(haber)])].

dict(spanish,han,perf,[f([tns(pres),per(3),pl(+)]),
        root(haber)])].

dict(spanish,habias,perf,[f([tns(past),per(2),pl(-)]),
        root(haber)])].

dict(spanish,habia,perf,[f([tns(past),per(3),pl(-)]),
        root(haber)])].

dict(spanish,habia,perf,[f([tns(past),per(1),pl(-)]),
        root(haber)])].

dict(spanish,habiamos,perf,[f([tns(past),per(1),pl(+)]),
        root(haber)])].

dict(spanish,habian,perf,[f([tns(past),per(3),pl(+)]),
        root(haber)])].

dict(spanish,'habre''',perf,[f([tns(fut),per(1),pl(-)]),
        root(haber)])].

dict(spanish,'habra''s',perf,[f([tns(fut),per(2),pl(-)]),
        root(haber)])].

dict(spanish,'habra''',perf,[f([tns(fut),per(3),pl(-)]),
        root(haber)])].

dict(spanish,'habre''mos',perf,[f([tns(fut),per(1),pl(+)]),
        root(haber)])].

dict(spanish,'habra''n',perf,[f([tns(fut),per(3),pl(+)]),
        root(haber)])].

dict(spanish,estar,prog,[english(be)])].

dict(spanish,estoy,prog,[f([tns(pres),per(l),pl(-)]),
        root(estar)])].

dict(spanish,'esta''s',prog,[f([tns(pres),per(2),pl(-)]),
        root(estar)])].

dict(spanish,'esta''',prog,[f([tns(pres),per(3),pl(-)]),
        root(estar)])].

dict(spanish,estamos,prog,[f([tns(pres),per(l),pl(+)]),
        root(estar)])].

dict(spanish,estan,prog,[f([tns(pres),per(3),pl(+)]),
        root(estar)])].

dict(spanish,estado,prog,[f([perf(+)]),root(estar)])].

dict(spanish,estabas,prog,[f([tns(past),per(2),pl(-)]),
        root(estar)])].

dict(spanish,estaba,prog,[f([tns(past),per(3),pl(-)]),
        root(estar)])].

dict(spanish,estaba,prog,[f([tns(past),per(1),pl(-)]),
        root(estar)])].

dict(spanish,estabamos,prog,[f([tns(past),per(l),pl(+)]),
        root(estar)])].
dict(spanish, estaban, prog, [f([tns(past), per(3), pl(+)]),
    root(estar)]).

dict(spanish, 'estare''s', prog, [f([tns(fut), per(4), pl(-)]),
    root(estar)])

dict(spanish, 'estara''s', prog, [f([tns(fut), per(2), pl(-)]),
    root(estar)])

dict(spanish, 'estara''s', prog, [f([tns(fut), per(3), pl(-)]),
    root(estar)])

dict(spanish, 'estara''n', prog, [f([tns(fut), per(2), pl(-)]),
    root(estar)])

dict(spanish, 'estare''mos', prog, [f([tns(fut), per(1), pl(+)]),
    root(estar)])

dict(spanish, 'estara''n', prog, [f([tns(fut), per(3), pl(+)]),
    root(estar)])

...
dict(Spanish, diciendo, v, [f([tns(pres), per(1), pl(-)]), root(decir)]).

dict(Spanish, digo, v, [f([tns(pres), per(3), pl(-)]), root(decir)].

dict(Spanish, digo, v, [f([tns(pres), per(1), pl(-)]), root(decir)].

dict(Spanish, dij, stem3, [f([tns(past), per(3), pl(-)]), root(dar)].

dict(Spanish, duerm, stem2, [f([tns(pres), per(1), pl(-)]), root(dormir)].

dict(Spanish, dormir, v, [english(sleep), prog(durmiendo), stem2(duerm), stem4(durm)]).
dict(spanish,tiempo,n,[english(time)]).
dict(spanish,tien,stem2,[f(tns(pres)),root(tener)]).
dict(spanish,tuv,stem3,[f(tns(past)),root(tener)]).
dict(spanish,tuvieron,v,[f([tns(past),per(3),pl(+)])],
   root(tener)]).
dict(spanish,tuvimos,v,[f([tns(past),per(1),pl(+)])],
   root(tener)]).
dict(spanish,v,stem1,[f(tns(pres)),root(ir)]).
dict(spanish,v,stem4,[f(tns(past)),root(ver)]).
dict(spanish,vamos,v,[f([tns(pres),per(1),pl(+)])],root(ir)]).
dict(spanish,vender,v,[english(sell)]).
dict(spanish,vendr,stem5,[f(tns(fut)),root(venir)]).
dict(spanish,vengo,v,[f([per(1),pl(-)],tns(pres))],
   root(venir))].
dict(spanish,verir,v,[irr([vengo,vinieron,viniendo]),
   stem2(vien),stem3(vin),stem5(vendr),english(come)]).
dict(spanish,veo,v,[f([tns(pres),per(1),pl(-)])],root(ver)]).
dict(spanish,ver,v,[english(see),perf(visto),irr([veo,vio]),
   stem4(v)]).
dict(spanish,verdad,n,[f(fem(+)),english(truth)]).
dict(spanish,vien,stem2,[f(tns(pres)),root(venir)]).
dict(spanish,viniendo,v,[f(prog(+)),root(venir)]).
dict(spanish,vinieron,v,[f([tns(past),per(3),pl(+)])],
   root(venir)]).
dict(spanish,vio,v,[f([tns(past),per(3),pl(-)])],root(ver)]).
dict(spanish,visto,v,[f(perf(+)),root(ver)]).
dict(spanish,voy,v,[f([tns(pres),per(1),pl(-)])],root(ir)]).
dict(spanish,yendo,v,[f(prog(+)),root(ir)]).
Appendix D. Prolog Code - EXEC

/* EXECUTIVE */
gb :-
    readinput(SL,TL,Surface), !,
    morph(SL,Surface, NewSurface), !,
    parse(SL,NewSurface, Sstructure),
    write('...Parsed'), nl, !,
    rtransformation(SL,Sstructure, Dstructure),
    write('...R-transformed'), nl,
    show(ds,'D-Structure =',Dstructure), !,
    bagof(TDstructure,
        translate(SL,Dstructure,TL, TDstructure),
        TDstructures),
    write('...Translated'), nl, !,
    transform_and_generate(TL,TDstructures).

/* CATEGORIAL ATTRIBUTES */
is_trace(\[[X,F],e\])  :- member(indx(_),F).
is_nptrace(\[[n,F],e\])  :- member(ana(+),F).
is_variable(\[[X,F],e\])  :- member(ana(-),F), member(pro(-),F).
is_pro(\[[n,FN],e\])  :- member(pro(+),FN),
        not member(indx(_),FN).
is_PRO(\[[n,\[\]],e\]).

is_sdeleter(L,W,C)  :- extractLocal(L,W,C,_,sdel, [+]).
is_transitive(L,V)  :- extract(L,V,v,_,subcat,[n]).

lexical(_\[-\],e)  :- !,fail.
lexical(e)  :- !,fail.
lexical(_).

index(\[[X,[indx(I)|\[\]]|\[\]],I\].

features(\[[\_\_,F]|\[\]]|\[\],F).

/* AUXILIARY PREDICATES */
extractLocal(L,W,C,RW,Funct,Args) :-
dict(L,W,C,Lexinfo),
    extractFunc(Lexinfo,Funct,AW),
    extractRoot(L,W,C,Lexinfo,RW,_,Funct,AR),
    combineF(AW,AR,Args).
/* Extract Args from current word, Root, and English root */

extract(L,W,C,RW,Funct,Args) :-
  dict(L,W,C,Lexinfo),
  extractFunc(Lexinfo,Funct,AW),
  extractRoot(L,W,C,Lexinfo,RW,Funct,AR),
  extractEngl(L,C,RLexinfo,Funct,AE),
  combineF(AW,AR,Ax),
  combineF(Ax,AE,Args).

/* Extract Args from given lexical info */
extractFunc(Lexinfo,Funct,Args) :-
  X =.. [Funct,A],
  member(X,Lexinfo),
  makelist(A,Args).
extractFunc(Lexinfo,Funct,[]) :-
  X =.. [Funct,],
  not member(X,Lexinfo).

/* Extract Args from root word, if one exists */
extractRoot(L,W,C,Lexinfo,Root,RLexinfo,Funct,Args) :-
  member(root(Root),Lexinfo), !,
  diet(L,Root,C,RLexinfo),
  not member(root(_),RLexinfo),
  extractFunc(RLexinfo,Funct,Args).
extractRoot(_,W,_,Lex,W,Lex,_,[]).

/* For Spanish words, extract Args from its English translation, which must exist. */
extractEngl(spanish,C,RLexinfo,Funct,Args) :-
  member(english(expr([[w|Expr]]),RLexinfo),
  dict(english,W,C,ELexinfo),
  member(expr([[Expr|Expr_info]]),ELexinfo),
  extractFunc(Expr_info,Funct,Args).
extractEngl(spanish,C,RLexinfo,Funct,Args) :-
  member(english(EW),RLexinfo),
  makelist(EW,EWlist),
  member(EW1,EWlist),
  dict(english,EW1,C,ELexinfo),
  not member(root(_),ELexinfo),
  extractFunc(ELexinfo,Funct,Args).
extractEngl(english,_,[],[]).

/* */
get_expr(L,C,expr([[w|Expr]]),Args,[w|S1],S2) :-
  rootword(L,W,C, RW),
  dict(L,RW,C,Lexinfo),
  member(expr([[Expr|Expr_info]]),Lexinfo),
  expr_member(Expr,S1, S2),
  extractEngl(L,C,Expr_info,subcat, Args).

expr_member([],S,S).
expr_member([[w|X]],[[w|Y]],Z) :- expr_member(X,Y,Z).
/* */
rootword(L,W,C,RootW) :-
  dict(L,W,C,Lexinfo),
member(root(R),Lexinfo), !,
RootW = R.
rootword(L,W,C,W) :- dict(L,W,C,_).
/**--------------------------------------------------------------------------*/
combineF([],F,F) :- !.
combineF(F,[],F) :- !.
combineF(WF,[Rl|Rs],F) :- /* if Rl already in WF, */
    mbrtype(Rl,WF), /* WF takes precedence */
    combineF(WF,Rs,F).
combineF(WF,[Rl|Rs],[Rl|F]) :- /* add Rl to WF if not */
    not mbrtype(Rl,WF), /* already in WF */
    combineF(WF,Rs,F).

mbrtype(R,W) :- /* for 'f' features */
    R =.. [RA,_,], !,
    X =.. [RA,_,],
    member(X,W).
mbrtype(R,W) :- member(R,W). /* for 'subcat' */
/**--------------------------------------------------------------------------*/
must_appear([],_):- !. /* ALL members of Arg1 */
must_appear([F1|Fs],F2) :- /* must be in Arg2 */
    member(F1,F2), !,
    must_appear(Fs,F2).

must_agree([],_). /* If Arg1 is in Arg2, */
must_agree([F1|Fs],F2) :- /* it must agree */
    featureagree(F1,F2),
    must_agree(Fs,F2).

featureagree(_,[]). featureagree(F1,[F2|_]) :-
    F1=..[A1,V1], F2=..[A1,V2], !, V1=V2.
featureagree(F1,[F2|_]) :- featureagree(F1,F2).
/**--------------------------------------------------------------------------*/
feature_union(F1,F2,F3) :- must_agree(F1,F2), union(F1,F2,F3).
/**--------------------------------------------------------------------------*/
get(Ftype,Flist,[F]) :- F=..[Ftype,_,], member(F,Flist), !.
get(_,_,[]).
/****************************************************************************/
*/ GENERAL PURPOSE PREDICATES */
/****************************************************************************/
member(X,[X]).
member(X,[_|Y]) :- member(X,Y).

union(X,[],X) :- !.
union([],X,X) :- !.
union([X|R],Y,Z) :- member(X,Y), !, union(R,Y,Z).
union([X|R],Y,[X|Z]) :- union(R,Y,Z).

append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

subtract(L,[],L) :- !.
subtract([H|T],L,U) :- member(H,L), !, subtract(T,L,U).
subtract([H|T],L,[H|U]) :- !, subtract(T,L,U).
subtract(_, _, []). 

makelist([], []) :- !.
makelist([H|T], [H|T]) :- !.
makelist(X, [X]).

^**********************************************************^
/* PRETTY-PRINTING */
^**********************************************************/
show(Type, Label, Tree) :-
    showon(Type),
    nl, display(Label), nl,
    prettyprint(Tree, 0), nl, nl.
show(_, _, _).

prettyprint(e, _). /* don't print empty categories */
prettyprint([], _).
prettyprint([[X,F]|T], I) :-
    nl, tab(I), display(X),display(' '),display(F),
    J is I+2,
    ppx(T, J).
prettyprint([H|T], I) :- tab(I),
    prettyprint(H, I),
    prettyprint(T, I).
prettyprint(X, I) :- nl, tab(I), display(X).

ppx([], _).
ppx([H|T], I) :- prettyprint(H, I), ppx(T, I).
Appendix E. Prolog Code - READINPUT

/*********************************************/
/** READINPUT */
/*********************************************/
readinput(SLang,TLang,Surface) :-
  display('Enter source language: '),
  read(Input),
  readinput2(Input,SLang,TLang,Surface).
readinput2('ds',SLang,TLang,Surface) :-
  assert(showon(ds)),
  readinput(SLang,TLang,Surface).
readinput2('nds',SLang,TLang,Surface) :-
  retract(showon(ds)),
  readinput(SLang,TLang,Surface).
readinput2(SLang,SLang,TLang,Surface) :-
  display('Enter target language: '),
  read(TLang),
  display('Enter sentence in '),
  display(SLang),
  display(': '),
  read_in(Surface).

/*********************************************/
/* Read sentence: Clocksin & Mellish p.87-88 */
read_in([W|Ws]) :-
  getO(C), readword(C,W,Cl), restsent(W,Cl,Ws).

readword(C,W,Cl) :-
  single_character(C), !,
  name(W,[C]),
  getO(Cl).
readword(C,W,C2) :-
  in_word(C,NewC), !,
  getO(Cl),
  restword(Cl,Cs,C2),
  name(W,[NewC|Cs]).
readword(_,W,C2) :- getO(Cl), readword(Cl,W,C2).

restword(C,[NewC|Cs],C2) :-
  in_word(C,NewC), !,
  getO(Cl),
  restword(Cl,Cs,C2).
restword(C,[],C).
restword(W,_,[]) :- lastword(W), !.
restsent(_,_,[Ws]) :- readword(C,Wl,Cl),
  restsent(Wl,Cl,Ws).

in_word(C,C) :- C>96, C<123. /* a ... z */
in_word(C,L) :- C>64, C<91, L is C+32. /* A ... Z */
in_word(C,C) :- C>47, C<58. /* 0 1...9 */
in_word(39,39). /* ' */
lastword('.').
lastword('?').

single_character(63).  /* ? */
single_character(46).  /* . */
Appendix F. Prolog Code – MORPH

/**************************************************************************
/*  V
/* MORPHOLOGICAL ANALYSIS
/*  V
/**************************************************************************
morph(L,[],[]).
morph(L,Surface,Newsurface) :-
    contract(L,Surface,Surfacel),
    collocate(L,Surfacel,Newsurface),
    morphl(L,Newsurface).

morph([_],[],[]).
morph([w|S]) :- morph_word(L,W), morphL(L,S).

morph_word(_,'.').
morph_word(_,?').
morph_word(L,W) :- dict(L,W,_,_).
morph_word(L,W) :- suffixanalysis(L,W).
morph_word(L,W) :- update_dict(L,W).

/**************************************************************************
/* */
contract(_,_,[]).
contract(L,[A|S1],[B,C|S2]) :- contraction(L,A,[B,C]),
    contract(L,SI,S2).
contract(L,[x|SI],[x|S2]) :- contract(L,SI,S2).

collocate(_,_,[]).
collocate(L,[A,B,C,D|S1],[E|S2]) :- collocation(L,[A,B,C,D],E),
    collocate(L,SL,S2).
collocate(L,[A,B|S1],[C|S2]) :- collocation(L,[A,B],C),
    collocate(L,SL,S2).
collocate(L,[X|S1],[X|S2]) :- collocate(L,SL,S2).

/**************************************************************************
suffixanalysis(L,W) :-
suffixtable(L,Sfx,RootEnd,SfxC,SfxF),
    split(Stem,Sfx,W),
    split(Stem,RootEnd,Root),
    addinf1(L,W,Root,SfxC,SfxF), !.

    /* category type: stem_ */
addinf1(spanish,W,Root,stem,SfxF) :-
    name(Stem,[S,T,E,M|_]),
    name(stem,[S,T,E,M]),
    rootword(spanish,Root,Stem,RootW), !,
    dict(spanish,RootW,C,_), /* root is modal or verb */
addinf1(L,W,Root,C,SfxF) :-
    extractLocal(L,Root,C,Root,f,RootF),
    combineF(SfxF,RootF,F),
    adddict(L,W,Root,C,F),
    !,fail.
split(Stem,[],Stem) :- !. /* for stems with null endings */
split(Stem,Sfx,W) :-
  var(Stem), !, /* Find Stem, given W */
  name(W,Wlist),
  name(Sfx,Sfxlist),
  append(Stemlist,Sfxlist,Wlist),
  name(Stem,Stemlist).

split(Stem,Sfx,W) :- /* Find W, given Stem */
  name(Stem,Stemlist),
  name(Sfx,Sfxlist),
  append(Stemlist,Sfxlist,Wlist),
  name(W,Wlist).

adddict(L,W,RootW,C,F) :-
  uniqueadd(L,W,C,RootW,F),
  assert(dict(L,W,C,[f(F),root(RootW)])),
  display('Added:'), nl,
  display(dict(L,W,C,[f(F),root(RootW)])), nl.

uniqueadd(L,W,C,_,_) :- not dict(L,W,C,_,_).
uniqueadd(L,W,C,RootW,_) :-
  dict(L,W,C,Lexinfo),
  not member(root(RootW),Lexinfo).
uniqueadd(L,W,C,F) :-
  dict(L,W,C,Lexinfo),
  member(f(FI),Lexinfo),
  makelist(Fl,Fllist),
  not union(Fllist,[],F).

update_dict(L,W) :-
  not dict(L,W,_,_),
  display('Do you want to add "'),
  display(W),
  display('"? '),
  read(Reply),
  process_reply(L,Reply,W).

process_reply(Y,y,W) :- /* Yes, add to dictionary */
  display('Enter category: '), read(C),
  display('Enter lexical info: '), read(Lexinfo),
  assert(dict(L,W,C,Lexinfo)),
  display('Perm add:'), nl,
  display(dict(L,W,C,Lexinfo)), nl,
  tell(L),
  write(dict(L,W,C,Lexinfo)), put(46), nl,
  tell(user).

process_reply(L,t,W) :- /* Temp add to dictionary */
  display('Enter category: '), read(C),
  display('Enter lexical info: '), read(Lexinfo),
  assert(dict(L,W,C,Lexinfo)),
  display('Temp add:'), nl,
  display(dict(L,W,C,Lexinfo)), nl.

process_reply(L,r,W) :- /* Add Root to dictionary */
display('Enter root: '), read(Root),
display('Enter category: '), read(C),
display('Enter lexical info: '), read(Lexinfo),
assert(dict(L,Root,C,Lexinfo)),
display('Root add:'), nl,
display(dict(L,Root,C,Lexinfo)), nl,
tell(L),
write(dict(L,Root,C,Lexinfo)), put(46), nl,
tell(user),
morph_word(L,W). /* now re-morph word */

process_reply(_,_,_) :- abort.
Appendix G. Prolog Code - Core Grammars

/*******************************/
/*
/* CORE GRAMMAR: English */
/*
/*******************************/
null_subject(english, /* Base to Surface */
[[comp,FC],WH,[e,COMP, [[infl,FI],BaseNP,INFL]|Post],
[[comp,FC],WH,[e,COMP, [[infl,FI],[[n,FN],e,[e,N]],INFL]|Post]] :-
    not var(BaseNP),
    not lexical(BaseNP),
    not is_trace(BaseNP),
    not member(tns(-),FI), !, /* EC is not PRO */
    english_pronoun(N),
    assign_features(english,N,FI,FN).

null_subject(english,COMP,COMP). /* Surface to Base */

assign_features(L,N,FI,FN) :-
    extractLocal(L,N,n,_,f,Fl),
    defaultF(L,n,Fl,FN), !,
    must_agree(FI,FN).

english_pronoun(he).
english_pronoun(she).
english_pronoun(it).
english_pronoun(they).

ecm(english,V) :- /* Exceptional Case Marking */
    is_sdeleter(english,V,v), /* sbar-deleter */
    is_transitive(english,V). /* & transitive */

proper_governor(english,v).
proper_governor(english,p).

bounding_cat(english,infl).
bounding_cat(english,n).

/*****************************/
/* INSERTION RULES */
/*****************************/
it_insertion(english, /* base to surface */
[[comp,FC],WH,[e,COMP, [[infl,BaseFI],[[n,[]],e],INFL]|Post],
[[comp,FC],WH,[e,COMP, [[infl,FI],[[n,FN],e,[e,it]],INFL]|Post]]) :-
    var(FN),
    member(theta(-),BaseFI),
    makelist([per(3),pl(-)],FN),
    union(BaseFI,FN,FI), !.

it_insertion(english, /* surface to base */
[[comp,FC],WH,[e,COMP, [[infl,FI],[[n,[]],e],INFL]|Post],
[[comp,FC],WH,[e,COMP, [[infl,FI],[[n,FN],e,[e,it]],INFL]|Post]]) :-
not \text{var}(FN),
\text{member}(\text{theta}(-),FI), !.
\text{it} \text{_insertion}(\_,\text{COMP},\text{COMP}).
/*---------------------------------------------*/
/* if non-matrix COMP is empty, and clause is tensed, */
/* assign a default lexical COMP. */
\text{comp} \text{_insertion}(\text{english},\text{matrix}(-),
[[\text{comp},\text{FC}],\text{WH},[e,e,\text{INFL}]]|\text{Post}],
[[\text{comp},\text{FC}],\text{WH},[e,\text{that},\text{INFL}]]|\text{Post}]):=
\text{features}(\text{INFL},FI),
\text{not} \text{member}(\text{tns}(-),FI).
\text{comp} \text{_insertion}(\text{english},\_,\text{COMP},\text{COMP}).
/*---------------------------------------------*/
\text{modal} \text{_insertion}(\text{english},[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,e,\text{VP}]]]|\text{Post}],
[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{to},\text{VP}]]]|\text{Post}]):=
\text{features}(\text{INFL},FI), !.
\text{modal} \text{_insertion}(\text{english},[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,e,\text{VP}]]]|\text{Post}],
[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{will},\text{VP}]]]|\text{Post}]):=
\text{features}(\text{INFL},FI), !.
\text{modal} \text{_insertion}(\_,\_,\_,\text{COMP},\text{COMP}).
/*---------------------------------------------*/
\text{do} \text{support}(\text{english},\text{matrix}(\_),\text{ques},
[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,e,\text{VP}]]]|\text{Post}],
[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{to},\text{VP}]]]|\text{Post}]):=
\text{lexical}(\text{NP}),
\text{not} \text{member}(\text{tns}(\text{fut}),FI),
\text{not} \text{member}(\text{perf}(\_),FI),
\text{not} \text{member}(\text{prog}(\_),FI),
\text{not} \text{member}(\text{pass}(\_),FI), !.
\text{do} \text{support}(\_,\_,\_,\_,\text{COMP},\text{COMP}).
/*---------------------------------------------*/
/* INVERSION RULES */
/*---------------------------------------------*/
\text{adjunct}(\text{english},W) \rightarrow [W], \{ \text{dict}(\text{english},W,\text{infl},\_), !,
\text{not} \text{W}=\text{to} \}.
\text{adjunct}(\text{english},W) \rightarrow [W], \{ \text{dict}(\text{english},W,\text{perf},\_) \}.
\text{adjunct}(\text{english},W) \rightarrow [W], \{ \text{dict}(\text{english},W,\text{prog},\_) \}.
\text{inversion}(\text{english},\text{Matrix},\text{Mode},\text{BaseCOMP},\text{SurfaceCOMP}):=
\text{havebe} \text{raising}(\text{BaseCOMP},\text{COMP}1),
\text{sai} (\text{Matrix},\text{Mode},\text{COMP}1,\text{SurfaceCOMP}).
\text{rinversion}(\text{english},\text{SurfaceCOMP},\text{BaseCOMP}):=
\text{sai}(\_,\_,\text{COMP}1,\text{SurfaceCOMP}),
\text{havebe} \text{raising}(\text{BaseCOMP},\text{COMP}1).
/* Subject-AUX-Inversion */
\text{sai}(\text{matrix}(\_),\text{ques},[[\text{comp},\text{FC}],\text{WH},[e,\text{COMP},
[[\text{infl},\text{FI}],\text{NP},[e,\text{INFL},\text{VP}]]]|\text{Post}],{}}
var(N),
not lexical(NP),
(member(per(1),FI); member(per(2),FI)),
spanish_pronoun(N),
assign_features(spanish,N,FI,FN).
null_subject(spanish,COMP,COMP).
null_subject(spanish,  
[[comp,PC],WH,[e,COMP,  
[[inf1,FI],[[n,FI],[e,[e,N]],INFL]]|Post]],  
[[comp,PC],WH,[e,COMP,  
[[inf1,FI],[[n,FI],[e],INFL]]|Post]) :-
not var(N),
member(pro(+),FN). /* delete if lexical pronoun */

spanish_pronoun(yo).
spanish_pronoun(nosotros).
spanish_pronoun('tu').

proper_governor(spanish,v).
proper_governor(spanish,inf1).

bounding_cat(spanish,comp).
bounding_cat(spanish,n).

/*******************************
/* INSERTION RULES */
/*******************************
/* if non-matrix COMP is empty, and clause is tensed, */
/* assign a default lexical COMP. */
comp_insertion(spanish,matrix(-),
[[comp,PC],WH,[e,e,INFL]|Post],
[[comp,PC],WH,[e,que,INFL]|Post]) :-
features(INFL,FI),
not member(tns(-),FI), !.

comp_insertion(spanish,_,COMP, COMP).

/*******************************
/* RESTRUCTURING RULES */
/*******************************
adjunct(spanish,expr(W)) -->
restructure(W), { !, not W=[e,e,e,e] }.
restructure([e,e,e,v]) -- [v],
{ dict(spanish,V,v,Data),
  member(root(,),Data) }.
restructure([e,e,P2,V]) -- [P2,V],
{ dict(spanish,P2,prog,Data),
  member(root(,),Data),
  dict(spanish,V,v, ) }.
restructure([e,e,P2,e]) -- [P2],
{ dict(spanish,P2,prog,Data),
  member(root(,),Data) }.
restructure([e,P1,e,V]) -- [P1,V],
{ dict(spanish,P1,perf,Data),
  member(root(,),Data),
  dict(spanish,V,v, ) }.
restructure([e,P1,P2,V]) -- [P1,P2,V],
{ dict(spanish,P1,perf,Data),
member(root(_),Data),
dict(spanish,P2,prog,_)},
restructure([e,P1,P2,e]) -> [P1,P2],
{ dict(spanish,P1,perf,Data),
member(root(_),Data),
dict(spanish,P2,prog,_) }.
restructure([I,e,e,V]) -> [I,V],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,V,v,_) }.
restructure([I,e,P2,V]) -> [I,P2,V],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,P2,prog,_) }.
restructure([I,e,P2,e]) -> [I,P2,e],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,P2,prog,_) }.
restructure([I,P1,e,V]) -> [I,P1,V],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,P1,perf,_,)
}.
restructure([I,P1,P2,V]) -> [I,P1,P2],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,P1,perf,_,)
}.
restructure([I,P1,P2,e]) -> [I,P1,P2],
{ dict(spanish,I,inf1,Data),
member(root(_),Data),
dict(spanish,P1,perf,_,)
}.
restructure([I,e,e,e]) -> [I],
{ dict(spanish,I,inf1,Data),
member(root(_),Data) }.
/************ INVERSION RULES *************/
/*
/*****************************/
parse_inversion(Spanish,expr([_,'_','_','V']),
[[inf1,[],NP,[e,e,
[v,[]],e,e,[Args]|PostV]]]) -->
{ lexical(V),
extractLocal(spanish,V,v,_,f,FV) },
parse_subject(FV,DP),
parse_inversion2(V,Args),
postadjunct(spanish,v,PostV).
parse_inversion(L,_,INFL) --> x2(L,inf1,INFL).
parse_inversion2(V,Args) -->
{ extract(spanish,V,v,_,subcat,SC) },
arguments(spanish,SC,Args).
parse_inversion2(V,Args) -->
{ extract(spanish,V,v,_,subcat,SUBCAT) },
emptyArgs(spanish,SUBCAT,Args).

parse_subject(FV,NP) -->
x2(spanish,n,NP),
{ percolate(spanish,NP,[[n,FN]_]),
  must_agree(FN,FV) }.
parse_subject(FV,[[n,[]],e,[e,N]]) -->
[],
{ ( member(per(1),FV) ; member(per(2),FV) ), !,
  spanish_pronoun(N),
  assign_features(spanish,N,FV,_) }.
parse_subject(_,[[n,[]],e]) --> [].

inversion(spanish,_,_,BaseCOMP, SurfaceCOMP) :-
  member([comp,FC],BaseCOMP),
  member(wh(+),FC), !, /* oblig for wh- in COMP */
  v_preposing(BaseCOMP, SurfaceCOMP).
inversion(spanish,matrix(+),ques,BaseCOMP, SurfaceCOMP) :-
  !, v_preposing(BaseCOMP, SurfaceCOMP).
inversion(spanish,_,_,COMP,COMP).
rinversion(spanish,SurfaceCOMP, BaseCOMP) :-
  v_preposing(BaseCOMP, SurfaceCOMP), !.
rinversion(spanish,COMP,COMP).

v_preposing([[comp,FC],WH,[e,COMP],
  [[inf1,FI],NP,[e,BaseINFL],
  [[v,FV],BasePERF,
    [BasePROG,BaseVERB|Args]|PostV]]]],[PostC],
  [[comp,FC],WH,[e,COMP],
  [[inf1,FI],expr(Adjunct),
  [[inf1,FI],NP,[e,INFL],
  [[v,FV],PERF,
    [PROG,VERB|Args]|PostV]]]]],[PostC]) :-
  makelist([[BaseINFL,BasePERF,BasePROG,BaseVERB], Base],
    makelist([INFL,PERF,PROG,VERB], Surface),
    v_prepose(Base,Surface,Adjunct).

v_prepose([e,e,e,v],[e,e,e,e],[e,e,e,v]) :- !.
v_prepose([e,e,P2,V],[e,e,e,e],[e,e,P2,V]).
v_prepose([e,e,P2,V],[e,e,e,V],[e,e,P2,e]) :- !.
v_prepose([e,P1,e,V],[e,e,e,[e,P1,e,V]) :- !.
v_prepose([e,P1,P2,V],[e,e,e],[e,P1,P2,V]).
v_prepose([e,P1,P2,V],[e,e,e,V],[e,P1,P2,e]) :- !.
v_prepose([I,e,e,V],[e,e,e,[I,e,e,V]).
v_prepose([I,e,e,V],[e,e,e,V],[I,e,e,e]) :- !.
v_prepose([I,e,P2,V],[e,e,e,e],[I,e,P2,V]).
v_prepose([I,e,P2,V],[e,e,e,V],[I,e,P2,e]).
v_prepose([I,e,P2,V],[e,e,P2,V],[I,e,e,e]) :- !.
v_prepose([I,P1,e,V],[e,e,e,e],[I,P1,e,V]).
v_prepose([I,P1,e,V],[e,e,e,V],[I,e,e,e]) :- !.
v_prepose([I,P1,P2,V],[e,e,e,e],[I,P1,P2,V]).
v_prepose([I,P1,P2,V],[e,e,e,V],[I,P1,P2,e]).
v_prepose([I,P1,P2,V],[e,P1,P2,V],[I,e,e,e]) :- !.

/* DEFAULT FORMS */

default_tns(spanish,FI,FV,_, FI) :- member(tns(_),FI),
    not member(tns(_),FV).
default_tns(spanish,FI,FV,_, FI) :- member(tns(_),FV),
    not member(tns(_,FI)).
default_tns(spanish,FI,FV,_, [tns(-)|FI]) :-
    not member(tns(_),FI),
    not member(tns(_,FV)).

/* Default Lexical Features */
/* NOUNS */
defaultF(spanish,n,Fl,F4) :-
    default(per,3,FI,F2),
default(pl,-,F2,F3),
default(fem,-,F3,F4), !.
defaultF(spanish,art,Fl,F3) :-
    default(pl,-,Fl,F2),
default(fem,-,F2,F3), !.
defaultF(spanish,_,F,F).

/* Generate inflected auxiliaries */
gen_perf(spanish,_,Lexinfo,W) :- member(perf(W),Lexinfo).
gen_perf(spanish,V,_,W) :- split(Stem,ar,V),
    split(Stem,ado,W).
gen_perf(spanish,V,_,W) :- split(Stem,er,V),
    split(Stem,ido,W).
gen_perf(spanish,V,_,W) :- split(Stem,ir,V),
    split(Stem,ido,W).

genProg(spanish,_,Lexinfo,W) :- member(prog(W),Lexinfo).
genProg(spanish,V,_,W) :- split(Stem,ar,V),
    split(Stem,ando,W).
genProg(spanish,V,_,W) :- split(Stem,er,V),
    split(Stem,iendo,W).
genProg(spanish,V,_,W) :- split(Stem,ir,V),
    split(Stem,iendo,W).

/* AUXILIARY PREDICATES */
default(Feature,Value,Fin,Fout) :-
    F1 =.. [Feature,Value],
    F2 =.. [Feature,],
default2(F1,F2,Fin,Fout).
default2(_,F2,Fin,Fin) :- member(F2,Fin), !.
default2(F1,_,Fin,[F1|Fin]).
Appendix H. Prolog Code - XBAR

/* X-BAR SYSTEM */

parse(L,Surface,Sstructure) :-
    x2(L,comp,Sstructure,Surface,[]).

x2(_,_,_ --> [?], {!,fail}.
x2(_,_,_ --> [.], {!,fail}.

x2(L,infl,[[infl,[]],ADJU,INFL]) --> /* Inversion */
    adjunct(L,ADJU),
    parse_inversion(L,ADJU,INFL).

x2(L,C,[[C,[]],Spec,HD|Post]) -->
    specifier(L,C,Spec),
    xl(L,C,HD),
    postadjunct(L,C,Post).

x1(L,C,[Pre,HD|Complement]) -->
    preadjunct(L,C,Pre),
    x(L,C,HD,Args),
    arguments(L,Args,Complement).

x(L,C,W,Args) --> [W], { extract(L,W,C,_,subcat, Args) }.
x(_,comp,e,[infl]) --> [].
x(_,infl,e,[v]) --> [].

/* SPECIFIER STRUCTURES: */
specifier(L,comp,WH) --> wh_phrase(L,WH).
specifier(L,infl,WP) --> x2(L,n,WP).
specifier(_,infl,[[n,[]],e]) --> [1], {[}.
specifier(L,v,W) --> [W], { dict(L,W,art,_) }.
specifier(L,v,W) --> [W], { dict(L,W,perf,_) }.
specifier(_,_,e) --> [1]. /* optional */

/* ADJUNCT STRUCTURES: */
preadjunct(L,v,W) --> [W], { dict(L,W,prog,_) }.
preadjunct(_,e) --> [1]. /* optional */

postadjunct(_,n,[]) --> [1]. /* optional for NPs */
postadjunct(L,n,[PP|ADJU]) -->
    [1], x2(L,p,PP), {1}, postadjunct(L,n,ADJU).
postadjunct(L,v,[PP|ADJU]) ⊃
x2(L,p,PP), {!}, postadjunct(L,v,ADJU).

postadjunct(_,comp,[[mode,decl]]) ⊃ [!], {!}.
postadjunct(_,comp,[[mode,ques]]) ⊃ [?], {!}.
postadjunct(_,_,[[]]) ⊃ []. /* optional */

ARGUMENT STRUCTURES:

arguments(_,[],[[]]) ⊃ [].
arguments(L,[A1|As],[ARG1|ARGS]) ⊃
parseArg(L,A1,ARG1), arguments(L,As,ARGS).

parseArg(L,[Arg],TREE) ⊃ optionalArg(L,Arg,TREE).
parseArg(L,Arg,TREE) ⊃ x2(L,Arg,TREE).

optionalArg(L,Arg,TREE) ⊃ parseArg(L,Arg,TREE).
optionalArg(_,_,[[]]) ⊃ [].

emptyArgs(L,[A1|As],[Arg1|Args]) ⊃ parseArg(L,A1,Arg1),
emptyArgs(L,As,Args).
emptyArgs(L,n|As),[[n,[]],e]|Args)) ⊃
emptyArgs2(L,As,Args).
emptyArgs(L,p|As),[[p,[]],e]|Args)) ⊃
emptyArgs2(L,As,Args).

emptyArgs2(_,[],[[]]) ⊃ [].
emptyArgs2(L,[A1|As],[Arg1|Args]) ⊃ parseArg(L,A1,Arg1),
emptyArgs2(L,As,Args).

WH-PHRASE

wh_phrase(L,TREE) ⊃
x2(L,n,TREE), { wh_test(L,TREE) }.
wh_phrase(L,TREE) ⊃
x2(L,p,TREE), { wh_test(L,TREE) }.
wh_test(L,[[p,],_,[[],_,TREE]]) :- /* test for wh-NP */
percolate(L,TREE,TREE1), /* within PP */
features(TREE1,F),
member(wh(+),F).
wh_test(L,TREE) :-
percolate(L,TREE,TREE1),
features(TREE1,F), !,
member(wh(+),F).
/** REVERSE-TRANSFORMATIONS */

rtransformation(L, COMP1, COMP11) :-
  rinversion(L, COMP1, COMP2),
  percolate(L, COMP2, COMP3),
  rmove_affix(L, COMP3, COMP4),
  theta_extraction(COMP4, COMP5),
  do_support(L, _, COMP6, COMP5),
  it_insertion(L, COMP7, COMP6),
  null_subject(L, COMP8, COMP7),
  rmove_alpha(COMP8, COMP9),
  wh_feature_extraction(COMP9, COMP10),
  rtransform_lower(L, COMP10, COMP11).

rtransform_lower(L, [[comp, FC], WH, [e, COMP | Args1] | Post],
  [[comp, FC], WH, [e, COMP | Args2] | Post]) :-
  rtransform2(L, Args1, Args2).

rtransform2(L, [H1 | T1], [H2 | T2]) :-
  rtransform3(L, H1, H2),
  rtransform2(L, T1, T2).

rtransform3(L, [[comp, FC1] | COMP1], [[comp, FC2] | COMP2]) :-
  rtransformation(L, [[comp, FC1] | COMP1],
  [[comp, FC2] | COMP2]).

rtransform3(L, [[CP, FC], SC, [PC, C | Args1] | Post],
  [[CP, FC], SC, [PC, C | Args2] | Post]) :-
  rtransform2(L, Args1, Args2).

percolate(_, X, X) :- not lexical(X). /* empty NP or PP */

percolate(L, [[v, []], P1, [P2, V | Args1] | Post1],
  [[v, []], P1, [P2, V | Args2] | Post2]) :-
  perc_compl(L, Args1, Args2),
  perc_compl(L, Post1, Post2).

percolate(L, [[C, []], Spec1, HD1 | Post1],
  [[C, F], Spec2, HD2 | Post2]) :-
  perc_xl(L, C, HD1, HD2, FHD),
  perc_specword(L, C, Spec1, Spec2, FSC),
  feature_union(FSC, FHD, F),
  perc_compl(L, Post1, Post2).

perc_xl(L, C, [e, HD1 | Complement1],
  [e, HD2 | Complement2], F) :-
  perc_word(L, C, HD1, HD2, F),
  perc_compl(L, Complement1, Complement2).

perc_compl(_, [], []).
perc_compl(_, [[mode,M],[mode,M]]) :- !.
perc_compl(L,[H1|T1], [H2|T2]) :-
    perc_compl2(L,H1, H2),
    perc_compl(L,T1, T2).

perc_compl2(_, [[comp,FC]|COMP], [[comp,FC]|COMP]) :- !.
perc_compl2(L,X,Y) :- percolate(L,X,Y).
/* ------------------------------- */
perc_word(_, _, e, e, []) :- !. /* empty word */
perc_word(L, _, [X,_]|Y], [[X,F]|Z], F) :-
    !, percolate(L, [[X, ]|Y], [[X,F] Z]), !.
perc_word(L,C, expr([W|T]), expr([RW|T]), F) :- /* expr'n */
    !,extractLocal(L,W,C,RW,f, Fl),
defaultF(L,C,Fl, F).
perc_word(L,C,W, RW,F) :- /* lexical word */
    !,extractLocal(L,W,C,RW,f, F),
defaultF(L,C,Fl, F).
/*---------------------------------------------*/
perc_specword(_, _, e, e, []) :- !.
perc_specword(L,inf1,W,RW,F) :- percolate(L,_,W,RW,F).
/*------------------- */
/* REVERSE AFFIX MOVEMENT */
/*------------------- */
rmove_affix(L, [[comp,FC],WH, [e,COMP, 
    [inf1,FI],NP,[e,INFL, 
    [v,[]],PERF,[PROG|V|Args]|PostV]])|PostC],
    [[comp,FC],WH, [e,COMP, 
    [inf1,BaseFI],NP,[e,INFL, 
    [v,FV],e,[e,BaseV|Args]|PostV]])|PostC)) :-
    rmove_participle(L,PERF,PROG,V, BaseV,FV),
default_tns(L,FI,FV,V,FI), /* language-dependent */
union(FI,FV,FI2), /* merge VP&INFL feat. */
features(NP,FNP),
feature_union(FNP,FI2,BaseFI). /* merge NP&INFL feat. */

rmove_participle(L,PERF,PROG,expr([H|T]), 
    expr([BaseH|T]),FV) :-
    !, rmove_participle(L,PERF,PROG,H,BaseH,FV).
    /* expressions */
    /!
    /!
    /* auxless, V infinitive */

rmove_participle(L,e,e,V,F) :-
dict(L,V,v,Lexinfo),
not member(root(_),Lexinfo),
extractFunc(Lexinfo,f,F).
    /* auxless, V inflected */

rmove_participle(L,e,e,V,BaseV,F) :-
    !,extractLocal(L,V,V,BaseV,f,FV),
member(tns(_),FV),
subtract(FV,[perf(+)],F). /* V=[tns(past),perf(+)] */
    /* P2 progressive */

rmove_participle(L,e,P2,V,BaseV,[prog(+)|F]) :-
    extractLocal(L,P2,prog,_,f,F),
rmvolve_comp(COMP,COMP).

/* COMP to Argument */
rmvolve_comp2(WH,[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,[P2,V|Argsl]|Post]]],
[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,[P2,V|Argsl]|Post]]) :-
  rmvolve_to_obj(WH,Argsl,Argsl2).
/* NP COMP to PP-adj */
rmvolve_comp2(WH,[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,VP|PostV]]],
[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,VP|PostV]]) :-
  rmvolve_pp(WH,PostV1,PostV2).
/* NP COMP to PP-arg */
rmvolve_comp2([n,PF]|NN],[[inf1,FI1],NP,INFL],
[[inf1,FI2],[n,PF]|NN],INFL)) :-
  not lexical(NP),
  features(WH,F),
  feature_union(F,FI1,F1).
/* COMP to COMP */
rmvolve_comp2(WH,[[comp,FC,e,[e,COMP,INFL]|Post]],
[[comp,FC],WH,[e,COMP,INFL]|Post]).
/* recursive search */
rmvolve_comp2(WH,[[X,F],A1,[A2,XW|Argsl]|Post],
[[X,F],A1,[A2,XW|Argsl]|Post]) :-
  rmvolve_comp3(WH,Argsl,Argsl2).
/* PP COMP added as adj */
rmvolve_comp2([p,FP]|PP],[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,VP|PostV]]],
[[inf1,FI1],NP,[e,INFL,
[v,FV],P1,VP,[p,FP]|PP]|PostV])].
rmvolve_comp3(WH,[H1|T],[H2|T]) :- rmvolve_comp2(WH,H1,H2).
rmvolve_comp3(WH,[H|T1],[H|T2]) :- rmvolve_comp3(WH,T1,T2).
rmvolve_to_obj(WH,[H|T],[WH|T]) :-
  not lexical(H), member([C,],H), member([C,],WH), !.
rmvolve_to_obj(WH,[H|T1],[H|T2]) :- rmvolve_to_obj(WH,T1,T2).
rmvolve_pp([n,PF]|NP],[[p,FP],e,[e,P,PNP]|Post],
[[p,FP],e,[e,P,[n,PF]|NP]|Post]) :-
  not lexical(PNP).
rmvolve_pp(NP,[X|P1],[X|P2]) :- rmvolve_pp(NP,P1,P2).
/*--------------------*/
rmvolve_subj([[comp,FC],WH,[e,COMP],
[[inf1,FI1],NP,[e,INFL,VPl]|Post]],
lex(NP),
  member(theta(-),FI), !,
  combineF([per(3),pl(-)],FI,BaseFI),
  remove_lower_subj(NP,VP1,VP2).

remove_subj(COMP,COMP).

/* SUBJ to OBJ */
remove_lower_subj(NP,[v,FV],Pl,[P2,V|Arg1]|PostV),
  remove_subj(NP,Arg1,Arg2).

/* SUBJ to SUBJ */
remove_lower_subj(NP,[n,|NP],Pl,[P2,V|Arg1]|Post),
  remove_lower_subj(NP,Arg1,Arg2).

remove_obj(NP,BaseNP,NP) :- not lexical(BaseNP).
remove_obj(NP,[H1|T1],[H2|T2]) :- remove_obj(NP,H1,H2).
remove_obj(NP,[H|T1],[H|T2]) :- remove_obj(NP,T1,T2).

wh_feature_extraction([p,F],e,[e,P,|NP]),
  features(NP,FNP),
  member(wh(+),FN), !.
wh_feature_extraction([H1|T1], [H2|T2]) :-
  wh_feature_extraction(H1,H2),
  wh_feature_extraction(T1,T2), !.
wh_feature_extraction(X,X).
Appendix J. Prolog Code - TRANSLATE

/* **************TRANSLATE***********************/
/* X2 --> Specifier X1 Postadjuncts */
translate(SL,[[C,F1],[Spec1,HD1|Post1], TL,[[C,F2],[Spec2,HD2|Post2]] : -
  transl_xl(SL,C,F1,HD1, TL,F2,HD2),
  transl_spec(SL,C,Spec1,TL,F2,Spec2),
  transl_comp(SL,Post1, TL,Post2).

/* X1 --> Preadjunct X Complements */
transl_xl(SL,C,F1,[Pre,HD1|Complement1], TL, F2,[Pre,HD2|Complement2]) : -
  transl_head(SL,C,Fl,HDl, TL,F2,HD2),
  transl_comp(SL,Complement1, TL,Complement2).

/* **************TRANSLATE***********************/
transl_head(_,_,F,e, _,F,e) :- !. /* empty head */
transl_head(_,comp,_,_, _,[],e) :- !. /* COMP --> e */
/* expression head */
transl_head(SL,C,Fl,expr([SLW|Expr]),  TL,F2,TLW) :-
  extractLocal(SL,SLW,C,_,expr,[Expr|Expr_info]),
  member(TLwords,Expr_info),
  TLwords =.. [TL,Words],
  makelist(Words,Wlist), !,
  pick_headlist(C,Wlist,Fl,TL,  F2,TLW).
  /* all others */
transl_head(SL,C,Fl,SourceW, TL,F2,TargetW) :-
  extractLocal(SL,SourceW,C,_,TL,TLwords), !,
  pick_headlist(C,TLwords,Fl,TL,F2,TargetW).
/*---------------------------------------------*/
pick_headlist(C,[Wl|_],Fl,  TL,F2,W2) :-
pick_headword(C,Wl,Fl,TL,F2,W2).
pick_headlist(C,[_|Ws],Fl,  TL,F2,W2) :-
pick_headlist(C,Ws,Fl,TL,F2,W2).
/*---------------------------------------------*/
pick_headword(_,e,F,_,  F,e) :- !.
pick_headword(_,expr(E),_,[],expr(E)) :- !.
pick_headword(n,Wl,F1,TL, F2,W3) :- /* transl  head noun */
  get(pl,F1,NUM),
  adjust(pl,n,TL,W1,NUM,W2), /* adjust for num */
  extractLocal(TL,W1,n,,f,F2a),
  defaultF(TL,n,F2a, F2b),
  combineF(F1,F2b, F2),
  capitalize_proper(TL,W2,W3), !.
pick_headword(infl,W,F,_,  F,W) :- !.
pick_headword(C,W,F1,TL, F2,W) :- /* transl other heads */
  extractLocal(TL,W,C,,f, F2a),
  defaultF(TL,C,F2a, F2b),
  combineF(F2b,F1, F2), !.
/* ** Translate specifier ** */
transl_spec(_,_,X,_,_,X) :- not lexical(X), !. /* empty */

transl_spec(SL,n,SourceW,TL,F,TargetW) :-
   extractLocal(SL,SourceW,n,_,TL,TLwords), !,
   pick_speclist(TLwords,TL,F, TargetW).

/* ** translate subject NP (infl spec) ** */
transl_spec(SL,inf1,[[n,Fl] NP1], TL, [[n,F2] NP2]) :-
   translate(SL, [[n,Fl] NP1], TL, [[n,F2] NP2]).

/* ** Translate complements ** */
transl_comp(_,e,_,e).  /* empty */
transl_comp(_,[],_,[]).  /* no complements */
transl_comp(_,[[mode,M]],_,[[mode,M]]) :- !.

/* ** Auxiliary functions ** */
capitalize_proper(L,W,Wout) :-
   diet(L,W,n,Lexinfo),
   member(proper(+),Lexinfo),
   capitalize(W,Wout).
capitalize_proper(_,W,W).

adjust(_,_,W,[]),W.      /* No inflection */
adjust(pl,_,W,[pl(-)],W).  /* Not plural */
adjust(fem,_,W,[fem(-)],W).  /* Not feminine */
adjust(pl,C,TL,W,[pl(+)],Target) :- /* pluralize */
   dict(TL,W,C,Lexinfo),
   test_f(pl(+),TL,W,C,Lexinfo,Target).
adjust(fem,C,TL,W,[fem(+)],Target) :- /* feminize */
   dict(TL,W,C,Lexinfo),
   test_f(fem(+),TL,W,C,Lexinfo,Target).

test_f(Fitem,L,Root,C,_,Root) :- /* already inflected */
   extractLocal(L,Root,C,_,f,F),
   member(Fitem,F), !.
test_f(Fitem,_,Root,_,Lexinfo,W) :- /* otherwise: inflect */
   Fitem =.. [Ftype,_,]
get(Ftype, Lexinfo, Fpred),
apply(Ftype, Root, Fpred, W).

apply(pl, Root, [], Target) :-
split(Root, s, Target).
apply(fem, Root, [], Target) :-
  split(Stem, os, Root),
  split(Stem, as, Target).
apply(fem, Root, [], Target) :-
  split(Stem, o, Root),
  split(Stem, a, Target).
apply(F, _, [Fpred], Target) :-
  Fpred =.. [F, Target].

/* default pl: add 's' */
/* default fem: */
/* delete "-os" */
/* add "-as" */
/* delete "-o" */
/* add "-a" */
/* add "-a" */
/* Else use inflection */
/* from Lexinfo */
Appendix K. Prolog Code - TRANSFORM

*********************************************************/
/*
TRANSFORMATIONS
*/
/*********************************************************/
transform_and_generate(_,[]).
transform_and_generate(_,[Dstructure|T]) :-
    bagof(Sstructure,
        transformation(L,Dstructure, Sstructure),
        Sstructures),
    generate_and_print(L,Sstructures),
    transform_and_generate(_,[T]).

transformation(L,Dstructure,Sstructure) :-
    transform(L,matrix(+),_,Dstructure,Sstructure),
    dbl_comp_filter(Sstructure),
    case_filter(L,Sstructure),
    binding_conditions(L,Sstructure),
    ecp(L,Sstructure),
    write('...Transformed'),nl.

transform(L,Matrix,Mode,COMP1, COMP11) :-
    transform_lower(L,matrix(-),Mode,COMP1, COMP2),
    set_tns(Matrix,COMP2, COMP3),
    move_alpha(L,COMP3, COMP4),
    it_insertion(L,COMP4, COMP5),
    null_subject(L,COMP5, COMP6),
    do_support(L,Matrix,Mode,COMP6, COMP7),
    move_affix(L,COMP7,COMP8),
    comp_insertion(L,Matrix,Mode,COMP8,COMP9),
    modal_insertion(L,COMP9,COMP10),
    inversion(L,Matrix,Mode,COMP10,COMP11).

/*********************************************************/
/*  Transform Lower Cycle  */
/*********************************************************/
transform_lower(_,_,Mode,[[comp,F]|COMP], [[comp,F]|COMP]) :-
    member([mode,Mode],COMP), !.

transform_lower(L,Matrix,Mode,[[C,F],Spec,[Pre,HD|Argsl] Post],
    [[C,F],Spec,[Pre,HD|Argsl2] Post]) :-
    transf_lower_args(L,Matrix,Mode,Argsl, Argsl2).

transf_lower_args(L,Matrix,Mode,[[[comp,F1]|COMP1]|T],
    [[[comp,F2]|COMP2]|T]) :-
    !,transform(L,Matrix,Mode,[[comp,F1]|COMP1],
    [[[comp,F2]|COMP2]|T]).

transform_lower_args(L,Matrix,Mode,[H1|T], [H2|T]) :-
    transform_lower(L,Matrix,Mode,H1, H2).

/*********************************************************/
/*  Set TNS  */
/*********************************************************/
set_tns(_, COMP, COMP). /* leave clause as is first time */
set_tns(matrix(-),[[comp,F],WH,[e,COMP,
  [[infl,F1],NP,INFL]|Post],
  [[comp,F],WH,[e,COMP,
    [[infl,F2],NP,INFL]|Post]]) :-
  alternate_tns(NP,F1, F2), !.
alternate_tns(NP,F1, [tns(pres)|F2]) :-
  member(tns(-),F1),
  subtract(F1,[tns(-)], F2),
  features(NP,FNP),
  union(FNP,F2, F3),
  default(per,3,F3,F4),
  default(pl,-,F4,F5).
alternate_tns(_,F1,[tns(-)|F2]) :-
  subtract(Fl,[tns(-)], F2).
/* MOVE ALPHA */
move_alpha(_,COMP,COMP). /* no movement */
  /* NP movement to subject */
move_alpha(L,[[comp,FC],WH,[e,COMP,
  [[infl,BaseFI],BaseNP,BaseINFL]|Post],
  [[comp,FC],WH,[e,COMP,
    [[infl,FI],NP,INFL]|Post]]) :-
  not lexical(BaseNP),
  member(theta(-),BaseFI),
  move(L,n,0,NP,[ana(+)],BaseINFL,INFL),
  features(NP,F),
  combineF(F,BaseFI,FI).
  /* NP movement to SUBJ, then COMP */
move_alpha(L,[[comp,FC],e,[e,COMP,
  [[infl,BaseFI],BaseNP,BaseINFL]|Post],
  [[comp,[wh(+)|FC]],WH,[e,COMP,
    [[infl,FI],NP,INFL]|Post]]) :-
  not lexical(BaseNP),
  member(theta(-),BaseFI),
  ((bounding-cat(L,infl),Level=1) ; Level=0),
  move(L,n,Level,WH,[ana(+)],BaseINFL,INFL),
  features(WH,F),
  member(wh(+),F),
  combineF(F,BaseFI,FI),
  index(WH,I),
  makelist([[n,[indx(I),pro(-),ana(-)]],e],NP).
  /* NP movement to COMP */
move_alpha(L,[[comp,FC],e,[e,e,BaseINFL]|Post],
  [[comp,[wh(+)|FC]],WH,[e,e,INFL]|Post]) :-
  move(L,n,0,WH,[pro(-),ana(-)],BaseINFL,INFL),
  features(WH,F),
  member(wh(+),F).
  /* PP movement to COMP */
move_alpha(L,[[comp,FC],e,[e,e,BaseINFL]|Post],
  [[comp,[wh(+)|FC]],WH,[e,e,INFL]|Post]) :-
move(L,p,0,WH,[pro(-),ana(-)],BaseINFL,INFL),
features(WH,F),
member(wh(+),F).

move(_,_,2,_,_,_) :- !, fail.
move(L,Alpha,_,[[Alpha,indx(I)|F]],Spec,HD, ANAPRO,
[[Alpha,F],Spec,HD],
[[Alpha,NewF],e]) :-
not member(indx(_,F),
append([indx(I)|F],ANAPRO,NewF).
move(L,Alpha,_,[[Alpha,F],Spec,HD], ANAPRO,
[[Alpha,F],Spec,HD],
[[Alpha,NewF],e]) :-
member(indx(I),F),
append(F,ANAPRO,NewF).
move(L,Alpha,Level,XP,ANAPRO,[[C,F]|T1], [[C,F]|T2]) :-
bounding_cat(L,C),
Level2 is Level1+1,
!, move(L,Alpha,Level2,XP,ANAPRO,T1,T2).

move(L,Alpha,Level,XP,ANAPRO,[[C,F]|T1], [[C,F]|T2]) :-
!, move(L,Alpha,Level,XP,ANAPRO,T1,T2).
move(L,Alpha,Level,XP,ANAPRO,[[C,F]|T1], [[C,F]|T2]) :-
!, move(L,Alpha,Level,XP,ANAPRO,T1,T2).

move_affix(L,[[comp,FC],WH,[e,COMP,
[[infl,FI],NP,[e,BaseINFL,BaseVP]]]|Post],
[[comp,FC],WH,[e,COMP,
[[infl,FI],NP,[e,INFL,VP]]]|Post]) :-
affix_infl(L,FI,BaseINFL, FV,INFL),
affix_vp(L,FV,BaseVP, VP), !.

affix_infl(_,F,e,F,e).
affix_infl(L,F,SourceW, [tns(-)|F],TargetW) :-
dict(L,SourceW,inf1,Lexinfo),
not member(root(_),Lexinfo),
gen_verb(L,inf1,F,SourceW,Lexinfo, TargetW).

affix_vp(L,FI,[[v,FV],__[__,[BaseV|Args]|PostV],
[[v,FV],PERF,[PROG,V|Args]|PostV]) :-
affect_to_perf(L,FI, PERF),
affect_to_prog(L,FI, PROG),
affect_to_verb(L,FI, BaseV, V).

affix_to_perf(L,FI,W) :- member(perf(+),FI),
create(perf,L,FI,W).
affix_to_perf(L,FI,W) :-
  ( member(prog(+),FI) ; member(pass(+),FI) ),
  create(prog,L,FI,W).
affix_to_perf(_,_,e).

affix_toprog(L,FI,W) :-
  member(perf(+),FI),
  ( member(prog(+),FI) ; member(pass(+),FI) ),
  extractLocal(L,W,prog,_[f[perf(+)]].
affix_toprog(L,FI,W) :-
  member(prog(+),FI), member(pass(+),FI),
  extractLocal(L,W,prog,_[f[prog(+)]].
affix_toprog(_,_,e).

create(C,L,F,W) :- /* If tns(-), select infinitive form */
  member(tns(-),F),
  rootword(L,_C,W).
create(C,L,F,W) :- /* otherwise: select inflected form: */
  not member(tns(-),F),
  get(per,F,PER),
  get(pl,F,NUM),
  get(tns,F,TENSE),
  append(PER,NUM,PERNUM),
  create2(L,C,TENSE,PERNUM,W).

create2(L,C,TENSE,PERNUM,W) :-
  dict(L,W,C,Lexinfo),
  member(f(F),Lexinfo),
  makelist(F,WF),
  must_appear(TENSE,WF),
  must_agree(PERNUM,WF).
create2(english,C,[tns(pres)],_,RW) :- /* */
  rootword(english,_C,RW).

affix_to_verb(L,FI,expr([BaseW|Expr]),expr([W|Expr])) :-
  affix_to_verb(L,FI,BaseW,W).

affix_to_verb(L,FI,BaseW,W) :-
  dict(L,BaseW,v,Lexinfo),
  not member(root(_),Lexinfo),
  affix_verb2(L,FI,Lexinfo,BaseW,W).

affix_verb2(L,FI,Lexinfo,BaseW,W) :-
  member(pass(+),FI), gen_perf(L,BaseW,Lexinfo,W).
affix_verb2(L,FI,Lexinfo,BaseW,W) :-
  member(prog(+),FI), gen_prog(L,BaseW,Lexinfo,W).
affix_verb2(L,FI,Lexinfo,BaseW,W) :-
  member(perf(+),FI), gen_perf(L,BaseW,Lexinfo,W).
affix_verb2(_,FI,_,W,W) :- member(tns(-),FI).
affix_verb2(english,FI,_,W,W) :- member(tns(fut),FI).
affix_verb2(L,FI,Lexinfo,BaseW, W) :-
gen_verb(L,v,FI,BaseW, Lexinfo, W).

/*******************

gen_verb(L,C,FI,BaseW, Lexinfo, W) :- /* irregular forms */
    member(irr(IRR),Lexinfo),
    makelist(IRR,IRRlist),
    get_ird(L,C,FI,BaseW, IRRlist,W).

gen_verb(L,_,FI,BaseW, Lexinfo, W) :- /* stem forms */
    member(StemCat,Lexinfo),
    StemCat =.. [StemNum,St],
    name(StemNum,[S,T,E,M|_]),
    name(stem,[S,T,E,M]),
    makelist(St,Stem),
    get_tern(StemNum,L,FI,BaseW,Stem,W).

gen_verb(english,C,FI,BaseW,_, BaseW) :- /* default form */
    member(tns(pres), FI),
    ( member(pl(+),FI) ;
      member(per(2),FI) ;
      member(per(1),FI) ).
gen_verb(english,inf1,FI,BaseW,_,BaseW).
gen_verb(L,C,FI,BaseW,_, W) :- /* generate form */
suffixtable(L,Sfx,Stemend,C,SfxF),
    makelist(SfxF,SfxList),
must_appear(SfxList,FI),
split(Stem,Stemend,BaseW),
split(Stem,Sfx,W),
morph_word(L,W).

/*-----------------------------*/

get_ird(L,C,FI,BaseW,[Wl|Ws], Wl) :-
dict(L,Wl,C,Lexinfo),
    member(root(BaseW),Lexinfo),
    member(f(F),Lexinfo),
    makelist(F,Flist),
    member(tns(_),Flist),
    must_agree(Flist,FI).

get_ird(L,C,FI,BaseW,[_|Ws],W) :- get_ird(L,C,FI,BaseW,Ws,W).

/*-----------------------------*/

get_tern(STEMtype,L,FI,BaseW,[Stem|_],W) :-
    get_proper_tern(STEMtype,L,FI,BaseW,Stem),
    create_tern(STEMtype,L,FI,Stem,W).

get_tern(STEMtype,L,FI,BaseW,[_|Stems],W) :-
    get_tern(STEMtype,L,FI,BaseW,Stems,W).

get_proper_tern(C,L,FI,BaseW,Stem) :-
dict(L,Stem,C,Lexinfo),
    member(root(BaseW),Lexinfo),
    member(f(F),Lexinfo),
    makelist(F,Flist),
    must_appear(Flist,FI).

create_tern(STEMtype,L,FI,Stem,W) :-
suffixtable(L,Sfx,_,STEMtype,SfxF),
    makelist(SfxF,SfxList),
    must_appear(SfxList,FI),
    split(Stem,Sfx,W).

/*****************************/
/* Doubly-Filled COMP Filter */

/* Case Filter */

case_filter(L,COMP) :- subj_pos2(L,VP), !, obj_pos2(COMP), !.

/* Variables:Case */
is_variable(NP), subject_pos(L,VP).

/* bypass inverted INFL */
subject_pos(L,[inf1,F1],[_,_,VP]) :- !, subject_pos2(L,[inf1,F1],[INFL]).

/* Case from +TNS */
subject_pos(L,[inf1,F1],[_,_,VP]) :- !, subject_pos(L,[inf1,F1],[INFL]).
subj_pos(L,[[_,_,e]]).

subj_pos2(,,[]).
subj_pos2(L,[H|T]) :- subj_pos(L,H), subj_pos2(L,T).
/*-----------------------------------------------*/
obj_pos([[v,FV],_,[_,_,[n,_,_,_]|Args]|_]) :-
    not member(pass(+),FV), !,
    obj_pos2(Args).
obj_pos([[inf1,FI],_,[[inf1,FI],NP,[_,_,VP]]|_]) :-
    obj_pos(VP).
obj_pos([[_,_,[_,_,Args]|_]]):-
    obj_pos2(Args).
obj_pos([[_,_,e]]).

obj_pos2([]).
obj_pos2([H|T]) :- obj_pos(H), obj_pos2(T).
/*-----------------------------------------------*/
/* BINDING CONDITIONS */
/*-----------------------------------------------*/
binding_conditions(L,COMP) :- condition_A(L,COMP), !,
    condition_C(COMP), !.

condition_A(L,[[X,_,],[_,XW, /* S-deletion */
    [[comp,_,WH,[e,COMP, /* (with inversion) */
    [inf1,_,,
    [[inf1,FI],NP,[_,_,VP]]]|_|]*]
    )]) :-
    is_nptrace(NP), !,
    WH = e,
    COMP = e,
    member(tns(-),FI), !,
    is_sdeleter(L,XW,X),
    condition_A2(L,VP).
condition_A(L,[[X,_,],[_,XW, /* S-deletion */
    [[comp,_,WH,[e,COMP, /* (no inversion) */
    [inf1,FI],NP,[_,_,VP]]]|_|]*]
    )]) :-
    is_nptrace(NP), !,
    WH = e,
    COMP = e,
    member(tns(-),FI),
    is_sdeleter(L,XW,X),
    condition_A2(L,VP).
condition_A(L,[[inf1,_,NP1,[_,_, */ obj bound to subj */
    [[v,_,],[_,_,Args]|_]]) :-
    member(NP2,Args),
    is_nptrace(NP2), !,
    index(NP1,I),
    index(NP2,J),
    I == J, !,
    condition_A2(L,Args).
condition_A(_,NP) :- /* otherwise NP-trace invalid */
    is_nptrace(NP), !,fail.
condition_A(L,[[infl,_,_],[[infl,FI][INFL]]]) :-
    !,condition_A(L,[[infl,FI][INFL]]).

condition_A(L,[[_,_],NP,[_,_|Args]|Post]) :-
    !,condition_A(L,NP),
    condition_A2(L,Args),
    condition_A2(L,Post).

condition_A(_,_).

condition_A2(L,[]).
condition_A2(L,[A|As])  :-  condition_A(L,A),
    condition_A2(L,As).

condition_C([[comp,_,_],_,[_,_,
    [[infl,_,_],
    [[infl,_,NP,[_,_|VP]]]|_]]|_))  :-
    !,var_free(NP,VP),
    condition_C2(VP).
condition_C([[comp,_,_],_,[_,_,
    [[infl,_,NP,[_,_|VP]]]|_]) :-
    var_free(NP,VP),
    condition_C2(VP).

var_free(NP,Args)  :-  index(NP,I),  !, free(I,Args).
var_free(_,_).

free(I,[H|T]) :- free_arg(I,H), free_arg2(I,H), free(I,T).
free(_,[]).
free_arg(I,X)  :-  not  (is_variable(X), index(X,J),  I == J).
free_arg2(I,[[infl,_,_],[[infl,FI][INFL]]]) :-
    !, free_arg2(I,[[infl,FI][INFL]]).
free_arg2(I,[[infl,_,NP,[_,_|Args]]|_])  :-
    !, free_arg(I,NP),
    free(I,Args).
free_arg2(I,[[_,_,_,[_,_|Args]]|_])  :-
    !, free(I,Args).

condition_C2([H|T]) :- condition_C3(H), condition_C2(T).
condition_C2([]).

condition_C3([[comp,_,|COMP]]) :-
    !, condition_C([[comp,_,|COMP]]).
condition_C3(_).

="%*****************************************************************************************************/
/*
/*                   ECP - Empty Category Principle
/*
/*
="/*****************************************************************************************************/
ecp(_,[mode, _]) :- !.
ecp(L,[[X,_,_],[_,XW, /* S-deletion */
[[comp,],e,[e,e, /* (with inversion) */
[[infl,],_,
[[infl,],NP,[_,_,VP]]],[_]|PostX]) :-
not lexical(NP),
is_sdeleter(L,XW,X), !,
not is_PRO(NP),
ecp(L,VP),
ecp2(L,PostX), !.
ecp(L,[X,],_[_,XW,
[[comp,],e,[e,e, /* (no inversion) */
[[infl,],NP,[_,_,VP]]],[_]|PostX]) :-
not lexical(NP),
is_sdeleter(L,XW,X), !,
not is_PRO(NP),
ecp(L,VP),
ecp2(L,PostX), !.
ecp(L,[[comp,],WH,[e,e, /* WH in COMP */
[[infl,],_,
[[infl,],NP,[_,_,VP]]],[_]|Post]) :- /* (with inversion) */
index(WH,_), !,
not is_PRO(NP),
ecp(L,VP), !.
ecp(L,[[infl,FI],NP,[_,_,VP]]) :- /* +TNS */
not lexical(NP),
not member(tns(-),FI), !,
proper_governor(L,infl),
ecp(L,VP), !.
ecp(L,[X,],_[_,[Args]|PostX]) :- /* head */
member(XP,Args),
is_trace(XP), !,
proper_governor(L,X),
ecp2(L,Args),
ecp2(L,PostX), !.
ecp(L,[X,],_[_,[PostX]) :- /* adjunct */
member(XP,PostX),
is_trace(XP), !,
proper_governor(L,X).
/* ECP: embedded clauses */
ecp(L,[[infl,],_,
[[infl,],NP,[_,_,VP]]]) :-
ecp(L,VP), !.
ecp(L,[_,[_,[Args]|PostX]) :-
ecp2(L,Args),
ecp2(L,PostX), !.
ecp(_,[_,[e]]).
ecp2(L,[H|T]) :- ecp(L,H), ecp2(L,T).
Appendix L. Prolog Code – GENERATE

/*************************************************************************************/
/* */
/* GENERATE */
/* */
/*************************************************************************************/

generate_and_print(_,[]).
generate_and_print(L,[Sstructure|T]) :-
generate(L,Sstructure, Surface),
printout(Surface),
generate_and_print(L,T).

generate(L,TStructure,Tsurface) :-
readleaves(TStructure,Wordlist1),
delete_e(Wordlist1,Wordlist2),
collocate(L,Wordlist3,Wordlist2),
contract(L,Tsurface,Wordlist3).

readleaves([],[]).
readleaves([mode,X], [X]).
readleaves([[-,F]|X], S) :- makelist(_,F), readleaves(X,S).
readleaves([H|T], S) :- readleaves(H,S1),
readleaves(T,S2),
append(S1,S2,S).

readleaves(expr(Expr), Expr).
readleaves(W, [W]).

delete_e([],[]).
delete_e([e|Tl],T2) :- delete_e(Tl,T2).
delete_e([H|Tl],[H|T2]) :- delete_e(Tl,T2).
/*************************************************************************************/
/* */
/*************************************************************************************/
printout([H|T]) :-
capitalize(H,HC),
display(HC),
display_sent(T),
nl.
printout(X) :- display('Invalid string to printout: '), nl,
display(X), abort.
capitalize(H,HC) :-
name(H,[HH|HT]),
HH>96, HH<123, HHC is HH-32,
name(HC,[HHC|HT]).
capitalize(H,H).
display_sent([decl]) :- display('.').
display_sent([ques]) :- display('?').
display_sent([H|T]) :- display(' '),
display(H),
display_sent(T).