## Chapter 4: Features and Augmented Grammars

4.1 Feature Systems and Augmented Grammars

4.2 Some Basic Feature Systems for English

4.3 Morphological Analysis and the Lexicon

4.4 A Simple Grammar Using Features

4.5 Parsing with Features

4.6 Augmented Transition Networks

4.7 Definite Clause Grammars

4.8 Generalized Feature Systems and Unification Grammars

**Summary**

**Related Work and Further Readings**

**Exercises for Chapter 4**
Context-free grammars provide the basis for most of the computational parsing mechanisms developed to date, but as they have been described so far, they would be very inconvenient for capturing natural languages. This chapter describes an extension to the basic context-free mechanism that defines constituents by a set of features. This extension allows aspects of natural language such as agreement and subcategorization to be handled in an intuitive and concise way.

Section 4.1 introduces the notion of feature systems and the generalization of context-free grammars to allow features. Section 4.2 then describes some useful feature systems for English that are typical of those in use in various grammars. Section 4.3 explores some issues in defining the lexicon and shows how using features makes the task considerably simpler. Section 4.4 describes a sample context-free grammar using features and introduces some conventions that simplify the process. Section 4.5 describes how to extend a chart parser to handle a grammar with features. The remaining sections, which are optional, describe how features are used in other grammatical formalisms and explore some more advanced material. Section 4.6 introduces augmented transition networks, which are a generalization of recursive transition networks with features, and Section 4.7 describes definite clause grammars based on PROLOG. Section 4.8 describes generalized feature systems and unification grammars.

4.1 Feature Systems and Augmented Grammars

In natural languages there are often agreement restrictions between words and phrases. For example, the NP "a men" is not correct English because the article a indicates a single object while the noun "men" indicates a plural object; the noun phrase does not satisfy the number agreement restriction of English. There are many other forms of agreement, including subject-verb agreement, gender agreement for pronouns, restrictions between the head of a phrase and the form of its complement, and so on. To handle such phenomena conveniently, the grammatical formalism is extended to allow constituents to have features. For example, we might define a feature NUMBER that may take a value of either s (for singular) or p (for plural), and we then might write an augmented CFG rule such as

\[ NP \rightarrow \text{ART N only when NUMBER}_1 \text{ agrees with NUMBER}_2 \]

This rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the first word agrees with the number feature of the second. This one rule is equivalent to two CFG rules that would use different terminal symbols for encoding singular and plural forms of all noun phrases, such as
NP-SING -> ART-SING N-SING

NP-PLURAL -> ART-PLURAL N-PLURAL

While the two approaches seem similar in ease-of-use in this one example, consider that all rules in the grammar that use an NP on the right-hand side would

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 84]

now need to be duplicated to include a rule for NP-SING and a rule for NP-PLURAL, effectively doubling the size of the grammar. And handling additional features, such as person agreement, would double the size of the grammar again and again. Using features, the size of the augmented grammar remains the same as the original one yet accounts for agreement constraints.

To accomplish this, a constituent is defined as a feature structure - a mapping from features to values that defines the relevant properties of the constituent. In the examples in this book, feature names in formulas will be written in boldface. For example, a feature structure for a constituent ART1 that represents a particular use of the word a might be written as follows:

ART1:

(CAT ART

ROOT a

NUMBER s)

This says it is a constituent in the category ART that has as its root the word a and is singular. Usually an abbreviation is used that gives the CAT value more prominence and provides an intuitive tie back to simple context-free grammars. In this abbreviated form, constituent ART1 would be written as

ART1: (ART ROOT a NUMBER s)

Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves
can occur as values. Special features based on the integers - 1, 2, 3, and so on - will stand for the first subconstituent, second subconstituent, and so on, as needed. With this, the representation of the NP constituent for the phrase "a fish" could be

\[ NP1: (NP \text{ NUMBERs} ) \]

\[ 1 (\text{ART ROOT a}) \]

\[ \text{NUMBER s}) \]

\[ 2 (\text{N ROOT fish}) \]

\[ \text{NUMBER s}) \]

Note that this can also be viewed as a representation of a parse tree shown in Figure 4.1, where the subconstituent features 1 and 2 correspond to the subconstituent links in the tree.

The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations. For example, a rule for simple noun phrases would be as follows:

\[ (NP \text{ NUMBER ?n}) - (ART \text{ NUMBER ?n}) (N \text{ NUMBER ?n}) \]

This says that an NP constituent can consist of two subconstituents, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical. According to this rule, constituent NP1 given previously is a legal constituent. On the other hand, the constituent

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 85]
Variables are also useful in specifying ambiguity in a constituent. For instance, the word fish is ambiguous between a singular and a plural reading. Thus the word might have two entries in the lexicon that differ only by the value of the NUMBER feature. Alternatively, we could define a single entry that uses a variable as the value of the NUMBER feature, that is,

\[(N \text{ ROOT fish NUMBER } ?n)\]

This works because any value of the NUMBER feature is allowed for the word fish. In many cases, however, not just any value would work, but a range of values is possible. To handle these cases, we introduce
constrained variables, which are variables that can only take a value out of a specified list. For example, the variable \(?n\{s\ p\}\) would be a variable that can take the value s or the value p. Typically, when we write such variables, we will drop the variable name altogether and just list the possible values. Given this, the word fish might be represented by the constituent

\[(N \text{ ROOT fish NUMBER } ?n\{sp\})\]

or more simply as

\[(N \text{ ROOT fish NUMBER } \{s\ p\})\]

Box 4.1

Box 4.1 Formalizing Feature Structures

There is an active area of research in the formal properties of feature structures. This work views a feature system as a formal logic. A feature structure is defined as a partial function from features to feature values. For example, the feature structure

\[\text{ARTI}: \ (\text{CAT ART} \ \
\text{ROOT a} \ \
\text{NUMBER s})\]

is treated as an abbreviation of the following statement in FOPC:

\[\text{ARTI(CAT)} = \text{ART} \land \text{ARTI(ROOT)} = \text{a} \land \text{ARTI(NUMBER)} = \text{s}\]

Feature structures with disjunctive values map to disjunctions. The structure

\[\text{THEI}: \ (\text{CAT ART} \ \
\text{ROOT the} \ \
\text{NUMBER } \{s\ p\})\]

would be represented as

\[\text{THEI(CAT)} = \text{ART} \land \text{THEI(ROOT)} = \text{the} \ \
\land (\text{THEI(NUMBER)} = \text{s} \lor \text{THEI(NUMBER)} = \text{p})\]

Given this, agreement between feature values can be defined as equality equations.
There is an interesting issue of whether an augmented context-free grammar can describe languages that cannot be described by a simple context-free grammar. The answer depends on the constraints on what can be a feature value. If the set of feature values is finite, then it would always be possible to create new constituent categories for every combination of features. Thus it is expressively equivalent to a context-free grammar. If the set of feature values is unconstrained, however, then such grammars have arbitrary computational power. In practice, even when the set of values is not explicitly restricted, this power is not used, and the standard parsing algorithms can be used on grammars that include features.

4.2 Some Basic Feature Systems for English

This section describes some basic feature systems that are commonly used in grammars of English and develops the particular set of features used throughout this book. Specifically, it considers number and person agreement, verb form features, and features required to handle subcategorization constraints. You should read this to become familiar with the features in general and then refer back to it later when you need a detailed specification of a particular feature.

Person and Number Features

In the previous section, you saw the number system in English: Words may be classified as to whether they can describe a single object or multiple objects. While number agreement restrictions occur in several different places in English, they are most importantly found in subject-verb agreement. But subjects and verbs must also agree on another dimension, namely with respect to the person. The possible values of this dimension are

First Person (1): The noun phrase refers to the speaker, or a group of people including the speaker (for example, I, we, you, and 0.

Second Person (2): The noun phrase refers to the listener, or a group including the listener but not including the speaker (for example, you, all of you).
Third Person (3): The noun phrase refers to one or more objects, not including the speaker or hearer.

Since number and person features always co-occur, it is convenient to combine the two into a single feature, AGR, that has six possible values: first person singular (is), second person singular (2s), third person singular (3s), and first, second and third person plural (ip, 2p, and 3p, respectively). For example, an instance of the word is can agree only with a third person singular subject, so its AGR feature would be 3s. An instance of the word are, however, may agree with second person singular or any of the plural forms, so its AGR feature would be a variable ranging over the values {2s 1p 2p 3p}.

**Verb-Form Features and Verb Subcategorization**

Another very important feature system in English involves the form of the verb. This feature is used in many situations, such as the analysis of auxiliaries and generally in the subcategorization restrictions of many head words. As described in Chapter 2, there are five basic forms of verbs. The feature system for verb forms will be slightly more complicated in order to conveniently capture certain phenomena. In particular, we will use the following feature values for the feature VFORM:

- base - base form (for example, go, be, say, decide)
- pres - simple present tense (for example, go, goes, am, is, say, says, decide)
- past - simple past tense (for example, went, was, said, decided)
- fin - finite (that is, a tensed form, equivalent to {pres past})
- ing - present participle (for example, going, being, saying, deciding)
- pastprt - past participle (for example, gone, been, said, decided)
- inf - a special feature value that is used for infinitive forms with the word to

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 88]
To handle the interactions between words and their complements, an additional feature, SUBCAT, is used. Chapter 2 described some common verb subcategorization possibilities. Each one will correspond to a different value of the SUBCAT feature. Figure 4.2 shows some SUBCAT values for complements consisting of combinations of NPs and VPs. To help you remember the meaning of the feature values, they are formed by listing the main category of each part of the complement. If the category is restricted by a feature value, then the feature value follows the constituent separated by a colon. Thus the value npvp:inf will be used to indicate a complement that consists of an NP followed by a VP with VFORM value inf. Of course, this naming is just a convention to help the reader; you could give these values any arbitrary name, since their significance is determined solely by the grammar rules that involve the feature. For instance, the rule for verbs with a SUBCAT value of _np_vp:inf would be

\[
(VP) \rightarrow (V \text{ SUBCAT} \_np\_vp:inf) \\
(NP) \\
(VP \text{ VFORM} \text{ inf})
\]

This says that a VP can consist of a V with SUBCAT value _np_vp:inf, followed by an NP, followed by a VP with VFORM value inf. Clearly, this rule could be rewritten using any other unique symbol instead of _np_vp:inf, as long as the lexicon is changed to use this new value.

Many verbs have complement structures that require a prepositional phrase with a particular preposition, or one that plays a particular role. For example, the verb give allows a complement consisting of an NP followed by a PP using the preposition to, as in "Jack gave the money to the bank". Other verbs, such as "put", require a prepositional phrase that describes a location, using prepositions such as "in", "inside", "on", and "by". To
express this within the feature system, we introduce a feature PFORM on prepositional phrases. A prepositional phrase with a PFORM value such as TO must have the preposition to as its head, and so on. A prepositional phrase with a PFORM value LOC must describe a location. Another useful PFORM value is MOT, used with verbs such as walk, which may take a

<table>
<thead>
<tr>
<th>Value</th>
<th>Example Prepositions</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO</td>
<td>to</td>
<td>I gave it to the bank.</td>
</tr>
<tr>
<td>LOC</td>
<td>in, on, by, inside, on top of</td>
<td>I put it on the desk.</td>
</tr>
<tr>
<td>MOT</td>
<td>to, from, along, ...</td>
<td>I walked to the store.</td>
</tr>
</tbody>
</table>

**Figure 4.3** Some values of the PFORM feature for prepositional phrases

---

Figure 4.3 Some values of the PFORM feature for prepositional phrases

<table>
<thead>
<tr>
<th>Value</th>
<th>Example Verb</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>_np_pp:to</td>
<td>give</td>
<td>Jack gave the key to the man.</td>
</tr>
<tr>
<td>_pp:loc</td>
<td>be</td>
<td>Jack is at the store.</td>
</tr>
<tr>
<td>_np_pp:loc</td>
<td>put</td>
<td>Jack put the box in the corner.</td>
</tr>
<tr>
<td>_pp:mot</td>
<td>go</td>
<td>Jack went to the store.</td>
</tr>
<tr>
<td>_np_pp:mot</td>
<td>take</td>
<td>Jack took the hat to the party.</td>
</tr>
<tr>
<td>_adjp</td>
<td>be, seem</td>
<td>Jack is happy.</td>
</tr>
<tr>
<td>_np_adjp</td>
<td>keep</td>
<td>Jack kept the dinner hot.</td>
</tr>
<tr>
<td>_s:that</td>
<td>believe</td>
<td>Jack believed that the world was flat.</td>
</tr>
<tr>
<td>_s:for</td>
<td>hope</td>
<td>Jack hoped for the man to win the prize.</td>
</tr>
</tbody>
</table>

**Figure 4.4** Additional SUBCAT values
prepositional phrase that describes some aspect of a path, as in We walked to the store. Prepositions that can create such phrases include to, from, and along. The LOC and MOT values might seem hard to distinguish, as certain prepositions might describe either a location or a path, but they are distinct. For example, while Jack put the box (in on by] the corner is fine, *Jack put the box (to from along] the corner is ill-formed. Figure 4.3 summarizes the PFORM feature.

This feature can be used to restrict the complement forms for various verbs. Using the naming convention discussed previously, the SUBCAT value of a verb such as put would be jip pp:loc, and the appropriate rule in the grammar would be

\[
\begin{align*}
(VP) & \rightarrow (V \text{ SUBCAT } _{np \_pp:loc}) \\
(NP) & \\
(PP & \text{ PFORM LOC})
\end{align*}
\]

For embedded sentences, a complementizer is often needed and must be subcategorized for. Thus a COMP feature with possible values for, that, and no-comp will be useful. For example, the verb tell can subcategorize for an S that has the complementizer that. Thus one SUBCAT value of tell will be _s:that. Similarly, the verb wish subcategorizes for an S with the complementizer for, as in We wished for the rain to stop. Thus one value of the SUBCAT feature for wish is _s:for. Figure 4.4 lists some of these additional SUBCAT values and examples for a variety of verbs. In this section, all the examples with the SUBCAT feature have involved verbs, but nouns, prepositions, and adjectives may also use the SUBCAT feature and subcategorize for their complements in the same way.

**Binary Features**

Certain features are binary in that a constituent either has or doesn’t have the feature. In our formalization a binary feature is simply a feature whose value is restricted to be either + or -. For example, the INV feature is a
binary feature that indicates whether or not an S structure has an inverted subject (as in a yes/no question). The S structure for the sentence Jack laughed will have an INV value —, whereas the S structure for the sentence Did Jack laugh? will have the INV value +. Often, the value is used as a prefix, and we would say that a structure has the feature +INV or -INV. Other binary features will be introduced as necessary throughout the development of the grammars.

The Default Value for Features

It will be useful on many occasions to allow a default value for features. Any-time a constituent is constructed that could have a feature, but a value is not specified, the feature takes the default value of -. This is especially useful for binary features but is used for nonbinary features as well; this usually ensures that any later agreement check on the feature will fail. The default value is inserted when the constituent is first constructed.

4.3 Morphological Analysis and the Lexicon

Before you can specify a grammar, you must define the lexicon. This section explores some issues in lexicon design and the need for a morphological analysis component

The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions. When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

Because words tend to follow regular morphological patterns, however, many forms of words need not be explicitly included in the lexicon. Most English verbs, for example, use the same set of suffixes to indicate different forms: -s is added for third person singular present tense, -ed for past tense, -ing for the present participle, and so on. Without any morphological analysis, the lexicon would have to contain every one of these forms. For the verb want this would require six entries, for want (both in base and present form), wants, wanting, and wanted (both in past and past participle form).

In contrast, by using the methods described in Section 3.7 to strip suffixes there needs to be only one entry for want. The idea is to store the base form of the
verb in the lexicon and use context-free rules to combine verbs with suffixes to derive the other entries. Consider the following rule for present tense verbs:

\[(V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{pres} \ \text{AGR} \ 3s) \rightarrow (V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{base}) \ (+S)\]

where +S is a new lexical category that contains only the suffix morpheme -s. This rule, coupled with the lexicon entry

\[\text{want:} \quad (V \text{ ROOT} \ \text{want} \ \text{SUBCAT} \ \{_{np\_vp:inf} _{np\_vp:inf}\} \ \text{VFORM} \ \text{base})\]

would produce the following constituent given the input string want -s

\[\text{want:} \quad (V \text{ ROOT} \ \text{want} \ \text{SUBCAT} \ \{_{np\_vp:inf} _{np\_vp:inf}\} \ \text{VFORM} \ \text{pres} \ \text{AGR} \ 3s)\]

Another rule would generate the constituents for the present tense form not in third person singular, which for most verbs is identical to the root form:

\[(V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{pres} \ \text{AGR} \ \{ls \ 2s \ lp \ 2p \ 3p\}) \rightarrow (V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{base})\]

But this rule needs to be modified in order to avoid generating erroneous interpretations. Currently, it can transform any base form verb into a present tense form, which is clearly wrong for some irregular verbs. For instance, the base form be cannot be used as a present form (for example, "We be at the store"). To cover these cases, a feature is introduced to identify irregular forms. Specifically, verbs with the binary feature +IRREG-PRES have irregular present tense forms. Now the rule above can be stated correctly:

\[(V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{pres} \ \text{AGR} \ \{ls \ 2s \ lp \ 2p \ 3p\}) \rightarrow (V \text{ ROOT} \ ?r \ \text{SUBCAT} \ ?s \ \text{VFORM} \ \text{base} \ \text{IRREG-PRES} \ -)\]
Because of the default mechanism, the IRREG-PRES feature need only be specified on the irregular verbs. The regular verbs default to -, as desired. Similar binary features would be needed to flag irregular past forms (IRREG-PAST, such as saw), and to distinguish -en past participles from -ed past participles (EN--PASTPRT). These features restrict the application of the standard lexical rules, and the irregular forms are added explicitly to the lexicon. Grammar 4.5 gives a set of rules for deriving different verb and noun forms using these features.

Given a large set of features, the task of writing lexical entries appears very difficult. Most frameworks allow some mechanisms that help alleviate these problems. The first technique - allowing default values for features - has already been mentioned. With this capability, if an entry takes a default value for a given feature, then it need not be explicitly stated. Another commonly used technique is...

**Present Tense**

1. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM} \text{ pres AGR} \{3s\}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base IRREG-PRES} \rightarrow +S)\)

2. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM} \text{ pres AGR} \{1s 2s 1p 2p 3p\}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base IRREG-PRES} \rightarrow)\)

**Past Tense**

3. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM} \text{ past AGR} \{1s 2s 3s 1p 2p 3p\}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base IRREG-PAST} \rightarrow +ED)\)

**Past Participle**

4. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM pastp}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base EN-PASTPRT} \rightarrow +ED)\)

5. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM pastp}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base EN-PASTPRT} +) +EN,\)

**Present Participle**

6. \((V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM ing}) \rightarrow (V \text{ ROOT} ?r \text{ SUBCAT} ?s \text{ VFORM base}) +ING\)

**Plural Nouns**

7. \((N \text{ ROOT} ?r \text{ AGR} \{3p\}) \rightarrow (N \text{ ROOT} ?r \text{ AGR} \{3s \text{ IRREG-PL}\} \rightarrow +S)\)

**Grammar 4.5** Some lexical rules for common suffixes on verbs and nouns
Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

to allow the lexicon writing to define clusters of features, and then indicate a cluster with a single symbol rather than listing them all. Later, additional techniques will be discussed that allow the inheritance of features in a feature hierarchy.

Figure 4.6 contains a small lexicon. It contains many of the words to be used in the examples that follow. It contains three entries for the word "saw" - as a noun, as a regular verb, and as the irregular past tense form of the verb "see" - as illustrated in the sentences

The saw was broken.

Jack wanted me to saw the board in half.

I saw Jack eat the pizza.

With an algorithm for stripping the suffixes and regularizing the spelling, as described in Section 3.7, the derived entries can be generated using any of the basic parsing algorithms on Grammar 4.5. With the lexicon in Figure 4.6 and Grammar 4.5, correct constituents for the following words can be derived: been, being, cries, cried, crying, dogs, saws (two interpretations), sawed, sawing, seen, seeing, seeds, wants, wanting, and wanted. For example, the word cries would be transformed into the sequence cry +s, and then rule 1 would produce the present tense entry from the base form in the lexicon.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 93]
Figure 4.6: A lexicon
Often a word will have multiple interpretations that use different entries and different lexical rules. The word saws, for instance, transformed into the sequence saw +s, can be a plural noun (via rule 7 and the first entry for saw), or the third person present form of the verb saw (via rule 1 and the second entry for saw). Note that rule I cannot apply to the third entry, as its VFORM is not base.

The success of this approach depends on being able to prohibit erroneous derivations, such as analyzing seed as the past tense of the verb "see". This analysis will never be considered if the FST that strips suffixes is correctly designed. Specifically, the word see will not allow a transition to the states that allow the -ed suffix. But even if this were produced for some reason, the IRREG-PAST value + in the entry for see would prohibit rule 3 from applying.

4.4 A Simple Grammar Using Features

This section presents a simple grammar using the feature systems and lexicon developed in the earlier sections. It will handle sentences such as the following:

The man cries.
The men cry.
The man saw the dogs.
He wants the dog.
He wants to be happy.
He wants the man to see the dog.

He is happy to be a dog.

It does not find the following acceptable:

* The men cries.

* The man cry.

* The man saw to be happy.

* He wants.

* He wants the man saw the dog.

Before developing the grammar, some additional conventions are introduced that will be very useful throughout the book. It is very cumbersome to write grammatical rules that include all the necessary features. But there are certain regularities in the use of features that can be exploited to simplify the process of writing rules. For instance, many feature values are unique to a feature (for example, the value inf can only appear in the VFORM feature, and _np_vp:inf can only appear in the SUBCAT feature). Because of this, we can omit the feature name without introducing any ambiguity. Unique feature values will be listed using square parentheses. Thus (VP SUBCAT inf) will be abbreviated as VP[inf]. Since binary features do not have unique values, a special convention is introduced for them. For a binary feature B, the constituent C[+B] indicates the constituent (C B +).

Many features are constrained so that the value on the mother must be identical to the value on its head subconstituent. These are called head features. For instance, in all VP rules the VFORM and AGR values are the same in the VP and the head verb, as in the rule

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 95]

**BOX 4.2 Systemic Grammar**

An important influence on the development of computational feature-based systems was systemic grammar (Halliday, 1985). This theory emphasizes the functional role of linguistic constructs as they affect communication. The grammar is organized as a set of choices about discourse function that determine the structure of the sentence. The choices are organized into hierarchical structures called systems. For example, the mood system would capture all the choices that affect the mood of the sentence. Part of this structure looks as follows:
This structure indicates that once certain choices are made, others become relevant. For instance, if you decide that a sentence is in the declarative mood, then the choice between bound and relative becomes relevant. The choice between yes/no and wh, on the other hand, is not relevant to a declarative sentence.

Systemic grammar was used in Winograd (1973), and Winograd (1983) contains a good discussion of the formalism. In recent years it has mainly been used in natural language generation systems because it provides a good formalism for organizing the choices that need to be made while planning a sentence (for example, see Mann and Mathiesson (1985) and Patten (1988)).

```
(VP VFORM ?v AGR ?a) ->
    (V VFORM ?v AGR ?a SUBCAT _np_vp:inf)
    (NP)
    (VP VFORM inf)
```

If the head features can be declared separately from the rules, the system can automatically add these features to the rules as needed. With VFORM and AGR declared as head features, the previous VP rule can be abbreviated as

```
VP -> (V SUBCAT _np_vp:inf) NP (VP VFORM inf)
```

The head constituent in a rule will be indicated in italics. Combining all the abbreviation conventions, the rule could be further simplified to

```
VP -> V [_vp_vp:inf] NP VP[inf]
```

A simple grammar using these conventions is shown as Grammar 4.7. Except for rules 1 and 2, which must enforce number agreement, all the rest of the feature constraints can be captured using the conventions that have been
Grammar 4.7 A simple grammar in abbreviated form

1. $S[-inv] \rightarrow (NP \ AGR \ ?a) \ (VP[\{pres\ past\}] \ AGR \ ?a)$
2. $NP \rightarrow (ART \ AGR \ ?a) \ (NAGR \ ?a)$
3. $NP \rightarrow PRO$
4. $VP \rightarrow V[-_none]$  
5. $VP \rightarrow V[-_np] \ NP$
6. $VP \rightarrow V[-_vp:inf] \ VP[inf]$  
7. $VP \rightarrow V[-_np \ vp:inf] \ NP \ VP[inf]$  
8. $VP \rightarrow V[-_adjp] \ ADJP$
9. $VP[inf] \rightarrow TO \ VP[base]$  
10. $ADJP \rightarrow ADJ$
11. $ADJP \rightarrow ADJ[-_vp:inf] \ VP[inf]$

Head features for $S$, $VP$: VFORM, AGR  
Head features for $NP$: AGR

Grammar 4.7 A simple grammar in the abbreviated form
Grammar 4.8 The expanded grammar showing all features

introduced. The head features for each category are declared at the bottom of the figure. This grammar is an abbreviation of Grammar 4.8. Consider how rule 1 in Grammar 4.7 abbreviates rule 1 in Grammar 4.8. The abbreviated rule is

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 97]
Figure 4.9 Two sample parse trees with feature values

$S[-inv] \rightarrow (NP \ AGR \ ?a) \ (VP \ [{pres \ past}] \ AGR \ ?a)$

The unique values can be expanded in the obvious way: the value [-inv] becomes (INV -) and the value [{pres past}] becomes (VP [past]).
past) becomes (VFORM ?v{pres past}). The head features for S are AGR and VFORM, so these features must be added to the S and VP head. The resulting rule is

\[(S \; INV \; VFORM \; ?v\{pres \; past\} \; AGR \; ?a) \rightarrow \]

\[(NP \; AGR \; ?a)\]

\[(VP \; VFORM \; ?v \; \{pres \; past\} \; AGR \; ?a)\]

as shown in Grammar 4.8.

The abbreviated form is also very useful for summarizing parse trees. For instance, Figure 4.9 shows the parse trees for two of the previous sample sentences, demonstrating that each is an acceptable sentence.

Consider why each of the ill-formed sentences introduced at the beginning of this section are not accepted by Grammar 4.7. Both "The men cries" and "The man cry" are not acceptable because the number agreement restriction on rule 1 is not satisfied: The NP constituent for the men has the AGR value 3p, while the VP cries has the AGR value 3s. Thus rule 1 cannot apply. Similarly, "the man cry" is not accepted by the grammar since the man has AGR 3s and the VP cry has as its AGR value a variable ranging over (1s 2s 1p 2p 3p). The phrase "the man saw to be happy" is not accepted because the verb saw has a SUBCAT value _np. Thus only rule 5 could be used to build a VP. But rule 5 requires an NP complement, and it is not possible for the words "to be happy" to be a legal NP.

The phrase "He wants" is not accepted since the verb wants has a SUBCAT value ranging over {_np_vp:inf _np_vp:inf}, and thus only rules 5, 6, and 7 could apply to build a VP. But all these rules require a nonempty complement of some kind. The phrase "He wants the man saw the dog" is not accepted for similar reasons, but this requires a little more analysis. Again, rules 5, 6, and 7 are possible with the verb wants. Rules 5 and 6 will not work, but rule 7 looks close, as it requires an NP and a VP[inf]. The phrase "the man" gives us the required NP, but "saw the dog" fails to be a VP[inf]. In particular, "saw the man" is a legal VP, but its VFORM feature will be past, not inf.

4.5 Parsing with Features
The parsing algorithms developed in Chapter 3 for context-free grammars can be extended to handle augmented context-free grammars. This involves generalizing the algorithm for matching rules to constituents. For instance, the chart-parsing algorithms developed in Chapter 3 all used an operation for extending active arcs with a new constituent. A constituent \( X \) could extend an arc of the form

\[
C \rightarrow C_1 \ldots C_i \circ X \ldots C_n
\]

to produce a new arc of the form

\[
C \rightarrow C_1 \ldots C_i X \circ \ldots C_n
\]

A similar operation can be used for grammars with features, but the parser may have to instantiate variables in the original arc before it can be extended by \( X \). The key to defining this matching operation precisely is to remember the definition of grammar rules with features. A rule such as

1. \((\text{NP } \text{AGR } ?a) \rightarrow \circ (\text{ART } \text{AGR } ?a) (\text{N } \text{AGR } ?a)\)

says that an NP can be constructed out of an ART and an N if all three agree on the AGR feature. It does not place any restrictions on any other features that the NP, ART, or N may have. Thus, when matching constituents against this rule, the only thing that matters is the AGR feature. All other features in the constituent can be ignored. For instance, consider extending arc 1 with the constituent

2. \((\text{ART ROOT A AGR 3s})\)

To make arc 1 applicable, the variable \(?a\) must be instantiated to 3s, producing

3. \((\text{NP AGR 3s}) \rightarrow \circ (\text{ART AGR 3s}) (\text{N AGR 3s})\)

This arc can now be extended because every feature in the rule is in constituent 2:

4. \((\text{NP AGR 3s}) \rightarrow (\text{ART AGR 3s}) \circ (\text{N AGR 3s})\)

Now, consider extending this arc with the constituent for the word dog:

5. \((\text{N ROOT DOG1 AGR 3s})\)

This can be done because the AGR features agree. This completes the arc

6. \((\text{NP AGR 3s}) \rightarrow (\text{ART AGR 3s}) (\text{N AGR 3s})\)

This means the parser has found a constituent of the form \((\text{NP AGR 3s})\).
This algorithm can be specified more precisely as follows: Given an arc $A$, where the constituent following the dot is called $NEXT$, and a new constituent $X$, which is being used to extend the arc,

a. Find an instantiation of the variables such that all the features specified in NEXT are found in $X$.

b. Create a new arc $A'$, which is a copy of $A$ except for the instantiations of the variables determined in step (a).

c. Update $A'$ as usual in a chart parser.

For instance, let $A$ be arc 1, and $X$ be the ART constituent 2. Then NEXT will be (ART AGR ?a). In step a, NEXT is matched against $X$, and you find that ?a must be instantiated to 3s. In step b, a new copy of $A$ is made, which is shown as arc 3. In step c, the arc is updated to produce the new arc shown as arc 4.

When constrained variables, such as ?a{3s 3p}, are involved, the matching proceeds in the same manner, but the variable binding must be one of the listed values. If a variable is used in a constituent, then one of its possible values must match the requirement in the rule. If both the rule and the constituent contain variables, the result is a variable ranging over the intersection of their allowed values. For instance, consider extending arc 1 with the constituent (ART ROOT the AGR ?v{3s 3p}), that is, the word "the". To apply, the variable ?a would have to be instantiated to ?v{3s 3p}, producing the rule

$$\text{(NP AGR ?v{3s 3p}) -> (ART AGR ?v{3s 3p}) o (N AGR ?v{3s 3p})}$$

This arc could be extended by (N ROOT dog AGR 3s), because ?v{3s 3p} could be instantiated by the value 3s. The resulting arc would be identical to arc 6. The entry in the chart for the is not changed by this operation. It still has the value ?v{3s 3p}. The AGR feature is restricted to 3s only in the arc.

Another extension is useful for recording the structure of the parse. Subconstituent features (1, 2, and so on, depending on which subconstituent is being added) are automatically inserted by the parser each time an arc is extended. The values of these features name subconstituents already in the chart.
Figure 4.10 The chart for "He wants to cry".

With this treatment, and assuming that the chart already contains two constituents, ARTL and Ni, for the words the and dog, the constituent added to the chart for the phrase the dog would be:

\[
(\text{NP AGR 3s} \\
1 \text{ ART1} \\
2 \text{ N1})
\]
where ART1 (ART ROOT the AGR {3s 3p}) and N1 = (N ROOT dog AGR {3s}). Note that the AGR feature of ART1 was not changed. Thus it could be used with other interpretations that require the value 3p if they are possible. Any of the chart-parsing algorithms described in Chapter 3 can now be used with an augmented grammar by using these extensions to extend arcs and build constituents. Consider an example. Figure 4.10 contains the final chart produced from parsing the sentence He wants to cry using Grammar 4.8. The rest of this section considers how some of the nonterminal symbols were constructed for the chart.

Constituent NP1 was constructed by rule 3, repeated here for convenience:

3. \((NP \text{ AGR } ?a) \rightarrow (PRO \text{ AGR } ?a)\)

To match the constituent PRO1, the variable \(?a\) must be instantiated to 3s. Thus the new constituent built is

\[
NP1: (\text{CAT NP AGR 3s 1 PRO1})
\]

Next consider constructing constituent VP1 using rule 4, namely

4. \((VP \text{ AGR } ?a \text{ VFORM } ?v) \rightarrow (V \text{ SUBCAT } \_\text{none AGR } ?a \text{ VFORM } ?v)\)

For the right-hand side to match constituent V2, the variable \(?v\) must be instantiated to base. The AGR feature of V2 is not defined, so it defaults to -. The new constituent is

\[
VP1: (\text{CAT VP AGR } - \text{ VFORM base 1 V2})
\]

Generally, default values are not shown in the chart. In a similar way, constituent VP2 is built from TO1 and VP1 using rule 9, VP3 is built from V1 and VP2 using rule 6, and S1 is built from NP1 and VP3 using rule 1.
4.6 Augmented Transition Networks

Features can also be added to a recursive transition network to produce an augmented transition network (ATN). Features in an ATN are traditionally called registers. Constituent structures are created by allowing each network to have a set of registers. Each time a new network is pushed, a new set of registers is created. As the network is traversed, these registers are set to values by actions associated with each arc. When the network is popped, the registers are assembled to form a constituent structure, with the CAT slot being the network name.

Grammar 4.11 is a simple NP network. The actions are listed in the table below the network. ATNs use a special mechanism to extract the result of following an arc. When a lexical arc, such as arc 1, is followed, the constituent built from the word in the input is put into a special variable named "*". The action

\[
\text{DET} := * 
\]

then assigns this constituent to the DET register. The second action on this arc,

\[
\text{AGR} := \text{AGR}* 
\]

assigns the AGR register of the network to the value of the AGR register of the new word (the constituent in "*").

Agreement checks are specified in the tests. A test is an expression that succeeds if it returns a nonempty value and fails if it returns the empty set or nil.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 102]
**Grammar 4.11** A simple NP network

<table>
<thead>
<tr>
<th>Arc</th>
<th>Test</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
<td>DET := *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AGR := AGR</td>
</tr>
<tr>
<td>2</td>
<td>AGR ∩ AGR</td>
<td>HEAD := *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AGR := AGR ∩ AGR</td>
</tr>
<tr>
<td>3</td>
<td>none</td>
<td>NAME := *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AGR := AGR</td>
</tr>
</tbody>
</table>
If a test fails, its arc is not traversed. The test on arc 2 indicates that the arc can be followed only if the AGR feature of the network has a non-null intersection with the AGR register of the new word (the noun constituent in "*").

Features on push arcs are treated similarly. The constituent built by traversing the NP network is returned as the value "*". Thus in Grammar 4.12, the action on the arc from S to S1,

\[
\text{SUBJ} := *
\]

would assign the constituent returned by the NP network to the register SUBJ. The test on arc 2 will succeed only if the AGR register of the constituent in the SUBJ register has a non-null intersection with the AGR register of the new constituent (the verb). This test enforces subject-verb agreement.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 103]
### Trace of S Network

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Position</th>
<th>Arc Followed</th>
<th>Registers Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>1</td>
<td>arc 4 succeeds</td>
<td>SUBJ ← (NP DET the HEAD dog AGR 3s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(for recursive call see trace below)</td>
<td><strong>MAIN-V</strong> ← saw</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>AGR</strong> ← 3p</td>
</tr>
<tr>
<td>5</td>
<td>S1</td>
<td>3</td>
<td>arc 5 (checks if 3p ∩ 3p)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>S2</td>
<td>4</td>
<td>arc 6 (for recursive call trace, see below)</td>
<td><strong>OBJ</strong> ← (NP NAME Jack AGR 3s)</td>
</tr>
<tr>
<td>9</td>
<td>S3</td>
<td>5</td>
<td>pop arc succeeds</td>
<td>returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(S SUBJ (NP DET the HEAD dog AGR 3s))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>MAIN-V saw</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>AGR</strong> 3p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>OBJ</strong> (NP NAME Jack AGR 3s))</td>
</tr>
</tbody>
</table>

### Trace of First NP Call: Arc 4

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Position</th>
<th>Arc Followed</th>
<th>Registers Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NP</td>
<td>1</td>
<td>1</td>
<td><strong>DET</strong> ← the</td>
</tr>
<tr>
<td>3</td>
<td>NP1</td>
<td>2</td>
<td>2 (checks if {3s 3p} ∩ 3p)</td>
<td><strong>AGR</strong> ← {3s 3p}</td>
</tr>
<tr>
<td>4</td>
<td>NP2</td>
<td>3</td>
<td>pop</td>
<td><strong>HEAD</strong> ← dog</td>
</tr>
</tbody>
</table>

### Trace of Second NP Call: Arc 6

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Position</th>
<th>Arc Followed</th>
<th>Registers Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NP</td>
<td>4</td>
<td>3</td>
<td><strong>NAME</strong> ← John</td>
</tr>
<tr>
<td>8</td>
<td>NP2</td>
<td>5</td>
<td>pop</td>
<td>returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(NP NAME John AGR 3s)</td>
</tr>
</tbody>
</table>

**Figure 4.13** Trace tests and actions used with *The 2 dog 3 saw 4 Jack 5*
With the lexicon in Section 4.3, the ATN accepts the following sentences:

    The dog cried.
    The dogs saw Jack.
    Jack saw the dogs.

Consider an example. A trace of a parse of the sentence "The dog saw Jack" is shown in Figure 4.13. It indicates the current node in the network, the current

<table>
<thead>
<tr>
<th>Arc</th>
<th>Test</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td>SUBJ := *</td>
</tr>
<tr>
<td>S2/1</td>
<td>AGR_SUBJ ∩ AGR.*</td>
<td>MOOD := DECL</td>
</tr>
<tr>
<td>S3/1</td>
<td>SUBCAT_MADV ∩ (_np _np _np)</td>
<td>MAIN-V := *</td>
</tr>
<tr>
<td>S3/2</td>
<td>SUBCAT_MADV ∩ _none</td>
<td>OBJ := *</td>
</tr>
<tr>
<td>S4/1</td>
<td>SUBCAT_MADV ∩ _np _np</td>
<td>SUBCAT := SUBCAT_MADV ∩ (_np _np _np)</td>
</tr>
<tr>
<td>S5/1</td>
<td></td>
<td>IOBJ := OBJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OBJ := *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODS := Append(MODS, *)</td>
</tr>
</tbody>
</table>
word position, the arc that is followed from the node, and the register manipulations that are performed for the successful parse. It starts in the S network but moves immediately to the NP network from the call on arc 4. The NP network checks for number agreement as it accepts the word sequence The dog. It constructs a noun phrase with the AGR feature plural. When the pop arc is followed, it completes arc 4 in the S network. The NP is assigned to the SUBJ register and then checked for agreement with the verb when arc 3 is followed. The NP "Jack" is accepted in another call to the NP network.

An ATN Grammar for Simple Declarative Sentences

Here is a more comprehensive example of the use of an ATN to describe some declarative sentences. The allowed sentence structure is an initial NP followed by a main verb, which may then be followed by a maximum of two NPs and many PPs, depending on the Verb. Using the feature system extensively, you can create a grammar that accepts any of the preceding complement forms, leaving the actual verb-complement agreement to the feature restrictions. Grammar 4.14 shows the S network. Arcs are numbered using the conventions discussed in Chapter 3. For instance, the arc S3/1 is the arc labeled 1 leaving node 83. The NP network in Grammar 4.15 allows simple names, bare plural nouns, pronouns, and a simple sequence of a determiner followed by an adjective and a head noun.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 105]
Grammar 4.15 The NP network

Allowable noun complements include an optional number of prepositional phrases. The prepositional phrase network in Grammar 4.16 is straightforward. Examples of parsing sentences with this grammar are left for the exercises.
Presetting Registers

One further extension to the feature-manipulation facilities in ATNs involves the ability to preset registers in a network as that network is being called, much like parameter passing in a programming language. This facility, called the **SENDR** action in the original ATN systems, is useful to pass information to the network that aids in analyzing the new constituent.

Consider the class of verbs, including want and pray, that accept complements using the infinitive forms of verbs, which are introduced by the word to. According to the classification in Section 4.2, this includes the following:

\[\text{Allen 1995 : Chapter 4 - Features and Augmented Grammars 106}\]

![Diagram of the PP network](http://www.uni-giessen.de/~g91062/Seminare/gk-cl/Allen95/al199504.htm (35 / 57) [2002-2-26 21:23:16])

**Grammar 4.16** The PP network

Grammar 4.16 The PP network

- _vp:inf_ Mary wants to have a party.
- _np_vp:inf_ Mary wants John to have a party.

In the context-free grammar developed earlier, such complements were treated as VPs with the VFORM value `inf`. To capture this same analysis in an ATN, you would need to be able to call a network corresponding to VPs but preset the VFORM register in that network to `inf`. Another common analysis of these constructs is to...
view the complements as a special form of sentence with an understood subject. In the first case it is Mary who would be the understood subject (that is, the host), while in the other case it is John. To capture this analysis, many ATN grammars preset the SUBJ register in the new S network when it is called.

4.7 Definite Clause Grammars

You can augment a logic grammar by adding extra arguments to each predicate to encode features. As a very simple example, you could modify the PROLOG rules to enforce number agreement by adding an extra argument for the number on every predicate for which the number feature is relevant. Thus you would have rules such as those shown in Grammar 4.17.

Consider parsing the noun phrase "the dog cried", which would be captured by the assertions

\[
\text{word}(\text{the}, 1, 2):- \\
\text{word}(\text{dog}, 2, 3):- 
\]

With these axioms, when the word the is parsed by rule 2 in Grammar 4.17, the number feature for the word is returned. You can see this in the following trace of the simple proof of

\[
\text{np}(1, \text{Number}, 3)
\]

Using rule 1, you have the following subgoals:

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 107]
1. \texttt{np(P1, Number, P3) :- art(P1, Number, P2), n(2, Number, P3)}
2. \texttt{art(l, Number, O) :- word(Word, l, O), isart(Word, Number)}
3. \texttt{isart(a, 3s) :-}
4. \texttt{isart(the, 3s) :-}
5. \texttt{isart(the, 3p) :-}
6. \texttt{n(l, Number, O) :- word(Word, l, O), isnoun(Word, Number)}
7. \texttt{isnoun(dog, 3s) :-}
8. \texttt{isnoun(dogs, 3p) :-}

**Grammar 4.17**

1. \texttt{s(P1, Number, s(Np, Vp), P3) :-}
   \texttt{np(P1, Number, Np, P2), vp(P2, Number, Vp, P3)}
2. \texttt{np(P1, Number, np(Art, N), P3) :-}
   \texttt{art(P1, Number1, Art, P2), n(P2, Number2, N, P3)}
3. \texttt{vp(P1, Number, vp(Verb), P2) :-}
   \texttt{v(P1, Verb, P2)}
4. \texttt{art(l, Number, art(Word), O) :-}
   \texttt{word(Word, l, O), isart(Word, Number)}
5. \texttt{n(l, Number, n(Word), O) :-}
   \texttt{word(Word, l, O), isnoun(Word, Number)}
6. \texttt{v(l, Number, v(Word), O) :-}
   \texttt{word(Word, l, O), isverb(Word, Number)}

**Grammar 4.18**

1. \texttt{art(l, Number, P2)
\[ n(P2, \text{Number}, 3) \]

The first subgoal succeeds by using rule 2 and proving

\[ \text{word}(\text{the}, 1, 2) \]
\[ \text{isart}(\text{the}, 3s) \]

which binds the variables in rule 2 as follows:

\[ \text{Number} \leftarrow 3s \]
\[ P2 \leftarrow 2 \]

Thus, the second subgoal now is

\[ n(2, 3s, 3) \]

Using rule 6, this reduces to the subgoals \text{word}(\text{Word}, 2, 3) and \text{isnoun}(\text{Word}, 3s), which are established by the input and rule 7, respectively, with \text{Word} bound to \text{dog}. Thus the parse succeeds and the number agreement was enforced.

The grammar can also be extended to record the structure of the parse by adding another argument to the rules. For example, to construct a parse tree, you could use the rules shown in Grammar 4.18. These rules would allow you to prove the following on the sentence "The dog cried":

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 108]
Figure 4.19 A tree representation of the structure

\[ S(1, 3s, s(np(\text{art(a)}), n(\text{dog})), v(\text{cried})), 4) \]

In other words, between positions 1 and 4 there is a sentence with number feature 3s and the structure

\[ s(np(\text{art(a)}), n(\text{dog})), v(\text{cried}) \]

which is a representation of the parse tree shown in Figure 4.19.

For specifying grammars, most logic-based grammar systems provide a more convenient format that is automatically converted into PROLOG clauses like those in Grammar 4.18. Since the word position arguments are on every predicate, they can be omitted and inserted by the system. Similarly, since all the predicates representing terminal symbols (for example, art, N, V) are defined systematically, the system can generate these rules automatically. These abbreviations are encoded in a format called definite clause grammars (DCGs).

For example, you could write Grammar 4.18 by specifying just three DCG rules. (The symbol "\(\rightarrow\)" is traditionally used to signal DCG rules.)

\[
\begin{align*}
    s \ (\text{Number}, s(Np, Vp)) & \rightarrow np(N1, Np), \ vp(N2, Vp) \\
    np \ (\text{Number}, np(\text{Art}, N)) & \rightarrow art(N1, Art), \ n(N2, N) \\
    vp \ (\text{Number}, vp(\text{Verb})) & \rightarrow v(\text{Number}, \text{Verb})
\end{align*}
\]
The result is a formalism similar to the augmented context-free grammars described in Section 4.3. This similarity can be made even closer if a feature/value representation is used to represent structure. For instance, let us replace all the argument positions with a single argument that is a feature structure. The DCG equivalent to the first five rules in Grammar 4.7 would be as shown in Grammar 4.20, where square parentheses indicate lists in PROLOG. Notice that rule 1 will not succeed unless the 5, NP, and VP all agree on the agr feature, because they have the same variable as a value.

Of course, this encoding will only work if every constituent specifies its feature values in the same order so that the feature lists unify. For example, if the

```
1. s((inv - agr Agr)) → np((agr Agr)), vp((agr Agr vform pres))
2. np((agr Agr2)) → art((agr Agr2)), n((agr Agr2))
3. np((agr Agr3)) → pro((agr Agr3))
4. vp((agr Agr4 vform Vf3)) →
   v((subcat_none agr Agr4 vform Vf3))
5. vp((agr Agr4 vform Vf3)) →
   v((subcat_nop agr Agr4 vform Vf3)) np()
```

**Grammar 4.20  A DCG version of Grammar 4.7**

Grammar 4.20 A DCG version of Grammar 4.7

S features consist of "inv", "agr", "inv", and "vform", every rule using an S must specify all these features in the same order. In a realistic grammar this would be awkward, because there will be many possible features for each constituent, and each one must be listed in every occurrence. One way to solve this problem is to extend the program that converts DCG rules to PROLOG rules so that it adds in any unspecified features with variable values. This way the user need only specify the features that are important.
4.8 Generalized Feature Systems and Unification Grammars

You have seen that feature structures are very useful for generalizing the notion of context-free grammars and transition networks. In fact, feature structures can be generalized to the extent that they make the context-free grammar unnecessary. The entire grammar can be specified as a set of constraints between feature structures. Such systems are often called unification grammars. This section provides an introduction to the basic issues underlying such formalisms.

The key concept of a unification grammar is the extension relation between two feature structures. A feature structure \( F_1 \) extends, or is more specific than, a feature structure \( F_2 \) if every feature value in \( F_1 \) is specified in \( F_2 \). For example, the feature structure

\[
(\text{CAT V} \\
\text{ROOT cry})
\]

extends the feature structure (CAT V), as its CAT value is V as required, and the ROOT feature is unconstrained in the latter feature structure. On the other hand, neither of the feature structures

\[
(\text{CAT V} \\
\text{ROOT cry}) \\
(\text{CAT V} \\
\text{VFORM pres})
\]

extend the other, because both lack information required by the other. In particular, the first lacks the VFORM feature required by the second, and the second lacks the ROOT feature required by the first.

Two feature structures unify if there is a feature structure that is an extension of both. The most general unifier is the minimal feature structure that is an extension of both. The most general unifier of the above two feature structures is

\[
(\text{CAT V} \\
\text{ROOT cry} \\
\text{VFORM pres})
\]
Note that this is an extension of both original feature structures, and there is no smaller feature structure that is an extension of both. In contrast, the structures

\[
\begin{align*}
(CAT &\ V \\
AGR &\ 3s)
\end{align*}
\]

\[
\begin{align*}
(CAT &\ V \\
AGR &\ 3p)
\end{align*}
\]

do not unify. There can be no FS that is an extension of both because they specify contradictory AGR feature values.

This formalism can be extended to allow simple disjunctive values (for example, \{3s 3p\}) in the natural way. For example, \((AGR 3s)\) extends \((AGR \{3s 3p\})\).

The notion of unification is all that is needed to specify a grammar, as all feature agreement checks and manipulations can be specified in terms of unification relationships. A rule such as \(S -* NP\ VP\) in Grammar 4.7 could be expressed in a unification grammar as

\[
\begin{align*}
X_0 &\rightarrow X_1 \ X_2 \\
CAT_0 &\ = S \\
CAT_1 &\ = NP \\
CAT_2 &\ = VP \\
AGR_0 &\ = AGR_1 = AGR_2 \\
VFORM_0 &\ = VFORM_2
\end{align*}
\]

This says that a constituent \(X_0\) can be constructed out of a sequence of constituents \(X_1\) and \(X_2\) if the CAT of \(X_0\) is 5, the CAT of \(X_1\) is NP, the CAT of \(X_2\) is VP, the AGR values of all three constituents are identical, and the VFORM values of constituents \(X_0\) and \(X_2\) are identical. If the CAT value is always specified, such rules can be abbreviated as follows:

\[
\begin{align*}
S &\rightarrow NP \ VP \\
AGR &\ = AGR_1 = AGR_2 \\
VFORM &\ = VFORM_2
\end{align*}
\]

where the CAT values are used in the rule. Also, the 0 subscript is omitted. Using these abbreviations, a subpart of Grammar 4.7 is rewritten as the unification grammar in Grammar 4.21. Since such grammars retain the structure of context-free rules, the standard parsing algorithms can be used on unification grammars. The next section shows how to interpret the feature equations to build the constituents.

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 111]
Grammar 4.21  A unification grammar

Grammar 4.21 unification grammar

Figure 4.22  Two noun phrase DAGs

Figure 4.22 Two noun phrase DAGs
Formal Development: Feature Structures as DAGs

The unification-based formalism can be defined precisely by representing feature structures as directed acyclic graphs (DAGs). Each constituent and value is represented as a node, and the features are represented as labeled arcs. Representations of the following two constituents are shown in Figure 4.22:

N1: (CAT N
  ROOT fish
  AGR {3s 3p})

N2: (CAT N
  AGR 3s)

The sources of a DAG are the nodes that have no incoming edges. Feature structure DAGs have a unique source node, called the root node. The DAG is said to be rooted by this node. The sinks of a DAG are nodes with no outgoing edges. The sinks of feature structures are labeled with an atomic feature or set of features (for example, {3s 3p}).

The unification of two feature structures is defined in terms of a graph-matching algorithm. This takes two rooted graphs and returns a new graph that is

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 112]

To unify a DAG rooted at node N_i with a DAG rooted at node N_j:

1. If N_i equals N_j, then return N_i and succeed.

2. If both N_i and N_j are sink nodes, then if their labels have a non-null intersection, return a new node with the intersection as its label. Otherwise, the DAGs do not unify.

3. If N_i and N_j are not sinks, then create a new node N. For each arc labeled F leaving N_i to node N_F_i,

   3a. If there is an arc labeled F leaving N_i to node N_F_i, then recursively unify N_F_i and N_F_j. Build an arc labeled F from N to the result of the recursive call.

   3b. If there is no arc labeled F from N_i, build an arc labeled F from N to N_F_i.

   3c. For each arc labeled F from N_j to node NF, where there is no F arc leaving N_j, create a new arc labeled F from N to N_F_j.

Figure 4.23 The graph unification algorithm
the unification of the two. The algorithm is shown in Figure 4.23. The result of applying this algorithm to nodes Ni and N2 in Figure 4.22 is the new constituent

\[ \text{N3: (CATN} \]
\[ \quad \text{ROOT fish} \]
\[ \quad \text{AGR 3s)} \]

You should trace the algorithm on this example by hand to see how it works. This is a very simple case, as there is only one level of recursion. The initial call with the nodes N1 and N2 is handled in step 3, and each recursive call simply involves matching of sink nodes in step 2.

With this algorithm in hand, the algorithm for constructing a new constituent using the graph unification equations can be described. Once this is developed, you can build a parser using any standard parsing algorithm. The algorithm to build a new constituent of category C using a rule with feature equations of form \( F_i = V \), where \( F_i \) indicates the \( F \) feature of the \( i \)th subconstituent, is shown as Figure 4.24.

Consider an example. Assume the following two constituents shown in Figure 4.25 are defined already. In LISP notation, they are

\[ \text{ART1: (CAT} \]
\[ \quad \text{ROOT the} \]
\[ \quad \text{AGR \{3s 3p\))} \]
\[ \text{N1: (CATN} \]
\[ \quad \text{ROOT fish} \]
\[ \quad \text{AGR \{3s 3p\})) \]

The new NP will be built using rule 2 in Grammar 4.21. The equations are

\[ \text{CAT}_0 = \text{NP} \]
\[ \text{CAT}_1 = \text{ART} \]
\[ \text{CAT}_2 = \text{N} \]
\[ \text{AGR} = \text{AGR}_1 = \text{AGR}_2 \]

The algorithm produces the constituent represented by the DAG in Figure 4.26.
Given a rule $X_0 \rightarrow X_1 \ldots X_n$ and set of feature equations of form $F_i = V$, where $SC_1, \ldots, SC_n$ are the subconstituents corresponding to $X_1, \ldots, X_n$, this algorithm builds a DAG that satisfies all the feature equations.

1. Create a node $CC_0$ to be the root of the new feature structure.
2. Make a copy of each DAG rooted by $SC_i$ (call the new root of each $CC_i$), and add an arc labeled $i$ from $CC_0$ to each $CC_i$.
3. For each feature equation of form $F_i = V$, where $V$ is a value, follow the $F$ link from node $CC_i$ to a node $N_i$, and unify $N_i$ with $V$.
4. For each feature equation (of form $F_i = G_j$),
   4a. If there is an $F$ link from $CC_i$ and a $G$ link from $CC_j$, then
       i. follow the $F$ link to node $N_i$ and the $G$ link to $N_j$;
       ii. unify $N_i$ and $N_j$, using the graph unification algorithm, to create new node $X$;
       iii. change all arcs pointing to either $N_i$ or $N_j$ to point to $X$;
   4b. If there is no $F$ link from $CC_i$, but there is a $G$ link from $CC_j$ to node $N_j$, create an $F$ link from $CC_i$ to $N_j$;
   4c. If there is no $G$ link from $CC_j$, but there is an $F$ link from $CC_i$ to $N_i$, create a $G$ link from $CC_j$ to $N_i$.

Figure 4.24 An algorithm to build a new constituent using feature equations

Figure 4.25 Lexical entries for *the* and *fish*
Figure 4.26 The graph for the NP the fish

Figure 4.26 The graph for the NP the fish

[Allen 1995 : Chapter 4 - Features and Augmented Grammars 114]
Continuing the example, assume that the VP is happy is analyzed similarly and is represented as in Figure 4.27.

To simplify the graph, the CAT arcs are not shown. Figure 4.28 shows the analysis of The fish is happy constructed from rule 1 in Grammar 4.21. The value of the AGR slot is now the same node for S1, NF1, ART1, N1, VP1, and V1. Thus the value of the AGR feature of CCL, for instance, changed when the AGR features of NP1 and VP1 were unified.

So far, you have seen unification grammars used only to mimic the behavior of the augmented context-free grammars developed earlier. But the unification framework is considerably richer because there is no requirement that rules be based on syntactic categories. For example, there is a class of phrases in English called **predicative** phrases, which can occur in sentences of the form

\[ \text{NP be ...} \]

This includes prepositional phrases (He is in the house), noun phrases (He is a traitor), and adjective phrases (He is happy). Grammars often include a binary feature, say PRED, that is true of phrases that can be used in this way. In a standard CFG you would need to specify a different rule for each category, as in

\[
\begin{align*}
\text{VP} & \rightarrow (V \text{ ROOT be}) (\text{NP PRED} +) \\
\text{VP} & \rightarrow (V \text{ ROOT be}) (\text{PP PRED} +)
\end{align*}
\]
\[ VP \rightarrow (V \textbf{ROOT} \text{be}) (\text{ADJP} \text{PRED} +) \]

With the unification grammar framework, one rule handles all the categories, namely

\[ X_0 \rightarrow X_1 X_2 \]

\[ \text{CAT}_0 = VP \]

\[ \text{CAT}_1 = V \]

\[ \text{ROOT}_1 = \text{be} \]

\[ \text{PRED}_2 = + \]

in which any constituent \( X_2 \) with the \(+PRED\) feature is allowed.
Of course, if the categories of constituents are not specified, it is not so clear how to adapt the standard CFG parsing algorithms to unification grammar. It can be shown that as long as there is a finite subset of features such that at least one of the features is specified in every constituent, the unification grammar can be converted into a context-free grammar. This set of features is sometimes called the context-free backbone of the grammar.

Another powerful feature of unification grammars is that it allows much more information to be encoded in the lexicon. In fact, almost the entire grammar can be encoded in the lexicon, leaving very few rules in the grammar. Consider all the rules that were introduced earlier to deal with different verb subcategorizations. These could be condensed to a few rules: one for verbs that subcategorize for one subconstituent, another for verbs that subcategorize for two, and so on. The actual category restrictions on the complement structure would be encoded in the lexicon entry for the verb. For example, the verb put might have a lexicon entry as follows:

\[
\text{put: (CAT V SUBCAT (FIRST (CAT NP) SECOND (CAT PP LOC +))})
\]

The general rule for verbs that subcategorize for two constituents would be

\[
\text{VP} \rightarrow \text{V} \ X2 \ X3 \quad 2 = \text{FIRST}_{\text{SUBCAT}1} \\
\quad 3 = \text{SECOND}_{\text{SUBCAT}1}
\]

This rule would accept any sequence \( \text{V} \ X2 \ X3 \), in which \( X2 \) unifies with the FIRST feature of the SUBCAT feature of \( X1 \) and \( X3 \) unifies with the SECOND feature of the SUBCAT feature of \( X1 \). If \( V \) is the verb put, this rule would require \( X2 \) to unify with (CAT NP) and \( X3 \) to unify with (CAT PP LOC +), as desired. Of course, this same rule would put completely different constraints on \( X2 \) and \( X3 \) for a verb like want, which would require \( X3 \) to unify with (CAT VP VFORM inf).

Such techniques can dramatically reduce the size of the grammar. For instance, a reasonable grammar would require at least 40 rules to handle verb subcategorizations. But no rule involves more than three subconstituents in the verb complement. Thus these 40 rules could be reduced to four in the unification grammar, including a rule for null complements. In addition, these same rules could be reused to handle all nouns and adjective subcategorizations, if you generalize the category restrictions on \( X1 \).
Summary

This chapter has extended the grammatical formalisms introduced in Chapter 3 by adding features to each constituent and augmenting the grammatical rules. Features are essential to enable the construction of wide-coverage grammars of natural language. A useful set of features for English was defined, and morphological analysis techniques for mapping words to feature structures (that is, constituents) were developed. Several different forms of augmentation were examined. The first allowed context-free rules to specify feature agreement restrictions in addition to the basic category, producing augmented context-free grammars. The standard parsing algorithms described in Chapter 3 can be extended to handle such grammars. The second technique produced augmented transition networks, in which different procedural tests and actions could be defined on each arc to manipulate feature structures. The final method was based on the unification of feature structures and was used to develop unification grammars.

Related Work and Further Readings

Augmented CFGs have been used in computational models since the introduction of attribute grammars by Knuth (1968), who employed them for parsing programming languages. Since then many systems have utilized annotated rules of some form. Many early systems relied on arbitrary LISP code for annotations, although the types of operations commonly implemented were simple feature
BOX 4.3 Lexical Functional Grammar

A linguistic theory that has been influential in the development of computational formalisms is lexical functional grammar (Kaplan and Bresnan, 1982), usually abbreviated as LFG. LFG can be viewed as a type of unification grammar. A typical LFG rule is as follows:

\[ S \rightarrow \text{NP} \quad \text{VP} \]
\[ (\uparrow \text{SUBJ}) = \downarrow \quad \uparrow = \downarrow \]

The up arrow (\(\uparrow\)) indicates the constituent named on the left-hand side of the rule (the S constituent); the down arrow (\(\downarrow\)) indicates the constituent to which the annotation is attached. Thus this rule is equivalent to the following in the notation:

\[ S \rightarrow \text{NP} \quad \text{VP} \quad \text{SUBJ} = 1 \]
\[ 0 = 2 \]

Note the unification of the entire S and VP structure, making them the same constituent. Computationally, this can be viewed as an efficient way to transfer all the registers from the VP to the S, but it has linguistic implications as well, as discussed in Kaplan and Bresnan (1982). The effect is that any further modification to the S or VP structure will affect both, since they are now the same constituent.

LFGs encode most of their information in the lexicon. In particular, lexical entries may indicate which slots they will fill in the constituent that contains them. For example, the entries for \(a\), \(the\), and \(bird\) might be as follows:

\[ a \quad \text{ART} \quad (\uparrow \text{SPEC}) = \text{INDEF} \]
\[ (\uparrow \text{AGR}) = (3s) \]

\[ the \quad \text{ART} \quad (\uparrow \text{SPEC}) = \text{DEF} \]

\[ bird \quad \text{N} \quad (\uparrow \text{AGR}) = (3s) \]
\[ (\uparrow \text{HEAD}) = \text{BIRD} \]

The up-arrow annotations actually fill in slots in the NP structure. Using the rule

\[ \text{NP} \rightarrow \text{ART} \quad \text{N} \]

on the phrase \(a\) \(bird\) would result in the AGR feature of the NP being set to \(3s\) when the word \(a\) is parsed, and then this value is unified with \(3s\) to check number agreement when the noun \(bird\) is parsed. With the article \(the\), no number agreement...
tests and structure building along the lines of those discussed here. Examples of such systems are Sager (1981) and Robinson (1982).

The particular features used in this chapter and the techniques for augmenting rules are loosely based on work in the generalized phrase structure grammar (GPSG) tradition (Gazdar, Klein, Pullum, and Sag, 1985). GPSG introduces a finite set of feature values that can be attached to any grammatical symbol. These play much the same role as the annotations developed in this book except that there are no feature names. Rather since all values are unique, they define both

\[ \text{NP} \{ \text{PL}, 3, F \} \]

Since the number of feature values is finite, the grammar is formally equivalent to a context-free grammar with a symbol for every combination of categories and features. One of the important contributions of GPSG, however, is the rich structure it imposes on the propagation of features. Rather than using explicit feature equation rules, GPSG relies on general principles of feature propagation that apply to all rules in the grammar. A good example of such a general principle is the head feature convention, which states that all the head features on the parent constituent must be identical to its head constituent. Another general principle enforces agreement restrictions between constituents. Some of the conventions introduced in this chapter to reduce the number of feature equations that must be defined by hand for each rule are motivated by these theoretical claims. An excellent survey of GPSG and LFG is found in Sells (1985).


The discussion of unification grammars is based loosely on work by Kay (1982) and the PATR-II system (Shieber, 1984; 1986). There is a considerable amount of active research in this area. An excellent survey of the area is found in Shieber (1986). There is also a growing body of work on formalizing different forms of feature structures. Good examples are Rounds (1988), Shieber (1992), and Johnson (1991).
Exercises for Chapter 4

1. (easy) Using Grammar 4.7, draw the complete charts resulting from parsing the two sentences The man cries and He wants to be happy, whose final analyses are shown in Figure 4.9. You may use any parsing algorithm you want, but make sure that you state which one you are using and that the final chart contains every completed constituent built during the parse.

2. (medium) Define the minimal set of lexicon entries for the following verbs so that, using the morphological analysis algorithm, all standard forms of the verb are recognized and no illegal forms are inadvertently produced. Discuss any problems that arise and assumptions you make, and suggest modifications to the algorithms presented here if needed.

<table>
<thead>
<tr>
<th>Base</th>
<th>Present Forms</th>
<th>Past</th>
<th>Past-Participle</th>
<th>Present-Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>go, goes</td>
<td>went</td>
<td>gone</td>
<td>going</td>
</tr>
<tr>
<td>sing</td>
<td>sing, sings</td>
<td>sang</td>
<td>sung</td>
<td>singing</td>
</tr>
<tr>
<td>bid</td>
<td>bid, bids</td>
<td>bid</td>
<td>bidden</td>
<td>bidding</td>
</tr>
</tbody>
</table>

3. (medium) Extend the lexicon in Figure 4.6 and Grammar 4.7 so that the following two sentences are accepted:

   He was sad to see the dog cry.

   He saw the man saw the wood with the saw.

Justify your new rules by showing that they correctly handle a range of similar cases. Either implement and test your extended grammar using the supplied parser, or draw out the full chart that would be constructed for each sentence by a top-down chart parser.
4. (medium)

a. Write a grammar with features that will successfully allow the following phrases as noun phrases:

three o’clock
quarter after eight
ten minutes to six
seven thirty-five
half past four

but will not permit the following:

half to eight three
twenty o’clock
ten forty-five after six

Specify the feature manipulations necessary so that once the parse is completed, two features, HOUR and MINUTES, are set in the NP constituent. If this requires an extension to the feature mechanism, carefully describe the extension you assume.

b. Choose two other forms of grammatical phrases accepted by the grammar. Find an acceptable phrase not accepted by your grammar. If any nongrammatical phrases are allowed, give one example.

5. (medium) English pronouns distinguish case. Thus I can be used as a sub-ject, and me can be used as an object. Similarly, there is a difference between he and him, we and us, and they and them. The distinction is not made for the pronoun you. Specify an augmented context-free grammar and lexicon for simple subject-verb-object sentences that allows only appropriate pronouns in the subject and object positions and does number agreement between the subject and verb. Thus it should accept (hit him, but not me love you. Your grammar should account for all the pronouns mentioned in this question, but it need have only one verb entry and need cover no other noun phrases but pronouns.

[Allen 1995: Chapter 4 - Features and Augmented Grammars 120]
a. Specify some words by category and give four structurally different sentences accepted by this network.

b. Specify an augmentation for this network in the notation defined in this chapter so that sentences with the main verb give are allowed only if the subject is animate, and sentences with the main verb be may take either an animate or inanimate subject. Show a lexicon containing a few words that can be used to demonstrate the network’s selectivity.

7. (medium) Using the following unification grammar, draw the DAGs for the two NP structures as they are when they are first constructed by the parser, and then give the DAG for the complete sentence (which will include all subconstituents of S as well) The fish is a large one. You may assume the lexicon in Figure 4.6, but define lexical entries for any words not covered there.

1. $S \rightarrow NP\ VP$  
   \[\text{INV} = -\]  
   \[\text{VFORM}_2 = \text{pres}\]  
   \[\text{AGR} = \text{AGR}_1 = \text{AGR}_2\]

2. $NP \rightarrow \text{ART} \ N$  
   \[\text{AGR} = \text{AGR}_1 = \text{AGR}_2\]

3. $NP \rightarrow \text{ART} \ \text{ADJ} \ N$  
   \[\text{AGR} = \text{AGR}_1 = \text{AGR}_3\]

4. $VP \rightarrow V \ NP$  
   \[\text{VFORM} = \text{VFORM}_1\]  
   \[\text{AGR} = \text{AGR}_1 = \text{AGR}_2\]  
   \[\text{ROOT} = \text{BE}_1\]
Chapter 5: Grammars for Natural Language

5.1 Auxiliary Verbs and Verb Phrases

5.2 Movement Phenomena in Language

5.3 Handling Questions in Context-Free Grammars

5.4 Relative Clauses

5.5 The Hold Mechanism in ATNs

5.6 Gap Threading

Summary

Related Work and Further Readings

Exercises for Chapter 5

Augmented context-free grammars provide a powerful formalism for capturing many generalities in natural language. This chapter considers several aspects of the structure of English and examines how feature systems can be used to handle them. Section 5.1 discusses auxiliary verbs and introduces features that capture the ordering and agreement constraints. Section 5.2 then discusses the general class of problems often characterized as movement phenomena. The rest of the chapter examines various approaches to handling
movement. Section 5.3 discusses handling yes/no questions and wh-questions, and Section 5.4 discusses relative clauses. The remaining sections discuss alternative approaches. Section 5.5 discusses the use of the hold mechanism in ATNs and Section 5.6 discusses gap threading in logic grammars.

5.1 Auxiliary Verbs and Verb Phrases

English sentences typically contain a sequence of auxiliary verbs followed by a main verb, as in the following:

I can see the house.
I will have seen the house.
I was watching the movie.
I should have been watching the movie.

These may at first appear to be arbitrary sequences of verbs, including have, be, do, can, will, and so on, but in fact there is a rich structure. Consider how the auxiliaries constrain the verb that follows them. In particular, the auxiliary have must be followed by a past participle form (either another auxiliary or the main verb), and the auxiliary be must be followed by a present participle form, or, in the case of passive sentences, by the past participle form. The auxiliary do usually occurs alone but can accept a base form following it (I did eat my carrots!). Auxiliaries such as can and must must always be followed by a base form. In addition, the first auxiliary (or verb) in the sequence must agree with the subject in simple declarative sentences and be in a finite form (past or present). For example, *I going, *we be gone, and *they am are all unacceptable.

This section explores how to capture the structure of auxiliary forms using a combination of new rules and feature restrictions. The principal idea is that auxiliary verbs have subcategorization features that restrict their verb phrase complements. To develop this, a clear distinction is made between auxiliary and main verbs. While some auxiliary verbs have many of the properties of regular verbs, it is important to distinguish them. For example, auxiliary verbs can be placed before an adverbial not in a sentence, whereas a main verb cannot:

I am not going!
You did not try it.
He could not have seen the car.
In addition, only auxiliary verbs can precede the subject NP in yes/no questions:

**Did you see the car? Can I try it?**

* Eat John the pizza?

In contrast, main verbs may appear as the sole verb in a sentence, and if made into a yes/no question require the addition of the auxiliary *do*:

<table>
<thead>
<tr>
<th>I ate the pizza.</th>
<th>Did I eat the pizza?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The boy climbed in the window.</td>
<td>Did the boy climb in the window?</td>
</tr>
<tr>
<td>I have a pen.</td>
<td>Do I have a pen?</td>
</tr>
</tbody>
</table>

The primary auxiliaries are based on the root forms "be" and "have". The other auxiliaries are called modal auxiliaries and generally appear only in the finite tense forms (simple present and past). These include the following verbs organized in pairs corresponding roughly to present and past verb forms "do (did)", "can (could)", "may (might)", "shall (should)", "will (would)", "must", "need", and "dare". In addition, there are phrases that also serve a modal auxiliary function, such as "ought to", "used to", and "be going to".

Notice that "have" and "be" can be either an auxiliary or a main verb by these tests. Because they behave quite differently, these words have different lexical entries as auxiliaries and main verbs to allow for different properties; for example, the auxiliary "have" requires a past-participle verb phrase to follow it, whereas the verb "have" requires an NP complement.

As mentioned earlier, the basic idea for handling auxiliaries is to treat them as verbs that take a VP as a complement. This VP may itself consist of another auxiliary verb and another VP, or be a VP headed by a main verb. Thus, extending Grammar 4.7 with the following rule covers much of the behavior:
VP -> (AUX COMPFORM ?s) (VP VFORM ?s)

The COMPFORM feature indicates the VFORM of the VP complement. The values of this feature for the auxiliaries are shown in Figure 5.1.

There are other restrictions on the auxiliary sequence. In particular, auxiliaries can appear only in the following order:

<table>
<thead>
<tr>
<th>Modal</th>
<th>have</th>
<th>be (progressive)</th>
<th>be (passive)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The song might have been being played as they left

[Allen 1995: Chapter 5 - Grammars for Natural Language 125]

To capture the ordering constraints, it might seem that you need eight special rules: one for each of the four auxiliary positions plus four that make each optional. But it turns out that some restrictions can fall out from the feature restrictions. For instance, since modal auxiliaries do not have participle forms, a modal auxiliary can never follow "have" or "be" in the auxiliary sequence. For example, the sentence

* He has might see the movie already.

violates the subcategorization restriction on "have". You might also think that the auxiliary "have" can never appear in its participle forms, as it must either be first in the sequence (and be finite) or follow a modal and be an infinitive. But this is only true for simple matrix clauses. If you consider auxiliary sequences appearing in VP complements for certain verbs, such as "regret", the participle forms of "have" as an auxiliary can be required, as in

I regret having been chosen to go.

As a result, the formulation based on subcategorization features over-generates. While it accepts any legal sequence of auxiliaries, it would also accept

* I must be having been singing.

This problem can be solved by adding new features that further restrict the complements of the auxiliary "be" so that they do not allow additional auxiliaries (except "be" again for the passive). A binary head feature MAIN could be introduced that is + for any main verb, and - for auxiliary verbs. This way we can restrict the VP complement for "be" as follows:

VP -> AUX[be] VP[ing, +main]

The lexical entry for "be" would then have to be changed so that the original auxiliary rule does not apply. This could be done by setting its CQMPFORM feature to -. The only remaining problem is how to allow the passive. There are several possible approaches. We could, for instance, treat the "be" in the passive
construction as a main verb form rather than an auxiliary. Another way would be simply to add another rule allowing a complement in the passive form, using a new binary feature PASS, which is + only if the VP involves passive:

\[
VP \rightarrow \text{AUX[be]} \, \text{VP[ing, +pass]}
\]

The passive rule would then be:

\[
\text{VP[+pass]} \rightarrow \text{AUX [be]} \, \text{VP[pastprt, main]}
\]

While these new rules capture the ordering constraints well, for the sake of keeping examples short, we will generally use the first rule presented for handling auxiliaries throughout. While it overgenerates, it will not cause any problems for the points that are illustrated by the examples.

---

**Figure 5.2  Lexicon entries for some auxiliary verbs**
Figure 5.2 Lexicon entries for some auxiliary verbs

A lexicon for some of the auxiliary verbs is shown in Figure 5.2. All the irregular forms would need to be defined as well.

**o Passives**

The rule for passives in the auxiliary sequence discussed in the previous section solves only one of the problems related to the passive construct. Most verbs that include an NP in their complement allow the passive form. This form involves using the normal "object position" NP as the first NP in the sentence and either omitting the NP usually in the subject position or putting it in a PP with the preposition "by". For example, the active voice sentences

- *I will hide my hat in the drawer.*
- *I hid my hat in the drawer.*
- *I was hiding my hat in the drawer.*

can be rephrased as the following passive voice sentences:

- *My hat will be hidden in the drawer.*
- *My hat was hidden in the drawer.*
- *My hat was being hidden in the drawer.*

The complication here is that the VP in the passive construction is missing the object NP. One way to solve this problem would be to add a new grammatical rule for every verb subcategorization that is usable only for passive forms, namely all rules that allow an NP to follow the verb. A program can easily be written that would automatically generate such passive rules given a grammar. A

[Allen 1995: Chapter 5 - Grammars for Natural Language 127]
new binary head feature, PASSGAP, is defined that is + only if the constituent is missing the object NP. As usual, this feature would default to - if it is not specified in the left-hand side of a rule. The rule for a simple \_np subcategorization in Grammar 4.5 would then be realized as two rules in the new grammar:

\[
\text{VP}[-\text{passgap}] \rightarrow \text{V[\_np]} \ \text{NP} \\
\text{VP}[+\text{passgap}] \rightarrow \text{V[\_np]}
\]

Since the PASSGAP feature defaults to -, the only way the PASSGAP feature can become + is by the use of passive rules. Similarly, VP rules with lexical heads but with no NP in their complement are -PASSGAP, because they cannot participate in the passive construction. Figure 5.3 shows a fragment of Grammar 4.5 extended to handle auxiliaries and passives. The rules are as developed above except for additional variables in each rule to pass features around appropriately.
Figure 5.4 shows the analysis of the active voice sentence "Jack can see the dog" and the passive sentence "Jack was seen". Each analysis uses rule 1 for the S and rule 7 for the VP. The only difference is that the active sentence must use auxiliary rule 2 and NP rule 9, while the passive sentence must use auxiliary rule 5 and NP rule 8.

5.2 Movement Phenomena in Language

Many sentence structures appear to be simple variants of other sentence structures. In some cases, simple words or phrases appear to be locally reordered; sentences are identical except that a phrase apparently is moved from its expected

[Allen 1995: Chapter 5 - Grammars for Natural Language 128]
As you can readily see, yes/no questions appear identical in structure to their assertional counterparts except that the subject NPs and first auxiliaries have swapped positions. If there is no auxiliary in the assertional sentence, an auxiliary of root "do", in the appropriate tense, is used:

Jack is giving Sue a back rub.  
Is Jack giving Sue a back rub?

John went to the store.  
Did John go to the store?

As you can readily see, yes/no questions appear identical in structure to their assertional counterparts except that the subject NPs and first auxiliaries have swapped positions. If there is no auxiliary in the assertional sentence, an auxiliary of root "do", in the appropriate tense, is used:

He will run in the marathon next year.  
Will he run in the marathon next year?

Henry goes to school every day.  
Does Henry go to school every day?

Taking a term from linguistics, this rearranging of the subject and the auxiliary is called subject-aux inversion.
For example, consider the wh-questions that are related to the assertion

The fat man will angrily put the book in the corner.

If you are interested in who did the action, you might ask one of these questions:

Which man will angrily put the book in the corner?
Who will angrily put the book in the corner?

On the other hand, if you are interested in how it is done, you might ask one of the following questions:

How will the fat man put the book in the corner?
In what way will the fat man put the book in the corner?

If you are interested in other aspects, you might ask one of these questions:

What will the fat man angrily put in the corner?
Where will the fat man angrily put the book?
In what corner will the fat man angrily put the book?
What will the fat man angrily put the book in?

Each question has the same form as the original assertion, except that the part being questioned is removed and replaced by a wh-phrase at the beginning of the sentence. In addition, except when the part being queried is the subject NP, the subject and the auxiliary are apparently inverted, as in yes/no questions. This similarity with yes/no questions even holds for sentences without auxiliaries. In both cases, a "do" auxiliary is inserted:

I found a bookcase.
Did I find a bookcase?
What did I find?

Thus you may be able to reuse much of a grammar for yes/no questions for wh-questions. A serious problem remains, however, concerning how to handle the fact that a constituent is missing from someplace later in the sentence. For example, consider the italicized VP in the sentence

What will the fat man angrily put in the corner?
While this is an acceptable sentence, "angrily put in the corner" does not appear to be an acceptable VP because you cannot allow sentences such as *"I angrily put in the corner". Only in situations like wh-questions can such a VP be allowed, and then it is allowed only if the wh-constituent is of the right form to make a legal VP if it were inserted in the sentence. For example, "What will the fat man angrily put in the corner?" is acceptable, but *"Where will the fat man angrily put in the corner?" is not.

If you constructed a special grammar for VPs in wh-questions, you would need a separate grammar for each form of VP and each form of missing constituent. This would create a significant expansion in the size of the grammar.

This chapter describes some techniques for handling such phenomena concisely. In general, they all use the same type of approach. The place where a subconstituent is missing is called the gap, and the constituent that is moved is called the filler. The techniques that follow all involve ways of allowing gaps in constituents when there is an appropriate filler available. Thus the analysis of the VP in a sentence such as "What will the fat man angrily put in the corner?" is parsed as though it were "angrily put what in the corner", and the VP in the sentence "What will the fat man angrily put the book in?" is parsed as though the VP were "angrily put the book in what".

There is further evidence for the correctness of this analysis. In particular, all the well-formedness tests, like subject-verb agreement, the case of pronouns (who vs. whom), and verb transitivity, operate as though the wh-term were actually filling the gap. For example, you already saw how it handled verb transitivity. The question "What did you put in the cupboard?" is acceptable even though "put" is a transitive verb and thus requires an object. The object is a gap filled by the wh-term, satisfying the transitivity constraint. Furthermore, a sentence where the object is explicitly filled is unacceptable:

* What did you put the bottle in the cupboard?

This sentence is unacceptable, just as any sentence with two objects for the verb "put" would be unacceptable. In effect it is equivalent to

* You put what the bottle in the cupboard?

Thus the standard transitivity tests will work only if you assume the initial wh--term can be used to satisfy the constraint on the standard object NP position.

Many linguistic theories have been developed that are based on the intuition that a constituent can be moved from one location to another. As you explore these techniques further, you will see that significant generalizations can be made that greatly simplify the construction of a grammar. Transformational grammar (see Box 5.1) was based on this model. A context-free grammar generated a base sentence; then a set of transformations converted the resulting syntactic tree into a different tree by moving constituents. Augmented transition networks offered a new formalism that captured much of the behavior in a more computationally effective manner. A new structure called the hold list was introduced that allowed a constituent to be saved and used later in the parse by a new arc called the virtual (VIR) arc. This was the predominant computational mechanism for quite some time. In the early 1980s, however, new techniques were developed in linguistics that strongly influenced current computational systems.
The first technique was the introduction of \textit{slash} categories, which are complex nonterminals of the form X/Y and stand for a constituent of type X with a subconstituent of type Y missing. Given a context-free grammar, there is a simple algorithm that can derive new rules for such complex constituents. With this in hand, grammar writers have a convenient notation for expressing the constraints arising from unbounded movement. Unfortunately, the size of the resulting grammar can be significantly larger than the original grammar. A better approach uses the feature system. Constituents are stored in a special feature called GAP and are passed from constituent to subconstituent to allow movement. In this analysis the constituent S/NP is shorthand for an S constituent with the GAP feature NP. The resulting system does not require expanding the size of

\begin{boxedtext}
\textbf{BOX 5.1 Movement in Linguistics}

The term \textit{movement} arose in transformational grammar (TG). TG posited two distinct levels of structural representation: surface structure, which corresponds to the actual sentence structure, and deep structure. A CFG generates the deep structure, and a set of transformations map the deep structure to the surface structure. For example, the deep structure of "Will the cat scratch John?" would be:

The yes/no question is then generated from this deep structure by a transformation expressed schematically as follows:
\end{boxedtext}
In early TG (Chomsky, 1965), transformations could do many operations on trees, including moving, adding, and deleting constituents. Besides subj-aux inversion, there were transformational accounts of passives, wh-questions, and embedded sentences. The modern descendants of TG do not use transformations in the same way. In government-binding (GB) theory, a single transformation rule, Move-N, allows any constituent to be moved anywhere! The focus is on developing con-strains on movement that prohibit the generation of illegal surface structures.

the grammar significantly and appears to handle the phenomena quite well. The following sections discuss two different feature-based techniques for context-free grammars and the hold-list technique for ATNs in detail.
BOX 5.2 Different Types of Movement

While the discussion in this chapter will concentrate on wh-questions and thus will examine the movement of wh-terms extensively, the techniques discussed are also needed for other forms of movement as well. Here are some of the most common forms of movement discussed in the linguistics literature. For more details, see a textbook on linguistics, such as Baker (1989).

- **wh-movement** - move a wh-term to the front of the sentence to form a wh-question

  - *I never liked this picture.*
  - *This picture, I never liked.*

- **topicalization** - move a constituent to the beginning of the sentence for emphasis, as in

  - *I will see you tomorrow.*
  - *Tomorrow, I will see you.*

- **adverb preposing** - move an adverb to the beginning of the sentence, as in

  - *A book discussing evolution was written.*
  - *A book was written discussing evolution.*

As you consider strategies to handle movement, remember that constituents cannot be moved from any arbitrary position to the front to make a question. For example,

- *The man who was holding the two balloons will put the box in the corner.*

is a well-formed sentence, but you cannot ask the following question, where the gap is indicated by a dash:

- *What will the man who was holding - put the box in the corner?*

This is an example of a general constraint on movement out of relative clauses.
The goal is to extend a context-free grammar minimally so that it can handle questions. In other words, you want to reuse as much of the original grammar as possible. For yes/no questions, this is easily done. You can extend Grammar 4.7 with one rule that allows an auxiliary before the first NP and handles most examples:

$$ S \ [+\text{inv}] \rightarrow (\text{AUXAGR } ?a \ \text{SUBCAT } ?v) \ (\text{NP AGR } ?a) \ (\text{VP VFORM } ?v) $$

This enforces subject-verb agreement between the AUX and the subject NP, and ensures that the VP has the right VFORM to follow the AUX. This one rule is all

[Allen 1995: Chapter 5 - Grammars for Natural Language 133]

that is needed to handle yes/no questions, and all of the original grammar for assertions can be used directly for yes/no questions.

As mentioned in Section 5.2, a special feature GAP is introduced to handle wh-questions. This feature is passed from mother to subconstituent until the appropriate place for the gap is found in the sentence. At that place, the appropriate constituent can be constructed using no input. This can be done by introducing additional rules with empty right-hand sides. For instance, you might have a rule such as

$$(\text{NP GAP } ((\text{CAT NP}) (\text{AGR } ?a)) \ \text{AGR } ?a) \rightarrow \odot$$

which builds an NP from no input if the NP sought has a GAP feature that is set to an NP. Furthermore, the AGR feature of this empty NP is set to the AGR feature of the feature that is the value of the GAP. Note that the GAP feature in the mother has another feature structure as its value. To help you identify the structure of such complex values, a smaller font size is used for feature structures acting as values.

You could now write a grammar that passed the GAP feature appropriately. This would be tedious, however, and would also not take advantage of some generalities that seem to hold for the propagation of the GAP feature. Specifically, there seem to be two general ways in which the GAP feature propagates, depending on whether the head constituent is a lexical or nonlexical category. If it is a nonlexical category, the GAP feature is passed from the mother to the head and not to any other subconstituents. For example, a typical S rule with the GAP feature would be

$$(S \ \text{GAP } ?g) \rightarrow (\text{NP GAP-}) \ (\text{VP GAP } ?g)$$

The GAP can be in the VP, the head subconstituent, but not in the NP subject. For rules with lexical heads, the gap may move to any one of the nonlexical subconstituents. For example, the rule for verbs with a _np_vp:inf complement,

$$ \text{VP} \rightarrow \text{V[_np_vp:inf] NP PP} $$

would result in two rules involving gaps:

$$(\text{VP GAP } ?g) \rightarrow \text{V[_np_vp:inf] (NP GAP } ?g) \ (\text{PP GAP-})$$
\[(VP\ GAP\ ?g)\rightarrow V[_np vp:inf]\ (NP\ GAP\ -)\ (PP\ GAP\ ?g)\]

In other words, the GAP may be in the NP or in the PP, but not both. Setting the GAP feature in all but one subconstituent to - guarantees that a gap can be used in only one place.

An algorithm for automatically adding GAP features to a grammar is shown in Figure 5.5. Note that it does not modify any rule that explicitly sets the GAP feature already, allowing the grammar designer to introduce rules that do not follow the conventions encoded in the algorithm. In particular, the rule for subject-aux inversion cannot allow the gap to propagate to the subject NP.

[Allen 1995: Chapter 5 - Grammars for Natural Language 134]

For each rule \(Y \rightarrow X_1 \ldots H_i \ldots X_n\) with head constituent \(H_i\)

1. If the rule specifies a GAP feature in some constituent already, then skip.
2. If the head \(H_i\) is not a lexical category, then add a GAP feature to the head and the
   mother, and -GAP to the other subconstituents, producing a rule of form:
\[
(Y\ GAP\ ?g)\rightarrow (X_1\ GAP\ -)\ldots (H_i\ GAP\ ?g)\ldots (X_n\ GAP\ -)
\]
3. If the head \(H_i\) is a lexical category, then for each nonlexical subconstituent \(X_j\),
   add a rule of the form:
\[
(Y\ GAP\ ?g)\rightarrow (X_1\ GAP\ -)\ldots (X_j\ GAP\ ?g)\ldots (X_n\ GAP\ -)
\]

\[\text{Figure 5.5 An algorithm for adding GAP features to a grammar}\]

Using this procedure, a new grammar can be created that handles gaps. All that is left to do is analyze where the fillers for the gaps come from. In wh-questions, the fillers are typically NPs or PPs at the start of the sentence and are identified by a new feature WH that identifies a class of phrases that can introduce questions. The WH feature is signaled by words such as who, what, when, where, why, and how (as in how many and how carefully). These words fall into several different grammatical categories, as can be seen by considering what type of phrases they replace. In particular, who, whom, and what can appear as pronouns and can be used to specify simple NPs:

\[\text{Who ate the pizza?}\]

\[\text{What did you put the book in?}\]

The words "what" and "which" may appear as determiners in noun phrases, as in

http://www.uni-giessen.de/~g91062/Seminare/gk-cl/Allen95/al199505.htm (16 / 47) [2002-2-26 21:24:51]
What book did he steal?

Words such as "where" and "when" appear as prepositional phrases:

Where did you put the book?

The word "how" acts as an adverbial modifier to adjective and adverbial phrases:

How quickly did he run?

Finally, the word "whose" acts as a possessive pronoun:

Whose book did you find?

The wh-words also can act in different roles. All of the previous examples show their use to introduce wh-questions, which will be indicated by the WH feature value Q. A subset of them can also be used to introduce relative clauses, which will be discussed in Section 5.4. These will also have the WH feature value R. To make the WH feature act appropriately, a grammar should satisfy the following constraint: If a phrase contains a subphrase that has the WH feature, then the larger phrase also has the same WH feature. This way, complex phrases

![Figure 5.6 A lexicon for some of the wh-words](http://www.uni-giessen.de/~g91062/Seminare/gk-cl/Allen95/al199505.htm (17 / 47) [2002-2-26 21:24:51])
Grammar 5.7 A simple NP and PP grammar handling wh-words

containing a subconstituent with a WH word also can be used in questions. For example, the sentence

In what store did you buy the picture?

is a legal question since the initial PP in what store has the WH value Q because the NP what store has the WH value Q (because the determiner what has the WH value Q). With this constraint, the final S constituent will also have the WH value Q, which will be used to indicate that the sentence is a wh-question. Figure 5.6 gives some lexical entries for some wh-words. Note the introduction of new lexical categories: QDET for determiners that introduce wh-terms, and PP-WRD for words like when that act like prepositional phrases. Grammar 5.7 gives some rules for NPs and PPs that handle the WH feature appropriately.
Grammar 5.8 The unexpanded S grammar for wh-questions

With the WH feature defined and a mechanism for modifying a grammar to handle the GAP feature, most wh-questions can be handled by adding only two new rules to the grammar. Here are the rules for NP- and PP-based questions:

\[
S \rightarrow (NP[Q,-gap] AGR ?a) (S[+inv] GAP (NP AGR ?a))
\]
\[ S \rightarrow (PP[Q,-gap] \text{ PFORM } ?p) \ (S[+inv] \text{ GAP } (PP \text{ PFORM } ?p)) \]

Both of these rules set the value of the GAP feature to a copy of the initial WH constituent so that it will be used to fill a gap in the S constituent. Since the S constituent must have the feature +INV, it must also contain a subject-aux inversion. Given the NP and PP rules in Grammar 5.7, Grammar 5.8 provides the S and VP rules to handle questions. Grammar 5.9 shows all the derived rules introduced by the procedure that adds the GAP features. Rules 11, 12, and 13 are not changed by the algorithm as they already specify the GAP feature.

### Parsing with Gaps

A grammar that allows gaps creates some new complications for parsing algorithms. In particular, rules that have an empty right-hand side, such as

\[ (NP \text{ AGR } ?a \text{ GAP } (NP \text{ AGR } ?a)) \rightarrow \]
Grammar 5.9 The S grammar for wh-questions with the GAP feature

may cause problems because they allow an empty NP constituent anywhere. In a bottom-up strategy, for instance, this rule could apply at any position (or many times at any position) to create NPs that use no input. A top-down strategy fares better, in that the rule would only be used when a gap is explicitly predicted. Instead of using such rules, however, the arc extension algorithm can be modified to handle gaps automatically. This technique works with any parsing strategy.

Consider what extensions are necessary. When a constituent has a GAP feature that matches the constituent itself, it must be realized by the empty constituent. This means that an arc whose next constituent is a gap can be extended immediately. For example, consider what happens if the following arc is suggested by the parser:

\[(VP \text{ GAP } ?g) \to V\_[np_{-vp:inf}] (\text{NP GAP } \to) (\text{VP[inf] GAP } ?g)\]

\[19. \ (\text{VP[inf] GAP } ?g) \to TO (\text{VP[base] GAP } ?g)\]

\[20. \ (VP \text{ GAP } ?g) \to V\_[np_{-pp:loc}] (\text{NP GAP } ?g) (\text{PP[loc]} \text{ GAP } \to)\]

\[20'. \ ((VP \text{ GAP } ?g) \to V\_[np_{-pp:loc}] (\text{NP GAP } \to) (\text{PP[loc]} \text{ GAP } ?g)\]

\[\text{Head features for S, VP: VFORM, AGR}\]

Grammar 5.9 The S grammar for wh-questions with the GAP feature

may cause problems because they allow an empty NP constituent anywhere. In a bottom-up strategy, for instance, this rule could apply at any position (or many times at any position) to create NPs that use no input. A top-down strategy fares better, in that the rule would only be used when a gap is explicitly predicted. Instead of using such rules, however, the arc extension algorithm can be modified to handle gaps automatically. This technique works with any parsing strategy.

Consider what extensions are necessary. When a constituent has a GAP feature that matches the constituent itself, it must be realized by the empty constituent. This means that an arc whose next constituent is a gap can be extended immediately. For example, consider what happens if the following arc is suggested by the parser:

\[(VP \text{ GAP (NP AGR 3s))}\]

\[\to V\_[np_{-pp:loc}] o (\text{NP GAP (NP AGR 3s)) PP[LOC]}\]

The next constituent needed is an NP, but it also has a GAP feature that must be an NP. Thus the constituent must be empty. The parser inserts the constituent

\[(NP \text{ AGR 3s EMPTY } +)\]

[Allen 1995: Chapter 5 - Grammars for Natural Language 138]

BOX 5.3 The Movement Constraints

In linguistics the principles that govern where gaps may occur are called island constraints. The term draws on the metaphor of constituent movement. An island is a constituent from which no subconstituent can move out (just as a person cannot walk off an island). Here are a few of the constraints that have been
The A over A Constraint - No constituent of category A can be moved out of a constituent of type A. This means you cannot have an NP gap within an NP, a PP gap within a PP, and so on, and provides justification for not allowing non-null constituents of the form NP/NP, PP/PP, and so on. This disallows sentences such as

* What book$_1$ did you meet the author of $-$$_1$?

Complex-NP Constraint - No constituent may be moved out of a relative clause or noun complement. This constraint disallows sentences like

* To whom$_1$ did the man who gave the book $-$$_1$ laughed?

where the PP to whom would have been part of the relative clause who gave the book to whom (as in The man who gave the book to John laughed).

Sentential Subject Constraint - No constituent can be moved out of a constituent serving as the subject of a sentence. This overlaps with the other constraints when the subject is an NP, but non-NP subjects are possible as well, as in the sentence For me to learn these constraints is impossible. This constraint eliminates the possibility of a question like

What$_1$ is for me to learn $-$$_1$ impossible?

Wh-Island Constraint - No constituent can be moved from an embedded sentence with a wh-complementizer. For example, while Did they wonder whether I took the book? is an acceptable sentence, you cannot ask

*What$_1$ did they wonder whether I took $-$$_1$?

Coordinate Structure Constraint - A constituent cannot be moved out of a coordinate structure. For example, while Did you see John and Sam? is an acceptable sentence, you cannot ask

*Who$_1$ did you see John and $-$$_1$?

Note that these constraints apply to all forms of movement, not just wh-questions. For example, they constrain topicalization and adverb preposing as well.

to the chart, which can then extend the arc to produce the new arc

\[(VP \text{ GAP} (NP \text{ AGR 3s})) \rightarrow V[_np \text{ pp:loc}] (NP \text{ GAP} \text{ (NP AGR 3s)}) \circ PP(LOC)\]

A more precise specification of the algorithm is given in Figure 5.10.
Consider parsing "Which dogs did he see?" using the bottom-up strategy. Only the rules that contribute to the final analysis will be considered. Rule 7 in Grammar 5.7 applies to the constituent QDET1 (which) to build a constituent DET1, and rule 3 applies to the constituent N1 (dogs) to build a constituent CNP1, which then is combined with DET1 by rule 2 to build an NP constituent of the form

**NP1:** (NP AGR 3p
  
  WH Q
  
  1 QDET1
  
  2 CNP1)

This NP introduces an arc based on rule 12 in Grammar 5.8. The next word, "did", is an AUX constituent, which introduces an arc based on rule 11. The next word, "he", is a PRO, which creates an NP, NP2, using rule 1, and extends the arc based on rule 1. The chart at this stage is shown in Figure 5.11.
The word "see" introduces a verb V1, which can extend rule 16. This adds the arc labeled

\[(VP \text{ GAP } ?g) \rightarrow V[\_np] \circ (NP \text{ GAP } ?g)\]

Since the GAP value can match the required NP (because it is unconstrained), an empty NP is inserted in the chart with the form

\[\text{EMPTY-NP1: (NP AGR ?a GAP (NP AGR ?a)) EMPTY +}\]

This constituent can then be used to extend the VP arc, producing the constituent

\[\text{VP1: (VP VFORM inf GAP (NP AGR ?a) 1 V1 2 EMPTY-NP1)}\]

Now VP1 can extend the arc based on rule 11 to form the S structure:

[Allen 1995: Chapter 5 - Grammars for Natural Language 140]
Figure 5.11 The chart after the word he

Figure 5.11 The chart after the word he

\[
\text{S1: } (\text{S GAP (NP AGR ?a})
\]

\[
\begin{align*}
\text{INV} & + \\
1 & \text{AUX1} \\
2 & \text{NP2} \\
3 & \text{VP1})
\end{align*}
\]

Finally, NP1 and S1 can combine using rule 12 to produce an S constituent. The final chart is shown in Figure 5.12.

Thus, by using the WH and GAP features appropriately and by extending the parser to insert empty constituents as needed to fill gaps, the original grammar for declarative assertions can be extended to handle yes/no questions and wh-questions with only three new rules. The success of this approach depends on getting the right constraints on the propagation of the GAP feature. For instance, in a rule a gap should only be passed from the mother to exactly one of the subconstituents. If this constraint were not enforced, many illegal sentences would be allowed. There are some cases where this constraint appears to be violated. They are discussed in Exercise 6.

With a bottom-up parsing strategy, many empty constituents are inserted by the parser, and many constants with gaps are added to the chart. Most of these are not used in any valid analysis of the sentence. The top-down strategy greatly reduces the number of empty constants proposed and results in a considerably smaller chart.
This section examines the structure of relative clauses. A relative clause can be introduced with a rule using a new category, REL:

\[
\text{CNP} \rightarrow \text{CNP REL}
\]

A large class of relative clauses can be handled using the existing grammar for questions as they involve similar constructs of a wh-word followed by an S structure with a gap. The main difference is that only certain
wh-words are allowed (who, whom, which, when, where, and whose) and the S-structure is not inverted. The special class of wh-words for relative clauses is indicated using the WH feature value R. Figure 5.6 showed lexicon entries for some of these words.

Given this analysis, the following rules

\[
\text{REL} \rightarrow (\text{NP} \ \text{WH} \ R \ AGR \ ?a) \ (S[-\text{inv}, \ fin] \ \text{GAP} \ (\text{NP} \ AGR \ ?a))
\]

\[
\text{REL} \rightarrow (\text{PP} \ \text{WH} \ R \ \text{PFORM} \ ?p) \ (S[-\text{inv}, \ fin] \ \text{GAP} \ (\text{PP} \ \text{PFORM} \ ?p))
\]

handle relative clauses such as

The man who we saw at the store
The exam in which you found the error
The man whose book you stole

[Allen 1995: Chapter 5 - Grammars for Natural Language 142]

**BOX 5.4 Feature Propagation In GPSG**

GPSG (generalized phrase structure grammar) (Gazdar et al., 1985) formalizes all feature propagation in terms of a set of general principles. The GAP feature corresponds to the SLASH feature in GPSG. Their claim is that no special mechanism is needed to handle the SLASH feature beyond what is necessary for other features. In particular, the SLASH feature obeys two general GPSG principles. The head feature principle, already discussed, requires all head features to be shared between a head subconstituent and its mother constituent. The foot feature principle requires that if a foot feature appears in any subconstituent, it must be shared with the mother constituent. The constraints on the propagation of the SLASH feature result from it being both a head feature and a foot feature.

GPSG uses meta-rules to derive additional rules from a basic grammar. Meta-rules only apply to rules with lexical heads. One meta-rule accounts for gaps by taking an initial rule, for example,

\[
\text{VP} \rightarrow \text{V} \ \text{NP}
\]

and producing a new rule

\[
\text{VP/NP} \rightarrow \text{V} \ \text{NP}[+\text{NULL}]\]

where VP/NP is a VP with the SLASH feature NP, and the feature +NULL indicates that the constituent is "phonologically null", that is, not stated or present in the sentence.

GPSG meta-rules are used for other phenomena as well. For example, one meta-rule generates rules for passive voice sentences from rules for active voice sentences, while another generates subject-aux inversion rules from noninverted S rules. Mets-rules play much the same role as transformations, but they are limited to act on only a single rule at a time, and that rule must have a lexical head, whereas transformations may operate on arbitrary trees. The passive meta-rule looks something like this:

http://www.uni-giessen.de/~g91062/Seminare/gk-cl/Allen95/al199505.htm (27 / 47) [2002-2-26 21:24:51]
VP $\rightarrow$ V[TRANSITIVE] NP X $\Rightarrow$

VP[PASSGAP] $\rightarrow$ V X

Here the symbol X stands for any sequence of constituents. Thus this meta-rule says that if there is a rule for a VP consisting of a transitive verb, an NP, and possibly some other constituents, then there is another rule for a passive VP consisting of the same verb and constituents, except for the NP. As long as there is an upper limit to the number of symbols the variable X can match, you can show that the language described by the grammar expanded by all possible meta-rule applications remains a context-free language.

Because gaps don't propagate into the subject of a sentence, a new rule is needed to allow NPs like "The man who read the paper", where the wh-word plays the role of subject. This can be covered by the rule

[Allen 1995: Chapter 5 - Grammars for Natural Language 143]

REL $\rightarrow$ NP [R] VP[fin]

In addition, there is a common class of relative clauses that consist of "that" followed by an S with an NP gap, or a VP, as in "The man that we saw at the party" and "The man that read the paper". If you allow that to be a relative pronoun with WH feature R, then the above rules also cover these cases. Otherwise, additional rules must be added for "that".

Finally, there are relative clauses that do not start with an appropriate wh-phrase but otherwise look like normal relative clauses. Examples are

The paper John read
The damage caused by the storm
The issue creating the argument

The latter two examples involve what are often called reduced relative clauses. These require two additional rules allowing a relative clause to be a finite S with an NP gap or a VP in a participle form:

REL $\rightarrow$ (S[fin] GAP (NP AGR ?a))
REL $\rightarrow$ (VP VFORM {ing pastprt})

Notice that now there are two major uses of gaps: one to handle questions and the other to handle relative clauses. Because of this, you should be careful to examine possible interactions between the two. What happens to relative clauses within questions? They certainly can occur, as in

Which dog$_1$ did the man who$_2$ we saw$_2$ holding the bone feed$_1$?
The gaps are shown using dashes, and numbers indicate the fillers for the gaps. The existing grammar does allow this sentence, but it is not obvious how, since the GAP feature can only store one constituent at a time and the sentence seems to require storing two. The answer to this puzzle is that the way the GAP feature is propagated does not allow the initial Q constituent to move into the relative clause. This is because REL is not the head of the rule. In particular, the rule

\[
\text{CNP} \rightarrow \text{CNP REL}
\]

expands to the rule

\[
(\text{CNP GAP } ?g) \rightarrow (\text{CNP GAP } ?g) (\text{REL GAP})
\]

Thus the gap for the question cannot be found in the relative clause. As a result, when a new gap is proposed within the relative clause, there is no problem.

So the mechanism works in this case, but does it capture the structure of English well? In particular, can a question gap ever appear within a relative clause? The generally accepted answer appears to be no. Otherwise, a question like the following would be acceptable:

* Which dog_1 did the man_2 we saw —_2 petting —_1 laughed?

So the mechanism proposed here appears to capture the phenomena well.

[Allen 1995: Chapter 5 - Grammars for Natural Language 144]
Grammar 5.13 An NP network including wh-words

5.5 The Hold Mechanism in ATNs
Another technique for handling movement was first developed with the ATN framework. A data structure called the hold list maintains the constituents that are to be moved. Unlike GAP features, more than one constituent may be on the hold list at a single time. Constituents are added to the hold list by a new action on arcs, the hold action, which takes a constituent and places it on the hold list.

The hold action can store a constituent currently in a register (for example, the action HOLD SUBJ holds the constituent that is in the SUBJ register). To ensure that a held constituent is always used to fill a gap, an ATN system does not allow a pop arc to succeed from a network until any constituent held by an action on an arc in that network has been used. That is, the held constituent must have been used to fill a gap in the current constituent or in one of its subconstituents.

Finally, you need a mechanism to detect and fill gaps. A new arc called VIR (for virtual) that takes a constituent name as an argument can be followed if a constituent of the named category is present on the hold list. If the arc is followed successfully, the constituent is removed from the hold list and returned as the value of the arc in the identical form that a PUSH arc returns a constituent.

[Allen 1995: Chapter 5 - Grammars for Natural Language 145]
Grammar 5.14 An S network for questions and relative clauses

Grammar 5.14 shows an NP network that recognizes NPs involving wh-words as pronouns or determiners. As with CFGs, the feature WH is set to the value Q to indicate the NP is a wh-NP starting a question. It also accepts relative clauses with a push to the S network, shown in Grammar 5.14. This S network handles yes/no questions, wh-questions, and relative clauses. Wh-questions and relative clauses are processed by putting the initial NP (with a WH feature, Q, or R) onto the hold list, and later using it in a VIR arc. The network is organized so that all +INV sentences go through node S-INV, while all -INV sentences go through node VP. All wh-questions and relative clauses go through node WH-S and then are redirected back into the standard network based on whether or not the sentence is inverted. The initial NP is put on the hold list when arc Sf2 is followed. For noninverted questions, such as *Who ate the pizza?*, and for relative clauses in the NP of form *the man who ate the pizza*, the held NP is immediately used by the VIR arc WH-S/2. For other relative clauses, as in the NP *the man who we saw*, arc
WH-S/1 is used to accept the subject in the relative clause, and the held NP is used later on arc VCOMP/2. For inverted questions, such as *Who do I see?, arc WH-S/3 is followed to accept the auxiliary, and the held NP must be used later.

This network only accepts verbs with SUBCAT values _none and _np but could easily be extended to handle other verb complements. When extended, there would be a much wider range of locations where the held NP might be used. Figure 5.15 shows a trace of the sentence "The man who we saw cried". In parsing the relative clause, the relative pronoun who is held in step 5 and used by a VIR arc in step 8.

Note that this ATN would not accept *Who did the man see the boy?, as the held constituent who is not used by any VIR arc; thus the pop arc from the S network cannot be taken. Similarly, *The man who the boy cried ate the pie is unacceptable, as the relative pronoun is not used by a VIR arc in the S network that analyzes the relative clause.

Comparing the Methods

You have seen two approaches to handling questions in grammars: the use of the GAP feature and the use of the hold list in ATNs. To decide which is best, we must determine the criteria by which to evaluate them. There are three important considerations: coverage - whether the approach can handle all examples; selectivity - whether it rejects all ill-formed examples; and conciseness - how easily rules can be specified.

Under reasonable assumptions both methods appear to have the necessary coverage, that is, a grammar can be written with either approach that accepts any acceptable sentence. Given that this is the case, the important issues become selectivity and conciseness, and here there are some differences. For instance, one of the underlying assumptions of the GAP feature theory is that a symbol such as NP with an NP gap must be realized as the empty string; that is, you could not have an NP with an NP gap inside it. Such a structure could be parsable using the ATN hold list mechanism. In particular, a sentence such as *Who did the man who saw hit the boy?, while not comprehensible, would be accepted by the ATN in Grammars 5.13 and 5.14 because the hold list would contain two NPs; one for each occurrence of who. In the relative clause, the relative pronoun would be taken as the subject and the initial query NP would be taken as the

---

Trace of S Network

<table>
<thead>
<tr>
<th>Step</th>
<th>Node</th>
<th>Position</th>
<th>Arc Followed</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>S</td>
<td>1</td>
<td>S/1</td>
<td>SUBJ ← (NP DET the HEAD man AGR 3s MOD (S who we saw))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(for recursive call see trace below)</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>VP</td>
<td>6</td>
<td>VP/1</td>
<td>MAIN-V ← cried</td>
</tr>
<tr>
<td>12.</td>
<td>VCOMP</td>
<td>7</td>
<td>VCOMP/3 succeeds since no words left</td>
<td>returns (S SUBJ (NP DET the HEAD man)</td>
</tr>
</tbody>
</table>
Figure 5.15 A trace of an ATN parse for _The man who we saw cried_.

This sentence is not acceptable by any grammar using GAP features, however, because the GAP feature cannot be propagated into a relative clause since it is not the head constituent in the rule.
Even if it were the head, however, the sentence would still not be acceptable, since the erroneous analysis would require the GAP feature to have two values when it starts analyzing the relative clause.

The hold list mechanism in the ATN framework must be extended to capture these constraints, because there is no obvious way to prevent held constituents from being used anytime they are available. The only possible way to restrict it using the existing mechanism would be to use feature values to keep track of what held constituents are available in each context. This could become quite messy. You can, however, make a simple extension. You can introduce a new action - say, HIDE - that temporarily hides the existing constituents on the hold list until either an explicit action - say, UNHIDE - is executed, or the present constituent is completed. With this extension the ATN grammar to handle relative clauses could be modified to execute a HIDE action just before the hold action that holds the current constituent is performed.

In this case, however, it is not the formalism itself that embodies the constraints. Rather, the formalism is used to state the constraints; it could just as easily describe a language that violates the constraints. The theory based on GAP features, on the other hand, embodies a set of constraints in its definition of how features can propagate. As a result, it would be much more difficult to write a grammar that describes a language that violates the movement constraints.

While a true account of all the movement constraints is still an area of research, this simple analysis makes an important point. If a formalism is too weak, you cannot describe the language at all. If it is too strong, the grammar may become overly complicated in order to eliminate sentence structures that it can describe but that are not possible. The best solution, then, is a formalism that is just powerful enough to describe natural languages. In such a language many of the constraints that first appear to be arbitrary restrictions in a language might turn out to be a consequence of the formalism itself and need no further consideration.

While the technique of GAP feature propagation was introduced with context-free grammars, and the hold list mechanism was introduced with ATNs, these techniques are not necessarily restricted to these formalisms. For instance, you could develop an extension to CFGs that incorporates a hold list and uses it for gaps. Likewise, you might be able to develop some rules for GAP feature propagation in an ATN. This would be more difficult, however, since the GAP feature propagation rules depend on the notion of a head subconstituent, which doesn’t have a direct correlate in ATN grammars.

5.6 Gap Threading
A third method for handling gaps combines aspects of both the GAP feature approach and the hold list approach. This technique is usually called gap threading. It is often used in logic grammars, where two extra argument positions are added to each predicate—one argument for a list of fillers that might be used in the current constituent, and one for the resulting list of fillers that were not used after the constituent is parsed. Thus the predicate

\[ s(\text{position-in}, \text{position-out}, \text{fillers-in}, \text{fillers-out}) \]

is true only if there is a legal S constituent between position-in and position-out of the input. If a gap was used

```
1. s(In, Out, Fillers\text{in}, Fillers\text{out}) :- np(In, In1, Fillers\text{in}, Fillers\text{1}),
   vp(In1, Out, Fillers\text{1}, Fillers\text{out})
2. vp(In, Out, Fillers\text{in}, Fillers\text{out}) :- v(In, In1)
3. vp(In, Out, Fillers\text{in}, Fillers\text{out}) :- v(In, In1), np(In1, Out, Fillers\text{in}, Fillers\text{out})
4. np(In, Out, Fillers, Fillers) :- art(In, In1), cnp(In1, Out)
5. cnp(In, Out) :- n(In, In1), np-comp(In1, Out)
6. np-comp(In, In) :-
   (This covers the case where there is no NP complement.)
7. np-comp(In, Out) :- rel-intro(In, In1, Filler),
   s(In1, Out, (Filler nil), nil)
   (Here we hold the Rel-Intro constituent, and must use it in the following S.)
8. rel-intro(In, Out, [NP]) :- relpro(In, Out)
   (where relpro accepts any pronoun with WH feature R)
9. np(In, In, [NP | Fillers], Fillers) :-
   (This rule builds an empty np from a filler.)
```

**Grammar 5.16** A logic grammar using gap threading
to build the 5, its filler will be present in fillers-in, but not in fillers-out. For example, an S constituent with an NP gap would correspond to the predicate s(In, Out, (NP], nil). In cases where there are no gaps in a constituent, the fillers-in and fillers-out will be identical.

Consider an example dealing with relative clauses. The rules required are shown in Grammar 5.16. The various feature restrictions that would be needed to enforce agreement and subcategorization are not shown so as to keep the example simple.

To see these rules in use, consider the parse of the sentence *The man who we saw cried* in Figure 5.17. The relative clause is analyzed starting with step 7. Using rule 9, the word *who* is recognized as a relative pronoun, and the variable Filler is bound to the list [NP]. This filler is then passed into the embedded S (step 9), to the NP (step 10), and then on to the VP (step 12), since it is not used in the NP. From there it is passed to the NP predicate in step 14, which uses the filler according to rule 10. Note that no other NP rule could have applied at this point, because the filler must be used since the FillersOut variable is nil. Only rules that consume the filler can apply. Once this gap is used, the entire NP from positions 1 to 6 has been found and the rest of the parse is straightforward.

[Allen 1995: Chapter 5 - Grammars for Natural Language 150]
Just as a convenient notation was designed for definite clause grammars, which then could be simply translated into a PROLOG program, a notation has been designed to facilitate the specification of grammars that handle gaps. In particular, there is a formalism called extraposition grammar, which, besides allowing normal context-free rules, allows rules of the form

\[ \text{REL-MARK} \ldots \text{TRACE} \rightarrow \text{REL-PRO} \]

which essentially says that the constituent REL-MARK, plus the constituent TRACE later in the sentence, can be rewritten as a REL-PRO. Such a rule violates the tree structure of syntactic forms and allows the analysis shown in Figure 5.18 of the NP *the mouse that the cat ate*. Such rules can be compiled into a brie rrammar usin2 the gap-threading technique.
Consider how gap threading compares with the other approaches. Of course, by using additional arguments on
predicates, any of the approaches could be implemented. If viewed as simply an implementation technique,
then the gap-threading approach looks like a way to implement a hold list in a logic grammar. Since the
grammar designer can decide to propagate the hold list or not in the grammar, it provides the flexibility to
avoid some of the problems with the simple hold list mechanism. For instance, Grammar 5.16 does not pass the
fillers from outside a noun phrase into its relative clause. So problems like those discussed in the last section
can be avoided. It also would be possible to adopt the GAP feature introduction algorithm described in Section
5.3 so that it introduces gap-threading rules into a logic grammar. Thus, the approach provides the grammar
writer with great flexibility. Like the hold list approach, however, the propagation constraints must be explicitly
enforced rather than being a consequence of the formalism.
Summary

Many of the complex aspects of natural language can be treated as movement phenomena, where a constituent in one position is used to satisfy the constraints at another position. These phenomena are divided into two classes: bounded movement, including yes/no questions and passives; and unbounded movement, including wh-questions and relative clauses. The computational techniques developed for handling movement allow significant generalities to be captured between all of these different sentential forms. By using the feature system carefully, a basic grammar of declarative sentences can be extended to handle the other forms with only a few rules. Three different techniques were introduced to handle such phenomena. The first was the use of the special feature propagation rules using a feature called GAP. The second was the hold list mechanism in ATNs, which involves adding a new action (the hold action) and a new, arc type (the V1R arc). The third involved gap threading, with a hold-list-like structure passed from constituent to constituent using features. With suitable care, all of these approaches have been used successfully to build grammars of English with substantial coverage.

Related Work and Further Readings

The phenomena of unbounded dependencies has motivated a significant amount of research into grammatical formalisms. These fall roughly into several categories, depending on how close the grammar remains to the context-free grammars. Theories such as transformational grammar (Chomsky, 1965; Radford, 1981), for example, propose to handle unbounded dependencies completely outside the CFG framework. Theories such as lexical functional grammar (Kaplan and Bresnan, 1982) and generalized phrase structure grammar (GPSG) (Gazdar, 1982; Gazdar et al., 1985) propose methods of capturing the dependencies with the CFG formalism using features. There has also been con-siderable work in defining new formalisms that are slightly more
powerful than context-free grammars and can handle long-distance dependencies such as tree-adjoining grammars (TAGs) (see Box 5.5) and combinatory categorial grammars (Steedman, 1987).

The first reasonably comprehensive computational study of movement phenomena was in the ATN framework by Woods (1970). He went to some length to show that much of the phenomena accounted for by transformational grammar (Chomsky, 1965) could be parsed in ATNs using register testing and setting together with a hold list mechanism. The ATN section of this chapter is a simplified and cleaned-up account of that paper, incorporating later work by Kaplan (1973). Alternative presentations of ATNs can also be found in Bates (1978) and Winograd (1983).

Similar augmentation techniques have been adapted to systems based on CFGs, although in practice many of these systems have accepted many structures that are not reasonable sentences. This is because the grammars are built so that constituents are optional even in contexts where there could be no movement. The parser then depends on some ad hoc feature manipulation to eliminate some of these false positives, or on semantic interpretation to reject the ill-formed sentences that were accepted. A good example of how much can be done with this approach is provided by Robinson (1982).

BOX 5.5 Tree-Adjoining Grammar

Another approach to handling unbounded dependencies is tree-adjoining grammars (TAGs) (Joshi, 1985). There are no grammar rules in this formalism. Rather, there is a set of initial tree structures that describe the simplest sentences of the language, and a tree operation, called adjoining, that inserts one tree into another to create a more complex structure. For example, a simple initial tree is

```
S
  /\  
NP  VP
     /\  
V   NP
```

More complex sentences are derived using auxiliary trees, which capture the minimal forms of recursion in the language. For example, an auxiliary tree for allowing an adverbial in a verb phrase could be

![Auxiliary Tree Example](image1)

The adjunction operation involves inserting an auxiliary tree-capturing recursion of some constituent C into another tree that contains a constituent C. Adjoining the auxiliary tree for VP above into the initial tree for S produces the new tree

![Initial Tree](image2)

By basing the formalism on trees, enough context is provided to capture long-distance dependencies. In this theory there is no movement. Instead the constituents start off being close to each other, and then additional structure is inserted between them by the adjoining operation. The result is a formalism of slightly greater power than CFGs but definitely weaker than context-sensitive grammars.

The work on handling movement using gap threading and the definition of extraposition grammars can be found in Pereira (1981). Similar techniques can be found in many current parsers (such as Alshawi, 1992).
The section on using GAP feature propagation to handle movement is motivated from GPSG (Gazdar et al., 1985). In GPSG the propagation of features is determined by a set of general principles (see Box 5.4). Head-driven phrase structure grammar (Pollard and Sag, 1987; 1993) is a descendant of GPSG that is more oriented toward computational issues. It uses subcategorization information on the heads of phrases extensively and, by so doing, greatly simplifies the context-free grammar at the expense of a more complex lexicon.

Exercises for Chapter 5

1. (easy) Distinguish between bounded (local) movement and unbounded (nonlocal) movement. What extensions were added to the augmentation system to enable it to handle unbounded movement? Why is wh-movement called unbounded movement? Give examples to support your àkint.

2. (easy) Expand the following abbreviated constituents and rules into their full feature structure format.

   
   (S[-inv, Q] AGR ?a)
   
   NP[R, -gap]
   
   VP -> V[._np_s:inf] NP S[inf]

3. (medium) Using the grammar developed in Section 5.3, show the analyses of the following questions in chart form, as shown in Figure 5.12:

   In which town were you born?
   Where were you born?
   When did they leave?
   What town were you born in?
4. *(medium)* GPSG allows certain rules to have multiple head constituents. For instance, the rule for a conjunction between two verb phrases would contain two heads:

\[
\text{VP} \rightarrow \text{VP and VP}
\]

a. How does the presence of multiple heads affect the algorithm that produces the propagation of the GAP feature? In order to answer this question, consider some examples, such as the following sentences:

Who did you see and give the book to?
What man did Mary hate and Sue love?

Also consider that the following sentences are ill-formed:

* Who did you see and give the book to John?
* What man did Mary hate John and Sue love?

b. Write out the VP rule showing the GAP features, and then draw the chart for the sentence

Who did Mary see and Sue see?
using Grammar 5.8 augmented with your rule. You need only show the constituents that are used in the final analysis, but be sure to show all the feature values for each constituent.

5. (medium) Another phenomenon that requires a movement analysis is topicalization, which allows sentences of the form

   John, Mary saw.

   To John, Mary told the story.

   a. Argue that the same type of constraints that apply to wh-questions apply to topicalization. This establishes that the GAP feature propagation technique can be used.

   b. Do you need to introduce a new GAP-like feature to handle topicalization, or can the same GAP feature used for questions and relative clauses be re-used?

   c. Extend Grammar 5.8 to accept the forms of topicalization shown in the examples.

6. (medium) Consider the following ATN grammar for sentences. On arc NP3/I there is the action of holding the current noun phrase.

   a. Give a sequence of legal sentences that can be made arbitrarily long and still be accepted by this ATN.

   b. Which of the following sentences would be accepted by this ATN (assuming appropriate word categories for words)? For those that are accepted, outline the structure of the sentence as parsed by the network (that is, a plausible register value structure).
i. The man Mary hit hit the ball.
ii. The man the man hit the man hit.
iii. The man hit hit.
iv. Hit the man hit the man.

7. (hard) The following noun phrases arise in some dialogues between a computer operator and various users of the computer system.

Could you mount a magtape for me?

No ring please.

I am not exactly sure of the reason, but we were given a list of users we are not supposed to mount magtapes for, and you are on it.

Could you possibly retrieve the following two files? I think they were on our directory last night.

Any chance I can recover from the most recent system dump?

Extend the grammar developed in Sections 5.3 and 5.4 so that it accepts these noun phrases. Give lexicon entries for all the new words in the noun phrases, and show the final charts that result from parsing each NP.

8. (hard)

a. The grammar for parsing the auxiliary verb structure in English developed in Section 5.1 cannot handle
phrases of the following form:

- Jack *has to see* a doctor.
- The cat *had to be found*.
- Joe *has to be winning* the race.
- The book *would have had to have been found* by Jack.

Extend the grammar such that it accepts the preceding auxiliary sequences yet does not accept unacceptable sentences such as

- *Jack has to have to see* a doctor.
- *Janet had to played* the violin.
- *Will would to go* to the movies.

b. Perform a similar analysis, showing examples and counterexamples, of the use of phrases of the form be *going to* within the auxiliary system.