Slot Grammar makes it easier to write practical, broad-coverage natural language grammars, for the following reasons. (a) The system has a lexicalist character; although there are grammar rules, they are fewer in number and simpler because analysis is largely data-driven — through use of slots taken from lexical entries. (b) There is a modular treatment of different grammatical phenomena through different rule types, for instance rule types for expressing linear ordering constraints. This modularity also reduces the differences between the Slot Grammars of different languages. (c) Several grammatical phenomena, such as coordination and extraposition, are treated mainly in a language-independent shell provided with the system.

1 Introduction

The Slot Grammar system provides a convenient means for writing practical, broad-coverage grammars for natural languages. The grammar for a given language is simplified, and the differences between grammars of different languages are reduced, for the following reasons.

1 Slot Grammar has a highly lexicalist character. Grammatical analysis makes systematic use of slots (essentially syntactic relations) obtained from lexical entries. Although there are grammar rules, they tend to be fewer in number and simpler because the system is more data-driven.\footnote{This does not mean that complexity is just shoved to the lexicon. Mainly, it means making better use of lexical information, information that is needed anyway for applications of the grammar such as machine translation.}

2 There is a modular treatment of grammatical phenomena through different rule types. In particular, linear order is treated modularly by separate slot ordering rules. This not only
simplifies a given grammar, but also reduces differences across grammars because linear order is one of the main ways in which languages differ.

3) The Slot Grammar system has a language-independent shell, which includes not only the parser but also a treatment of several grammatical phenomena specific to natural language. In particular, the shell contains much of the treatment of coordination, left extraposition (left movement) and remote dependencies, implicit subjects, and punctuation. Of course there are aspects of these phenomena that depend on the particular natural language, so the general procedures are “parametrized” by calls to simple rules (mainly unit clauses) in specific grammars. One way of viewing the shell is that it makes the Slot Grammar system a higher-level, special-purpose system for writing natural language grammars. Unlike some augmented phrase structure grammar systems (such as DCGs), Slot Grammar would be inappropriate for the grammar of, say, computer programming languages. But its special-purpose features offer greater simplicity and compactness for natural language grammar.

4) The shell contains a parse evaluation system, extending Heidorn’s [1982] parse metric, which not only ranks final parses but is also used for pruning away unlikely partial analyses during parsing. (The parser is a bottom-up chart parser.) The Slot Grammar parse evaluator effects weighted preferences for complements over adjuncts and for parallelism in coordination, as well as for close attachment. The parse space pruning system contributes to the simplification of Slot Grammars for particular languages because it offers language-universal constraints which for some grammatical systems would go in the language-specific grammar.

The original work on Slot Grammar was done in 1976-78 and appeared in [McCord 1980]. This earlier system was developed without consideration of logic programming (and was implemented in Lisp). The logic-programming-based grammatical systems described in [McCord 1982, 1985, 1987] represented a combination of the original Slot Grammar techniques with the (augmented) phrase structure grammar techniques common in logic programming [Colmerauer 1978]. In this combined approach, Slot Grammar rules, expressed in terms of phrase structure rules and Prolog clauses, were used systematically for postmodification of open-class words; but elsewhere in the grammar, more standard phrase structure rules were used. In particular, the combined approach was used in the Modular Logic Grammar ModL [McCord 1985, 1986, 1989a].

Work on the current Slot Grammar system began in 1988. The idea has been to develop the original lexicalist, head-driven approach into a practical system taking good advantage of logic programming. The main motivation for the new version has been its use in the machine translation system LMT [McCord 1986, 1989a,c,d, McCord and Wolff 1988]. LMT originally used Modular Logic Grammar for source analysis, but has been revised to use Slot Grammar. This has been extremely useful in making LMT capable of dealing with multiple language pairs—because of the Slot Grammar shell and the lexicalism.

Slot Grammar has some features in common with currently popular grammatical systems such as LFG, FUG and HPSG (see Shieber 1986 for an overview), in which there are themes of
lexicalism and use of unification, and sometimes a dependency (head-driven) orientation. The original (1976-78) Slot Grammar work was done independently of these lines of work. The current Slot Grammar system differs from these related systems in several ways that will become apparent below. One general remark, though, is that (current) Slot Grammar could be said to be closer to Prolog — e.g. it uses term unification instead of attribute-value system unification. This may allow Slot Grammars to be more efficient and practical. And it does not entail a total sacrifice of formal neatness and purity, because the grammar rules can often be written declaratively (translating into (pure) Horn clauses).

For more comparison of the new Slot Grammar with related systems and with the earlier Slot Grammar system, see [McCord 1989b].

Slot Grammars for English, German, and Danish are being written respectively by the author, Ulrike Schwall, and Arendse Bernth, and are being used in versions of LMT with these source languages.

The output of Slot Grammar analysis is a syntax tree that shows both surface structure and deep grammatical relations. Such trees are the input to transfer in LMT. However, these trees are also quite amenable to deeper semantic analysis, partly because they already contain the predicate-argument structure for open-class words. In fact, there is a semantic interpreter which operates on the syntax trees and produces logical forms in the semantic representation language LFL [McCord 1987]. This in turn is input to the discourse understanding system LODUS of Bernth [1989, 1990].

The Slot Grammar output also seems quite amenable to the formulation of algorithms for syntactic constraints on anaphora [Lappin and McCord 1990]. Lappin is also working with the author on VP anaphora in the framework of Slot Grammar.

Section 2 of the paper describes what is accomplished with Slot Grammar analysis, in terms of its input and output. Section 3 discusses the lexical analysis phase of Slot Grammar analysis, which precedes syntactic analysis in a first pass. Section 4 describes the basic ingredients (rule types) of the Slot Grammar of a specific language. Sections 5, 6 and 7 are devoted respectively to extraposition, coordination, and implicit subjects (which are treated largely in the shell). The main algorithm for syntactic analysis is sketched in Section 8. Finally, Section 9 contains a description of parse evaluation and pruning.
2 Input and output of Slot Grammar analysis

Input consists of unrestricted text. The input is segmented by a "sentence" separator, and these segments are analyzed one at a time. The segments need not be complete sentences. (In the following, the input segments will usually be referred to as "sentences" for convenience, however.)

The output of Slot Grammar analysis of a sentence is a parse tree which shows both surface structure and deep grammatical relations, including remote dependencies, in a single structure. An example is shown in Figure 1.

![Parse tree example](image)

The parse tree is a kind of dependency tree; every node has a head word. In the display, there is only one line per node, showing (1) the tree connection lines, (2) the slot filled by the node (phrase), (3) the head word sense predication, i.e., a sense of the head word, together with its arguments, and (4) the feature structure of the node. In Figure 1, the feature structure have been abbreviated, by a display option, to their principal functors (normally the part of speech of the head word).

The first argument of the word sense predication is the marker variable for the node. For verbs, this is like an event or state variable; for nouns it represents the main entity referred to by the noun. The other arguments correspond (in the main) to complement slots associated with the head word. For example, the second argument of a verb sense predication corresponds to the verb's (logical) subject. These complement argument variables are unified, through slot filling, with the marker variables of the filler phrases. Note in Figure 1 that Who, with marker x2, is

---

2 There is a partial treatment of text-formatting tags for SCRIPT/GML which allows LMT to translate (some) SCRIPT source text for the source language into SCRIPT source for the target language.
shown as the object of both find and with, and Bob, with marker x6, is shown as the subject of want, find, and talk.

Internally, a syntax tree (node) is represented by a Prolog term

\[
\text{phrase}(X,H,\text{Sense},\text{Features},\text{SlotFrame},\text{Ext},\text{Mods})
\]

where the components are as follows: (1) \(X\) is the marker variable of the phrase. (2) \(H\) is an integer representing the position of the head word of the phrase. (3) \(\text{Sense}\) is the word sense of the head word. (4) \(\text{Features}\) is the feature structure of the head word \textit{and} the phrase. It is a logic term (not an attribute-value list), which is generally rather sparse in information, showing mainly the part of speech and inflectional features of the head word. (5) \(\text{SlotFrame}\) is the list of complement slots, each slot being in the internal form \(\text{slot}(\text{Slot},\text{Ob},X)\), where \(\text{Slot}\) is the slot name, \(\text{Ob}\) shows whether it is an obligatory form of \(\text{Slot}\), and \(X\) is the \textit{slot marker}. The slot marker is unified (essentially) with the marker of the filler phrase when the slot is filled, even remotely, as in left movement or coordination. (6) \(\text{Ext}\) is the list of slots that have been \textit{extraposed} or \textit{raised} to the level of the current phrase. (7) The last component \(\text{Mods}\) represents the modifiers (daughters) of the phrase, and is of the form \(\text{mods}(\text{L Mods},\text{R Mods})\) where \(\text{L Mods}\) and \(\text{R Mods}\) are the lists of left modifiers and right modifiers, respectively. Each member of a modifier list is of the form \(\text{Slot:\ Phrase}\) where \(\text{Slot}\) is a slot and \(\text{Phrase}\) is a phrase which fills \(\text{Slot}\). Modifier lists reflect surface order, and a given slot may appear more than once (if it is an adjunct). Thus modifier lists are not attribute-value lists.

Parse space pruning can be turned on or off dynamically by the user. When it is off, all (final) analyses are shown, ranked best-first by the parse evaluator. However, for long sentences the system may have a space overflow because of the large number of partial analyses in parsing (when pruning is off). When pruning is on, the final number of parses is very small — averaging about 1.5 per sentence. In spite of the drastic action of pruning away partial results during parsing, the system seems to do well at producing a correct result among the final results, modulo some reattachments of phrases such as PPs.

The parser always produces an output tree even when the input is not grammatical (for the given grammar). In such cases, the output is a "fitted" parse roughly similar to those produced by the PLNLP system [Jensen and Heidorn 1982]. LMT can use these fitted parses to produce a translation (sometimes good, sometimes bad!) for non-grammatical inputs.

There is a fairly complete system for handling files of sentences in either batch or interactive mode.
3 The lexical analysis phase

After sentence separation and tokenization have operated, the lexical analyzer operates on the words (tokens) of the sentence to produce word frame clauses which will be used by the syntactic analyzer (the main part of the parser).

A word frame clause represents a lexical analysis of a particular word in the sentence, specifying a sense, a feature structure (showing part of speech and inflection mainly), and a slot frame. Of course a word can have several analyses, so there may be several word frame clauses associated with one word. However, different occurrences of the same word may have different collections of analyses, because there are heuristics that remove some possible analyses, based on the local lexical context. For this reason, a word (token) is referred to in a word frame clause by its numerical position in the sentence. A word frame clause is then a unit (unconditional) clause of the form

\[ \text{wframe}(\text{WordNo}, \text{Sense}, \text{Features}, \text{SlotFrame}). \]

The four arguments of \text{wframe} are essentially the same as the second through fifth arguments of a phrase structure, as described in the preceding section. In fact, the bottom-up parser begins its work by constructing one-word phrase structures from \text{wframe} clauses, in the obvious way — namely by adding empty lists of modifiers and extraposed slots.

As an example of a word frame, one \text{wframe} clause for the word \text{given} in the sentence \text{Alice has given Bob a book} might be:

\[ \text{wframe}(3, s(\text{give}, 1), \text{verb(pastparta}), \] 
\[ \text{slot(subj(n),op,X}). \]
\[ \text{slot(obj,op,Y}). \]
\[ \text{slot(iobj,op,Z).nil}). \]

The sense name argument is generated automatically by the system. The feature structure argument shows that the word is the active past participle form of a verb.

Other word frame clauses could be generated for the word \text{given}. For example, for the sentence \text{The book was given to Bob}, a passive past participle frame would be appropriate:

\[ \text{wframe}(4, s(\text{give}, 1), \text{verb(pastpart}), \] 
\[ \text{slot(agent,op,Y}). \]
\[ \text{slot(subj(n),op,X}). \]
\[ \text{slot(iobj,op,Z).nil}). \]

Lexical analysis actually involves two passes over the words of the sentence: (1) morpholexical analysis and (2) lexical filtering. The first pass produces a list of word frames for each (derived or inflected) word in the sentence and the second pass erases (filters out) some of these analyses, based on lexical context.

The first pass itself involves two steps for each word. In the first step, morphological analysis (affix stripping and consultation of tables of irregular forms) is done in conjunction with look
up in lexicons stored in *external forms*, to be discussed below. This step produces derived analyses which are still basically in external form. Then a system for *lexical compiling* produces the wframae clauses (internal form) associated with these derived analyses. This system for morpholexical analysis is much as is described in preceding papers [McCord and Wolff 1988, McCord 1989a]. Much of the code for morpholexical analysis is in the shell.

In the lexical filtering pass, multiword lexical entries are used to replace sequences of adjacent words by single, conglomerate “super words” (which receive a single word number in word frame clauses). In addition, *lexical erasure rules* are applied to erase some word frame analyses based on surrounding analyses (useful for source languages like English with lots of part-of-speech ambiguity). For example, an active past participle frame, like the first one for *given* above, might be erased if there is no occurrence of a perfect *have* in the sentence. Lexical erasure rules are included in the Slot Grammar XSG of a specific language. There is a convenient rule formalism for them, and a rule compiler in the shell, but they will not be described further here due to lack of space.

**Lexicons**

There is a *standard external form* for lexicons useable by Slot Grammars. In this format, a lexical entry for a word *Word* is represented by a clause

\[
\text{Word} < \text{Analysis}
\]

where the *Analysis* shows word frame information in a convenient, abbreviated, external form. For example, an entry for *give* might be simply

\[
give < v(\text{obj.iobj}).
\]

The subject slot is omitted here because by default it is added by the lexical compiler. This external lexical form is essentially the same as is described in earlier papers on LMT.3

Two lexicons used by ESG (the English Slot Grammar) are the following. There is a small lexicon (with about 3000 entries) for some of the most common English words, stored in standard external form. Words not found in this lexicon are looked up in the UDICT lexicon [Byrd 1983, Klavans and Wacholder 1989], which has over 60,000 lemmas, in conjunction with a heuristic interface that produces entries in standard external form. The look up can use either UDICT in DAM form (with UDICT morphology) or a Prolog-based look up of the UDICT data stored in Prolog-readable files (together with the Slot Grammar shell morphology). When the latter method is used, the complete lexical analysis phase can be performed at a rate of about 10 milliseconds per word (or about 15 milliseconds for words coming only from the UDICT disk files).

---

3 There has been a revision and improvement of the form of lexical *transfer* information in LMT, with more options in the overall organization and storage of the MT lexicons; but the form of source lexicons is essentially unchanged.
Work is being done to get more accurate and complete information from on-line dictionaries. Jacques Robin is working on extracting Slot Grammar-style information from Longman's. Also the work of Mary Neff on lexical data bases [Neff and Boguraev 1989] and the use of an LDB for the Collins bilingual lexicons for LMT [Neff and McCord 1990] may supply information useful for source analysis as well as transfer.

4 The ingredients of a Slot Grammar

In this section we discuss the types of rules (and data) that appear in the Slot Grammar XSG of a specific language, i.e. what's not in the shell. There are four major types of rules:

- A declaration of *adjunct slots* for each part of speech.
- *Slot filler rules.*
- *Slot ordering rules.*
- *Obligatory slot rules.*

In addition, there are minor types of rules (often expressed as unit clauses) dealing with language-specific aspects of (“parameters” for) phenomena treated mainly in the shell, namely extraposition, coordination, punctuation, and parse evaluation.

For most of these rule types, there is a special formalism that allows convenient reference to the parts of the phrases dealt with by the rule, and there is a rule compiler in the shell which converts the rules to Prolog clauses.

The four subsections below describe the four major rule types. Some of the minor rule types are discussed in subsequent sections.

4.1 Adjunct slots

The list of complement slots associated with a word is idiosyncratic to the word and comes from the lexical entry for the word. On the other hand, a word may have associated *adjunct* slots, and these depend only on the lexical category (part of speech) of the word. The declaration of available adjunct slots for a part of speech POS is given in the grammar by a unit clause of the form

```
category(POS,Adjuncts).
```

For example, the declaration in ESG for verbs is currently
Adjunct slots differ from complement slots in that adjuncts are always optional and in general may be filled any number of times (there may be several modifiers in a given phrase, not necessarily adjacent, which fill the same adjunct slot).

4.2 Slot filler rules

The core of a Slot Grammar is its set of slot filler rules. Each slot known to the grammar (arising from lexical entries in the case of a complement slot, or declared for a part of speech in the case of an adjunct slot) will have one or more filler rules. The basic step in parsing is to choose a slot \textit{Slot} associated with (the head word of) a phrase \textit{H} and to see whether an adjacent phrase \textit{M} (on the left or right) can fill \textit{Slot} by satisfying a filler rule for \textit{Slot}. If so, \textit{M} is added as a modifier of \textit{H}, and the process is continued with the larger phrase.

A filler rule for \textit{Slot} is of the form

\textit{Slot} \mapsto \textit{Body}.

The \textit{Slot} on the left-hand side can be of three types: (1) a complement slot, (2) an adjunct slot, or (3) an extraposed slot, of the form \texttt{ext(Slot0, Level)}, where \textit{Slot0} is a complement slot and \textit{Level} is \texttt{ext} or \texttt{norm}. (The significance of \textit{Level} will be explained below.) The filler rule is correspondingly called a \textit{complement filler rule}, an \textit{adjunct filler rule}, or an \textit{extraposed filler rule}. The first two types of rules are called \textit{normal} filler rules. Normal filler rules are described in the current section, and extraposed filler rules are described in Section 5. However, the form of the \textit{Body} is the same for all three kinds of rules, and the rule compiler treats the body in the same way.

The \textit{Body} has the same overall form as the right-hand side of a Prolog clause, \textit{i.e.}, a combination with and's (\&) and or's (\lor) of goal predications; but there are \textit{special goals} that can refer to parts of the filler phrase or the head phrase. Special goals, known to the filler rule compiler, are compiled in a special way, and all other goals are compiled as themselves.

As an example, a filler rule for the object and indirect object slots might be

\begin{align*}
\textit{obj} & \mapsto \texttt{f(noun(*,acc.*,*))}.
\textit{iobj} & \mapsto \texttt{f(noun(*,dat.*,*))}.
\textit{iobj} & \mapsto \texttt{f(prep(to,*,*))}.
\end{align*}
Here \( f(\text{Feas}) \) is a special goal that requires that the filler phrase has feature structure \( \text{Feas} \). As another example, a rule for the subject slot

\[
\text{subj} \rightarrow f(\text{noun}(*,\text{nom.}\text{Num},*)) \&
\quad h\text{f}(\text{verb}(\text{fin}(*,\text{Num},*,*))).
\]

\( h\text{f}(\text{Feas}) \) requires that the head phrase (or higher phrase) has feature structure \( \text{Feas} \).

Special goals are two types: selector goals and feature-changing goals. Selector goals arise from the following scheme. There are several selector predicates for accessing parts of a phrase data structure. For example, \( f(\text{P},\text{Feas}) \) says that phrase \( \text{P} \) has feature structure \( \text{Feas} \). This selector is of course defined simply by the unit clause:

\[
f(\text{phrase}(*,*,*,\text{Feas},*,*,*),\text{Feas}).
\]

A more complex example is \( \text{eslot}(\text{P},\text{Slot},x) \), which says that phrase \( \text{P} \) has an empty (unfilled) complement slot \( \text{Slot} \) with marker \( x \). This is defined by

\[
\text{eslot}(\text{phrase}(*,*,*,\text{Frame},*,*),\text{Slot},\text{x}) \rightarrow
\quad \text{member}(\text{slot}(\text{Slot},\text{x},\text{Frame}) \& \text{var}(\text{x}).
\]

(This can be used both for generating empty slots and for testing whether a given slot is empty.) For all of these selector predicates, the first argument is a phrase. In filler rules, the filler phrase and head phrase are implicit. So the simple scheme is that for each selector predicate

\[
\text{pred}(\text{P},x_1,\ldots,x_n)
\]

where \( \text{P} \) is the phrase argument, we get two special goals

\[
\text{pred}(x_1,\ldots,x_n)
\quad \text{hpred}(x_1,\ldots,x_n)
\]

where the phrase argument is understood to be the filler phrase in the first case and the head phrase in the second case. In this way, the special goals \( f(\text{Feas}) \) and \( h\text{f}(\text{Feas}) \) arise from the selector predicate \( f(\text{P},\text{Feas}) \).

Selector predicates can also be used in their full forms in filler rules. One of the modifiers of the filler or head phrase could be selected (by use of a special goal), and then a full-form selector could be applied explicitly to this modifier phrase. However, it is not common to have to do this. Feature-changing goals will not be dealt with in the current paper; for details, see [McCord 1989b]. Also, see this reference for a description of the compilation of filler rules.
4.3 Slot ordering rules

There are two kinds of slot ordering rules: (1) head/slot ordering rules, expressing ordering of slots with respect to the head word, and (2) slot/slot ordering rules, expressing ordering of slots with respect to other slots. Although these rules are stated in terms of slots, the interpretation is of course in terms of fillers of slots and their left-to-right surface order. Every time a normal (non-extraposed) slot filler rule is applied, the parser checks that the position of the new filler is allowed by the ordering rules.

In some cases, ordering of slots is unconditional, but in other cases it can depend on characteristics of filler phrases or the head phrase. For example, in English the subj slot can be on the right of the head verb in yes/no questions when the head verb is an auxiliary (auxiliaries are treated as higher verbs in ESG).

Head/slot ordering rules are of either of the forms:

\[ \text{lslot(Slot)} \rightarrow \text{Body.} \]
\[ \text{rslot(Slot)} \rightarrow \text{Body.} \]

The antecedent side Body (and the arrow) may be omitted. These rules say respectively that Slot is a left slot (or right slot), i.e., that its filler is a left (or right) modifier of the head word, under the conditions of Body. The Body has the same form as the body of a filler rule, except that feature-changing goals are not allowed. Thus, selector goals can refer to any parts of the filler phrase or the head phrase.

As an example, constraints on the subj slot in English can be expressed by the rules

\[ \text{lslot(subj)} \rightarrow \text{hf(verb(fin(*,*,*,*))}. \]
\[ \text{rslot(subj)} \rightarrow \text{hf(verb(fin(*,*,ind:q:*))) & hsense(Head) & finaux(Head).} \]
\[ \text{finaux(be).} \]
\[ \text{finaux(have_perf).} \]
\[ \text{finaux(will).} \]
\[ \ldots \]

The first rule says that subj can be a left slot for a finite verb, and the second rule says that subj can be a right slot in a question sentence (feature q) if the verb is a finite auxiliary. (Absence of the feature q signals a declarative sentence.)

A slot/slot ordering rule is of the form:

\[ \text{LSlot}<<\text{RSlot} \rightarrow \text{Body} \]

where the Body may be omitted. This means that every filler for LSlot must precede every different filler for RSlot under the conditions of Body. The point in saying "different" here is that LSlot can equal RSlot, and then the ordering constraint says that LSlot can have no more
than one filler. (Two different fillers would have to precede each other.) For example, we could have the following rule for the noun determiner (adjunct) slot \text{nndet}:

\[ \text{nndet} \ll \text{All}. \]

Here \text{All} is just a Prolog variable, so this means that \text{nndet} precedes every slot, including itself. Therefore, there is at most one determiner, and this must precede every other noun modifier.

The body of a slot-slot ordering rule can contain selector goals allowing reference to the phrases involved, namely the filler of \text{Lslot}, the filler of \text{Rslot}, and the head phrase. Correspondingly, the selector goals are of the form \text{lpred}, \text{rpred}, and \text{hpred}, where \text{pred} is a selector predicate. For example, \text{lf(Feas)} selects the feature structure of the filler of \text{Lslot}, and \text{hframe(Frame)} selects the slot frame of the head phrase.

Examples of conditional slot-slot ordering rules are the following ones, expressing relative ordering of the direct and indirect objects of verbs:

\[ \text{iobj} \ll \text{obj} \leftarrow \text{lf(noun(*,*,*))}. \]
\[ \text{obj} \ll \text{iobj} \leftarrow \text{rf(prep(*,*,*))}. \]

These say that the indirect object precedes or follows the direct object according as the indirect object is a noun phrase or a prepositional phrase. (Cases of heavy direct objects that can follow a PP indirect object are ignored here.)

### 4.4 Obligatory slot rules

A complement slot may be \textit{optional} or \textit{obligatory}. (Adjuncts are always considered optional.) For a phrase to become a modifier of another phrase, all of the obligatory slots of the modifier phrase (after possible extraposition of one of its slots, as described in Section 5) must be filled. Also, an allowable top-level analysis must have all of its obligatory slots filled. An optional slot need not get a filler. As an example, the verb \textit{eat} might have an optional object slot, but the verb \textit{put} might have an obligatory object slot.

There can be optional and obligatory versions of the same slot. In fact, in internal form a slot \text{slot(Slot,ob,X)} is obligatory when \text{ob=ob}, and optional when \text{ob=op}. There is a convention (known by the lexical compiler) for slots shown in the external lexicon, that an obligatory form of a slot is obtained from the ordinary name by adding the character 1. Thus \text{obj1} is the obligatory form of \text{obj}, and its internal form is \text{slot(obj,ob,X)}. In the grammar, one just refers to \text{obj}.

Thus slots in particular slot frames of words may be obligatory because the words' lexical entries specify this, but it is also possible to write general rules in the grammar that require slots to be obligatory, either absolutely or conditionally. An obligatory slot rule is of one of the forms:
obl(Slot) ← Body.
obl(Slot,Slotl) ← Body.

As before, the Body and arrow may be omitted. The first type of rule rule says that Slot is obligatory under the conditions of Body. The second type of rule, a relative rule, says that Slot is obligatory if Slot1 is filled and the conditions of Body hold. The body has a form similar to that of the body in our previous rules, allowing reference to relevant phrase characteristics through selector goals. In the first type of rule, one may refer to the head phrase through goals of the form hpred. Such goals are allowed in the second type of rule, as well as goals (of the form pred) which refer to the filler of Slot1.

Examples of simple obligatory slot rules for English are the following:

obl(subj) ← hf(verb(fin(*,*,*,*))).
obl(obj,iobj) ← f(noun(*,*,*)).
obl(obj,prep).

The first rule says that the subject is obligatory in a finite verb phrase. The second rule says that the direct object is obligatory if the indirect object is filled by a noun phrase. The third rule says that the object of a preposition is obligatory unconditionally.4

5 Extrapolation

Recall from Section 2 that in a phrase structure

\[ \text{phrase}(X,H,\text{Sense},\text{Features},\text{SlotFrame},\text{Ext},\text{Mods}), \]

the argument \text{Ext} is used to hold \textit{extraposed} slots, \textit{i.e.}, slots that can be filled by left-extraposed phrases like \textit{who} in \textit{Who did Alice try to find}. The list \text{Ext} consists of slots in internal form, and it can contain more than one element, as we will see below.

The shell takes care of several of the problems with extrapolation, but there are two ingredients in the grammar itself that deal with it:

1. a declaration of \textit{extraposer} slots, and
2. extrapolated filler rules.

Extraposer slots are slots that allow extrapolation \textit{out of} their fillers. Consider the above example

\[ \text{obl}(\text{subj}) \rightarrow \text{hf}(\text{verb(fin(*,*,*,*))).} \]

\[ \text{obl(obj,iobj)} \rightarrow \text{f(noun(*,*,*)}. \]

\[ \text{obl(obj,prep)}. \]

As indicated above, obligatory slots need not be filled immediately on the level of their head words, but may be extrapolated and filled on a higher level. Thus sentences like \textit{Which chair was John sitting in?} are allowed even though \text{obj,prep} is obligatory.
Who did Alice try to find?

Here the verb phrase to find fills a slot obj(\text{inf}) for try. The obj(\text{inf}) slot is declared to be an extraposer by a clause

\text{extraposer(obj(\text{inf}))}.

in the grammar. This tells the parser that the unfilled obj slot for find can be extraposed to the level of try, becoming the sole member of the Ext argument of the phrase structure of try. It also remains in the frame of find. (The restrictions on what slots (like obj) can be extraposed will be described below.) Next, the phrase try to find fills a slot auxcmp(\text{inf(bare)}) for did. This slot is also declared an extraposer in the grammar. The still-unfilled obj slot is extraposed still higher, to the level of did. Then finally an extraposed filler rule allows who to fill this slot. The link of who to the object of find is shown in the phrase structure, because the marker of the obj slot down in the frame for find is unified with the marker for who.

The slots that can always be extraposed through a slot declared as an extraposer are the slots that are basically objects: obj, pobj(Prep), and objprep (the third one is the object of a preposition). There are others, but we will not list them all.

What about the extraposition of subjects? We need to capture the fact that we can say

\textit{Who did you say found the book?}

but not

\textit{Who did you say that found the book?}

If a slot S is declared a strongextraposer, then the subj slot (as well as the object-type slots named above) can be extraposed through S. In addition, the subj slot can be extraposed through an extraposer if it has already been extraposed (is in the Ext argument as opposed to the Frame argument of the filler phrase).

So let us see how this allows us to handle the preceding two examples. In the sentence, \textit{Who did you say found the book}, the verb phrase found the book fills a slot obj(\text{fin}) of say, and there is a declaration

\text{strongextraposer(obj(\text{fin}))}.

Hence the unfilled subj slot of found can be extraposed through obj(\text{fin}) to the level of say. When the verb phrase say found the book fills the auxcmp(\text{inf(bare)}) slot (an extraposer) of did, the subj slot of found can be extraposed, since it has already been extraposed. Then it is available to be filled by who using an extraposed filler rule.
Before looking at the second, ungrammatical example above, let us look at a similar example that is grammatical:

*What did you say that Alice found?*

Here *that* is viewed as a special type of subordinating conjunction with the feature structure $f_{subconj}$. It has a single complement slot, $fv_{comp}$, which can be filled by a finite verb phrase. Furthermore, $fv_{comp}$ is declared an *extraposer*, so that when the verb phrase *Alice found* fills $fv_{comp}$, the *obj* slot can be extraposed to the level of *that*. Now the $obj(fin)$ slot of *say* has a filler rule allowing it to be filled by a phrase with feature structure $f_{subconj}$, so that it can be filled by *that Alice found*, with further extraposition of *obj*, since $obj(fin)$ is a (strong) extraposer.

Now let us return to the example

*Who did you say that found the book?*

The point is simply that the complement slot $fv_{comp}$ of *that* is not declared a *strong* extraposer. Hence there is no chance for the *subj* slot of *found* to be extraposed. In fact, the parser does not allow *found the book* to fill the $fv_{comp}$ slot, because there is a slot (its *subj*) which (1) cannot be extraposed and (2) is obligatory. (The subject slot is declared obligatory for finite verb phrases.)

It should be noted that the only types of slots that can be extraposed are complement slots; adjunct slots are not extraposed. If one wanted to handle extraposed attachments of *wh*-adverbials as in *When did you invite him to speak?* (where one reading links *when* to *speak*), then one should allow extraposition of the markers of phrases. Making the marker of an embedded verb phrase available on a higher level would allow extraposed attachment of *wh*-adverbials to the embedded phrase. Currently, however, the system does not do this; adverbials are linked only to their surface head verbs.

The system does allow extraposition of prepositional object slots ($obj_{prep}$) through adjunct slots, allowing correct parsing of examples like

*Who did you say John was sitting with?*

where *who* fills the extraposed $obj_{prep}$ of *with*, and the *with*-phrase fills an adjunct slot for *sitting*.

It was mentioned above that the extraposed slot list $Ext$ can contain more than one element. This occurs with an ambiguous sentence like

*Which horse do you want to win?*
Here we give the verb *want* the slot frame `subj.obj.comp(inf)` (with `obj` optional), where the VP *to win* fills `comp(inf)`. The ambiguity results from the fact that *which horse* can fill either the `obj` slot of *want* or the `obj` slot of *win*. (As described below in Section 7 on *Implicit subjects*, the system decides that the implicit subject of *win* is *which horse* in the first case, and *you* in the second case.) The extraposed slot list of *do* contains two slots because both the object of *win* and the object of *want* are extraposed to the level of *do*. Thus *which horse* can fill either of these two slots, and **ESG** produces two parses.

As indicated in Section 4.2, the filling of extraposed slots is handled by *extraposed filler rules*, which are of the form

```
  ext(Slot,Level) => Body.
```

The argument `Level` is `ext` if the slot is chosen from the extraposed slot list `Ext` of the head phrase, and it is `norm` if the slot is chosen from the slot frame of the head phrase. We need to allow the latter for examples like the filling of the relative pronoun *that* in *the man that John saw*.

In **ESG** there are just two extraposed filler rules. One of them fills object-type slots and `subj` by `wh-noun` phrases (including relative pronouns). The other fills the `pobj(Prep)` slot by `wh-prepositional` phrases. In both cases, the marker of the `wh`-element is saved in the feature structure of the head verb phrase, so that when this verb phrase acts as a relative clause (filling the adjunct slot `nrel` for nouns), the `wh`-marker can be linked to the marker of the head noun.

What about *abbreviated* relative clauses, as in *the book I was trying to find*? In such cases, extraposition of slots proceeds just as for full relative clauses, so that in the example the `obj` slot of *find* is extraposed to the level of *was*. But there is no overt relative pronoun to trigger filling of this slot. Instead, the filling is triggered in the following way. There is a special adjunct slot `abbnrel` for nouns which is filled by abbreviated relative clauses. The filler rule for `abbnrel` looks for an unfilled normal or extraposed slot in the filler verb phrase that could receive a relative pronoun, and then "fills" this slot in the sense that its marker is unified (essentially) with the marker of the head noun. In addition, `abbnrel` employs some constraints on the form of the proposed abbreviated relative clause, which are useful in cutting down on spurious partial analyses in parsing.
6 Coordination

The following method for handling coordination was outlined in [McCord 1980], and was implemented (with improvements) in the recent adaptation of Slot Grammar to logic programming.

First let us discuss the types of coordinated phrases analyzed by the system, and the form of phrase analyses produced for them. The main form for such a phrase is

\[ \text{LM Preconj LC Conj RC RM} \]

where the substrings indicated are as follows. \text{Conj} is a coordinating conjunction (like \text{and} or \text{or}), or a punctuation symbol (like a comma) used in the capacity of a coordinating conjunction. \text{Preconj} is an optional \textit{preconjunct} that can accompany \text{Conj} (like \text{both} for \text{and}). \text{LC} and \text{RC} are the left and right conjuncts, respectively. Each of these conjuncts consists of a single phrase, although it may not be satisfied (it may have unfilled obligatory slots). \text{LM} and \text{RM} are the (optional) left and right common modifiers, respectively (each of these may be represented by several phrases). Some examples:

\text{The man sees and probably hears the car.}
\[
\begin{array}{cccc}
\text{LM} & \text{LC} & \text{Conj} & \text{RC} \\
------- & ----- & ----------- & -------
\end{array}
\]

\text{John sees and Mary hears the car.}
\[
\begin{array}{cccc}
\text{LC} & \text{Conj} & \text{RC} & \text{RM} \\
------- & ------- & ----- & -------
\end{array}
\]

The specification of coordinating conjunctions and their associated preconjuncts is of course language-specific, and is given in the grammar\footnote{This information is put in the grammar instead of the lexicon because of the special and limited nature of these words.} by clauses like:

\text{conj(\text{and}).}
\text{preconj(\text{and, both, bothand}).}

We separate \text{preconj} from \text{conj} because in some languages a given conjunction can have more than one preconjunct.

The phrase structure produced by the system for a coordinated phrase \text{LM Preconj LC Conj RC RM} is as follows.

(1) The head (sense) is basically the conjunction \text{Conj}, but is actually the following compound term:

\text{coord(Conj1, LHead, RHead}).

If a preconjunct \text{Preconj} is present, then \text{Conj1} is defined by
preconj(Conj,Preconj,Conjl).

Otherwise \( \text{conj}_1 = \text{Conj} \). The second and third arguments of \text{coord} are the heads of the conjuncts \( \text{LC} \) and \( \text{RC} \).

(2) The feature structure \( \text{Feas} \) of the coordinated phrase is manufactured from the feature structures \( \text{LFeas} \) and \( \text{RFeas} \) of the conjuncts by a procedure

\[
\text{coordfeas}(\text{Conj}, \text{LFeas}, \text{RFeas}, \text{Feas})
\]

specified in the grammar. (This constitutes the main language-specific information regarding coordination.) Typically, \( \text{LFeas} \) and \( \text{RFeas} \) will be required to be rather similar, and usually to be of the same part of speech, with the result \( \text{Feas} \) sharing at least this part of speech. For example, in ESG a finite verb phrase will coordinate only with another finite verb phrase (although their tenses, etc., are allowed to differ), and the result is a finite verb phrase. Thus, so far as features go, coordinating conjunctions are "parasitic" on their conjuncts.

(3) The slot frame of the coordinated phrase is manufactured by a procedure \text{coordframe} in the parser, using both the slot frames and the extraposed slots of the conjunct phrases. The idea is to "factor out" common slots in the conjuncts, raising them up to the level of the new coordinated phrase, making them available for filling on this level. In particular, common modifiers (in \( \text{LM} \) and \( \text{RM} \)) may fill slots in this factored-out frame.\(^6\) In the last example sentence above, \text{the car} fills a slot \text{obj} factored out from the frames of \text{sees} and \text{hears}. The procedure \text{coordframe} is the core of the treatment of coordination in the parser, and will be described in more detail below.

(4) The extraposed slot list \( \text{Ext} \) of the coordinated phrase is empty. Extraposed slot filling \text{can} be done on this level (or higher), but it is just done using slots stored in the frame. (There is no need to keep extraposed slots separate from normal slots at this point, because the "real" frame positions of slots are stored below in non-conjoined phrases.)

(5) The left modifiers of the coordinated phrase are as follows. The rightmost one is of the form \( \text{lconjunct:LP} \) where \( \text{LP} \) is the phrase representing the left conjunct \( \text{LC} \). The initial left modifiers (if any) represent the common left modifiers \( \text{LM} \). The form of the right modifiers is symmetric, with \( \text{RC} \) filling the slot \( \text{rconjunct} \).

Now let us look in more detail at the work of \text{coordframe}. The first thing to note is that in the process of factoring out slots, we must be ready to consider (unfilled) slots \( \text{LSlot} \) and \( \text{RSlot} \) of the conjuncts to produce a common slot even when \( \text{LSlot} \) and \( \text{RSlot} \) are not exactly the same. For instance, the slot \text{obj(fin)} mentioned in the preceding section can be filled not only by finite verb phrases, but also by noun phrases (it is a complement slot of \text{believe}, for example). This could be considered compatible with an \text{obj} slot, for the purposes of factoring out, and the result would be the "greatest common denominator" slot \text{obj}. Therefore we use a procedure

\(^6\) Such common modifiers will of course be complement modifiers, but common modifiers could also fill adjunct slots selected for the coordinated phrase solely on the basis of its feature structure.
coordslot(LSlot, RSlot, Slot)

which can produce a “g.c.d” slot Slot from LSlot and RSlot.

The next observation is that the results of extraposition are of direct value for coordination; extraposed slots can be used in the process of factoring out, becoming filled (in common slot form) by a phrase that is not fronted. Consider an example of Woods [1973]:

John drove his car through and completely demolished a plate glass window.

ESG handles this as follows. In the left conjunct drove his car through, the objprep slot of through is extraposed to the level of drove. This slot can be paired by coordslot with the obj slot in the right conjunct completely demolished, producing a common slot obj, which gets filled by a plate glass window. The subj slots (non-extraposed) of the two conjuncts are factored out and filled by John. The parse produced by ESG is shown in Figure 2.

Note that the marker x6 for John is also the subject variable for drive and for demolish, and the marker x7 for window is the object variable for through and for demolish.

One can make up examples involving several levels of extraposition, which are handled in a similar way by ESG.

Because of this double use of extraposition, the extraposed slot lists of the two conjuncts are pooled together with (concatenated onto) their normal slot frames for the work of coordframe. Let us call these combined frames for the left and right conjuncts LFrame and RFrame, respectively.

Then coordframe succeeds in producing a factored-out frame Frame from LFrame and RFrame, under the following conditions:
1. Whenever an unfilled slot \textit{LSlot} of \textit{LFrame} can be paired by \textit{coordslot} with an unfilled slot \textit{RSlot} of \textit{RFrame}, their markers are unified, and their \textit{Ob} components are suitably combined. The resulting (factored out) slot is made a member of \textit{Frame}.

2. Any unfilled slot of \textit{LFrame} or \textit{RFrame} that is not paired as in (1) must be optional (where we consider verb subjects obligatory).

The coordination system handles a bit more than the pattern

\[ \text{LM Preconj LC Conj RC RN} \]

specified above, namely the following three variations.

(a) If \textit{Conj} is a lexical coordinating conjunction (not punctuation), then it may be preceded by a comma or semicolon.

(b) An adverb is allowed between \textit{Conj} and \textit{RC} (and it is made the leftmost right modifier of the coordinated phrase). Such adverbs are allowed in coordination of phrases of any category, even ones not normally modified by adverbs. (This is due of course to the fact that coordination of any category can abbreviate sentential coordination.)

(c) The construction

\[ \text{Conj RC} \]

(where \textit{Conj} is lexical) is considered a valid phrase by itself. This is useful not only because such phrases do occur, but also because in full coordinated phrases the \textit{Conj RC} portion can be enclosed in \textit{brackets} (parentheses and other paired symbols), as in

\textit{The director will decide on (or soon consider) the case.}

There are general rules in the system for dealing with bracketing, and these apply to phrases, so that examples like the preceding are handled if \textit{Conj RC} is made a phrase. Therefore, even for building a full coordinated phrase, a preliminary step in the parser is to make the \textit{Conj RC} portion a phrase, giving it a chance to be bracketed. Then this structure is used as input to the building of the (more symmetric) structure for the full coordinated phrase described above.
It is useful to identify implicit subjects of non-finite verb phrases when possible. This is done (in some cases) by the Slot Grammar shell for non-finite verb phrase complements of verbs, in a pass performed after complete phrase analyses of the input string have been obtained. The method used does not always give correct results, but seems to be a useful approximation.  

Suppose we are given a verb phrase vP1, and a non-finite verb phrase complement vP2 of vP1, where the grammatical subject slot s2 of vP2 is unfilled. We “identify” s2 by unifying its marker with the marker of a suitably chosen slot sl in the frame of vP1. In many cases, s1 will already be filled, so that s2 will be related then to an overt filler.

So the problem is in choosing the higher slot sl. The full statement of the rule we follow is given below, but the basic idea (to which there are exceptions) is the following. If an object-type slot is present and filled in the frame of vP1, then s2 is taken to be this slot; otherwise s1 is the subj slot of vP1. The object-type slots are obj, pobj (Prep), and iobj.

As an example, suppose that want has the slot frame

```
subj. obj. comp (inf)
```

(where obj is an optional slot). Then in John wants to leave, the subject of leave would be linked to the subject of wants (because obj is not filled). But in John wants Bill to leave, it is linked to the obj (Bill) of wants.

There are exceptions to the basic rule stated above, due just to the nature of the head verb of vP1. In both of the sentences John promised to leave and John promised Bill to leave, the lower subject is linked to the higher subject. Such verbs are specially marked ss (“subject-to-subject”) in the lexicon. Some verbs, like advise, refuse to accept a link to their subjects (in active form), and these are specially marked so (“subject-to-object”) in the lexicon. In general, verbs (like expect) may have neither of the features ss, so.

The complete statement of our implicit subject rule is as follows. (Assume the same meanings for vP1, vP2, and s2 as above.)

1. If vP1 is passive (marked pastpart), then s2 is linked to the grammatical subject of vP1;
2. otherwise, if an object-type slot s1 is present and filled in the frame of vP1 and the head of vP1 is not marked ss, then s2 is linked to s1;
3. otherwise, if the head of vP1 is not marked so, then s2 is linked to the subject of vP1;
4. otherwise, no link is made.

There are still difficulties with this rule, because the correct linking can also depend on the nature of the embedded verb phrase, and on contextual conditions. So we have only an approximation.

---

7 A similar method was used in the logic grammar described in [McCord 1982].

8 I wish to thank Hubert Lehmann and Brigitte Barnett for a useful conversation in this regard.
To see why implicit subject identification is done after the final parse is obtained, consider the example discussed in Section 5 above: *Which horse do you want to win?* The point is simply that we do not know whether the object slot of *want* is filled until we have the complete parse, where we have decided whether *which horse* fills this slot or the object slot of *win*.

An alternative approach (followed in an earlier version of the system) allows the determination of implicit subjects during parsing, but at the expense of using more word frames for verbs like *want*.

8 The basic parsing algorithm

The overall idea of the parser is the following. After lexical analysis of the input word string, resulting in *word analyses* (as described in Section 3), the words are processed left-to-right, building up *phrase analyses* of substrings. All satisfied *phrase analyses* of the entire input string are considered valid parses. (A phrase is satisfied if all its obligatory slots are filled.)

More specifically: In the left-to-right processing, when a new word (or multiword) is encountered, we look at each word frame for it and construct the corresponding initial phrase having no modifiers and an empty extraposition (*ext*) argument. This phrase is considered a new *result*. A *result* records not only a *phrase* structure, but also the left and right boundaries of the phrase in the word string, a term representing a parse evaluation (to be described in Section 9) and a *state*, representing what kind of modifiers (left, right, or extraposed) the phrase has received. Specifically, *state* will be 0 if the phrase has no right or extraposed modifiers, but may have left modifiers; *state* will be 1 if the phrase has some right modifiers, but no extraposed modifiers; and *state* will be 2 if the phrase has some extraposed modifiers. Of course *state* is set to 0 when an initial result is created from a word frame.

Whenever a new result *R* is obtained, it is *inserted* into the store of previous results by a procedure *insert*. Basically, this means the following. Given *R*, *insert* selects a previous result *L* whose right boundary coincides with the left boundary of *R*, and tries to *combine* *L* and *R*. *Combining* will be explained in the next paragraph. If the combination succeeds, then a new *result* is obtained, and this is recursively *inserted*. Backtracking in the actions just described is forced by a *fail* goal in the definition of *insert*. Thus every adjacent result *L* will be tried, and all ways of combining *L* and *R* will be tried.

Two adjacent results can be *combined* in four different ways: (1) The phrase on the left can become a *normal slot filler* of the one on the right; (2) *vice versa*; (3) the phrase on the left can become an *extraposed slot filler* of the one on the right; or (4) the two phrases can be combined in coordination.

Coordination has been discussed above, so by now our exposition of the basic parsing algorithm is reduced to an explanation of the procedure for slot filling. The procedure involved is

\[
\text{fillslot}(\text{Side, Level, MPhrase, HPhrase, Phrase})
\]
(described in slightly simplified form). This lets $\text{MPhrase}$ fill a slot in $\text{HPhrase}$ on side $\text{Side}$, with result $\text{Phrase}$. The slot filling is normal or extraposed according as $\text{Level}$ is norm or ext.

There are three rules for $\text{fillslot}$, corresponding to the three types of slot filler rules invoked: (a) normal filler rules for complement slots, (b) (normal) filler rules for adjunct slots, and (c) extraposed filler rules (for complement slots).

For each of the three $\text{fillslot}$ rules, the basic steps involved are the following:

1. Choose an unfilled slot of the appropriate type.
2. Apply the appropriate filler rule.
3. Check ordering constraints, for normal slot filling (not done for extraposed filling).
4. Apply extraposition from the modifier phrase if possible (not done for extraposed filling).
5. Check that the modifier phrase is satisfied (modulo slots extraposed from it).

Of course the new phrase resulting from the modification is obtained by "updating" the head phrase with the new modifier, possible new extraposed slots, and a possible change in its feature structure.

Most of the operation of the six steps should be clear from the discussion in the preceding sections, but the following comments are worthwhile.

The first of the six steps adds non-determinism to $\text{fillslot}$. (In general, there can be more than one unfilled slot.)

In the actual code, steps 4 and 5 are combined, because it is more efficient to run through the modifier's slots only once. For each unfilled slot $\text{Slot}$ in the modifier's slot frame or extraposition list, if $\text{Slot}$ can be extraposed (Section 5) then this is done. Otherwise $\text{Slot}$ must not be obligatory in the modifier (Section 4.4).

Step 3 (checking ordering constraints) is performed as follows. Call the head phrase $P$, and suppose that the slot being filled by $\text{fillslot}$ is $\text{Slot}$, and its modifier $\text{MP}$ is on the right of $P$. (The treatment of a left modifier is essentially symmetric.) Then the following two checks are made, using the slot ordering rules of the grammar (in their compiled forms). (a) $\text{Slot}$ must be a right slot for $\text{MP}$ modifying $P$ (i.e. an $\text{rslot}$ rule for these items must hold). (b) It must not be the case that there is a filled slot $\text{Slot1}$ such that the slot/slot ordering relation $\text{Slot}$ << $\text{Slot1}$ holds. (Since the filler $\text{MP}$ of $\text{Slot}$ is going to be placed on the right, this would be a conflict.)

There is a slight complexity in the search for a filled slot $\text{Slot1}$ in the second check. The associated modifier could be simply among the immediate modifiers of $P$, but $P$ could be coordinated. If so, one must also look down in the conjuncts.
9 Parse evaluation and pruning

Our parse evaluator is based on a partial order

\[ R_1 \text{ betterthan } R_2 \]

defined on the parse space (the set of all results \( R \)). During parsing, if a new result \( R \) is obtained and there is a previous result that is betterthan \( R \), then \( R \) will not be considered further (will not be inserted). If \( R \) is not ruled out in this way, then any previous results that are worse than \( R \) will be deleted.

In addition, an equivalence relation is defined on the parse space, expressing broad similarity of feature structure in results. Results can be related by betterthan only when they are in the same equivalence class, so that parse pruning is done independently in each equivalence class. Within an equivalence class, betterthan is a total order, based on numerical scoring.

Currently, betterthan expresses preferences in three different dimensions:

1. close attachment,
2. preference for complements (over adjuncts), and
3. parallelism in coordination.

The component of the numerical scoring function that controls for close attachment is Heidorn's [1982] parse metric. These three aspects of preference can be at odds with one another. For example, in the sentence *John sent a file to Bill*, close attachment would favor attachment of *to Bill* to *file*, but complement preference would favor attachment to *sent* (because of the *iobj* slot for *sent*). To settle this conflict, betterthan is defined so that complement preference completely dominates close attachment (i.e., close attachment is considered only if there is no decision by complement preference).\(^9\)

The relation betterthan for results is defined in terms of the evaluation structures stored in results. We define these structures now, and a partial order, also called betterthan, on them. Then a result \( R_1 \) is taken to be betterthan a result \( R_2 \) if and only if \( R_1 \) and \( R_2 \) have the same boundaries and the evaluation structure of \( R_1 \) is betterthan that of \( R_2 \).

An evaluation structure for a result is a term of the form

\[ \text{eval}(H,F,\text{Score}) \]

---

\(^9\) The combined treatment of complement preference and close attachment given here has some similarities (in results, but not in the algorithm) with the treatment in [Wilks, Huang and Fass 1985]. The ideas were also implemented in ModL [McCord 1986, 1989a] in still another way. The lexical information necessary for setting up complement preferences in [Wilks, Huang and Fass 1985] was represented in terms of semantic cases, but it can also be represented in terms of syntactic cases (our slots) with associated semantic type requirements. Semantic type checking is not currently done systematically in the Slot Grammar system (although it could be added). Nevertheless, the purely syntactic preference methods described in this section often produce parses suitable for use in MT.
where the three components are as follows. (1) $H$ is the word number of the head word of the result (phrase). (2) $F$, the basic feature of the phrase, is obtained from the feature structure $\text{Feas}$ of the phrase as follows: In most cases, $F$ is the principal functor of $\text{Feas}$ (i.e., the part of speech), but if $\text{Feas}$ is verb(Infl) then $F$ is the principal functor of Infl. (3) $\text{Score}$ is a real number giving the basic score of the evaluation.

Before describing how the score is computed, let us say immediately that $\text{betterthan}$ is defined on evaluation structures by:

$$\text{eval}(H_1,F_1,S_1) \text{ betterthan } \text{eval}(H_2,F_2,S_2)$$

iff $H_1=H_2$ and $F_1=F_2$ and $S_1<S_2$.

Thus two evaluation structures (and hence two results with the same boundaries) are comparable by $\text{betterthan}$ if and only if they have the same heads and basic features. And within the equivalence class of results having the same heads, basic features and boundaries, we get a total ordering defined by the score.

Now let us turn to the computation of the score of a result (the score component of its evaluation structure). This is defined by a formula

$$\text{Score} = CPEval + AdEval + HEval.$$ 

The purposes of the three components are as follows.

- $CPEval$ controls for parallelism in coordinated phrases.
- $AdEval$ controls for complement preference, and is roughly the total number of adjuncts in the phrase.
- $HEval$ is the Heidorn evaluation of the phrase [Heidorn 1982], which controls for close attachment.

Let us look at the definitions of the three components. (Recall that a lower score is a better score.)

The Heidorn evaluation $H(F)$ of a phrase $F$ is defined recursively as the sum of all terms $0.1*\text{H(\text{Mod})+1}$, where $\text{Mod}$ varies over the modifiers of $F$. (One need not state the base of this recursive formula separately, since one arrives eventually at phrases with no modifiers, and then the sum over the empty list is understood to be zero.) Actually, the factor 0.1 used in the recursive formula is just a default. An exception rule, associated with a slot in the grammar, can specify a different factor. Also, exception rules can change the basic formula applied, in a way that will not be described here.

As mentioned above, the $AdEval$ component of a score (used for complement preference) is defined roughly as the total number of adjunct modifiers in the phrase (on all levels). Thus we could say that roughly for each adjunct slot filling, 1 is added to the score, and for each complement slot filling, 0 is added to the score (in this connection). This is the default, but it

---

10 The exposition here is slightly simplified over what is in the actual system.
can be overridden for a slot $\text{slot}$ if there is an evaluation rule for $\text{slot}$ in the grammar. Such a rule looks like a Prolog clause for the predicate

$$\text{adeval}(\text{slot}, A)$$

where $A$ is a number to be added for the slot in place of the default. The body of such a clause (if present) can contain selector goals (just as in slot filler rules) which refer to any parts of the filler phrase or the higher phrase.

Now let us look at the $\text{CPEval}$ component of a score. When two phrases are coordinated, $\text{coordfeas}$ requires that their feature structures be similar (normally having the same part of speech). But it may not require that the feature structures be exactly the same. For instance, the feature structures of prepositional phrases exhibit the head preposition, and we do not want to require that coordinated PPs have the same head preposition. But when we are deciding which of several PPs to coordinate a given one with, it is better (for greater parallelism) to choose one with the same head. This happens if the feature structures of the PPs are exactly the same. To capture this preference in the score for a coordinated phrase we add 0 or 1 to the $\text{CPEval}$ component according as the feature structures are “exactly the same” or not.\(^{11}\)

The $\text{CPEval}$ component controls for another aspect of parallelism in coordination, namely parallelism in the configurations of modifiers of each conjunct. The idea, as implemented currently, is just that it is preferrable that the conjuncts have nearly the same number of left modifiers and nearly the same number of right modifiers. Specifically, for each coordination, the number

$$|\#L\text{Mods}_1 - \#L\text{Mods}_2| + |\#R\text{Mods}_1 - \#R\text{Mods}_2|$$

is added to the $\text{CPEval}$ component of the score. Here $\#L\text{Mods}_1$ and $\#L\text{Mods}_2$ are the numbers of left modifiers in the left and right conjuncts, respectively, and $\#R\text{Mods}_1$ and $\#R\text{Mods}_2$ are the numbers of right modifiers in the left and right conjuncts, respectively. (There is a slight amendment to this if one of the conjuncts itself is coordinated.) As an example, for the phrase the old men and women, the analysis [the old] [men and women] gets 0 added to $\text{CPEval}$, but the analysis [the old men] and [women] gets 2 added to $\text{CPEval}$.

Finally, the $\text{CPEval}$ component controls for similarity in the slot frames of the two conjuncts. Recall from Section 6 that the procedure $\text{coordframe}$ does require some similarity in the slot frames of the two conjuncts, but the frames may differ in the presence of some optional slots. However, such slots can create a penalty that is added to the $\text{CPEval}$ component. There are defaults for such penalties, and one may also specify exceptions in the grammar.

\(^{11}\) This notion can be defined recursively as follows. (1) Any atomic term is exactly the same as itself. (2) Any two variables are exactly the same (they need not be unified). (3) Two compound terms are exactly the same if they have the same principal functor and the same arity, and their corresponding arguments are exactly the same. There is a built-in predicate $\text{=}=$ in VM/Prolog that checks this directly.
References


Shieber, S. M. [1986] An Introduction to Unification-Based Approaches to Grammar, CSLI Lecture Notes No. 4, Center for the Study of Language and Information, Stanford, CA.

