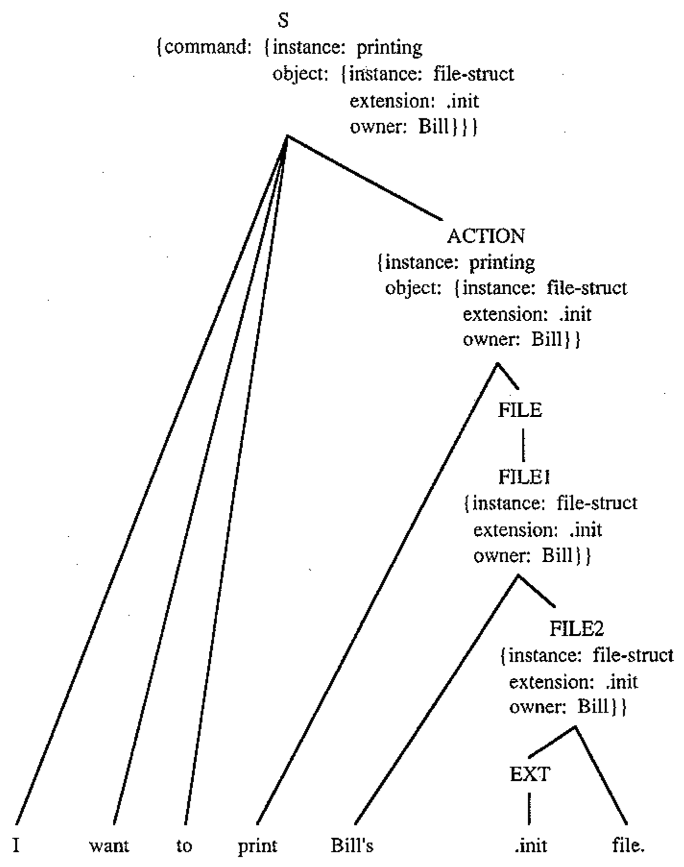


Figure 15.11: The Result of Parsing with a Semantic Grammar

Notice that in this approach, we have combined into a single process all five steps of Section 15.1.1 with the exception of the final part of pragmatic processing in which the conversion to the system's command syntax is done.

The principal advantages of semantic grammars are the following:

- When the parse is complete, the result can be used immediately without the additional stage of processing that would be required if a semantic interpretation had not already been performed during the parse.
- Many ambiguities that would arise during a strictly syntactic parse can be avoided since some of the interpretations do not make sense semantically and thus cannot be generated by a semantic grammar. Consider, for example, the sentence "I want to print stuff.txt on printer3." During a strictly syntactic parse, it would not be possible to decide whether the prepositional phrase, "on printer3" modified "want" or "print." But using our semantic grammar, there is no general notion of a prepositional phrase and there is no attachment ambiguity.
- Syntactic issues that do not affect the semantics can be ignored. For example, using the grammar shown above, the sentence, "What is the extension of .lisp file?" would be parsed and accepted as correct.

There are, however, some drawbacks to the use of semantic grammars:

- The number of rules required can become very large since many syntactic generalizations are missed.
- Because the number of grammar rules may be very large, the parsing process may be expensive.

After many experiments with the use of semantic grammars in a variety of domains, the conclusion appears to be that for producing restricted natural language interfaces quickly, they can be very useful. But as an overall solution to the problem of language understanding, they are doomed by their failure to capture important linguistic generalizations.

15.3.2 Case Grammars

Case grammars [Fillmore, 1968; Bruce, 1975] provide a different approach to the problem of how syntactic and semantic interpretation can be combined. Grammar rules are written to describe syntactic rather than semantic regularities. But the structures the rules produce correspond to semantic relations rather than to strictly syntactic ones. As an example, consider the two sentences and the simplified forms of their conventional parse trees shown in Figure 15.12.

Although the semantic roles of "Susan" and "the file" are identical in these two sentences, their syntactic roles are reversed. Each is the subject in one sentence and the object in another.

Using a case grammar, the interpretations of the two sentences would both be

```
(printed (agent Susan)
         (object File))
```

My lawn hates the cold.

Now there is no animate subject available, and so the metaphorical use of lawn acting as an animate object should be accepted.

Unfortunately, to solve the lexical disambiguation problem completely, it becomes necessary to introduce more and more finely grained semantic markers. For example, to interpret the sentence about Susan's diamond correctly, we must mark one sense of diamond as SHIMMERABLE, while the other two are marked NONSHIMMERABLE. As the number of such markers grows, the size of the lexicon becomes unmanageable. In addition, each new entry into the lexicon may require that a new marker be added to each of the existing entries. The breakdown of the semantic marker approach when the number of words and word senses becomes large has led to the development of other ways in which correct senses can be chosen. We return to this issue in Section 15.3.4.

Sentence-Level Processing

Several approaches to the problem of creating a semantic representation of a sentence have been developed, including the following:

- Semantic grammars, which combine syntactic, semantic, and pragmatic knowledge into a single set of rules in the form of a grammar. The result of parsing with such a grammar is a semantic, rather than just a syntactic, description of a sentence.
- Case grammars, in which the structure that is built by the parser contains some semantic information, although further interpretation may also be necessary.
- Conceptual parsing, in which syntactic and semantic knowledge are combined into a single interpretation system that is driven by the semantic knowledge. In this approach, syntactic parsing is subordinated to semantic interpretation, which is usually used to set up strong expectations for particular sentence structures.
- Approximately compositional semantic interpretation, in which semantic processing is applied to the result of performing a syntactic parse. This can be done either incrementally, as constituents are built, or all at once, when a structure corresponding to a complete sentence has been built.

In the following sections, we discuss each of these approaches.

15.3.1 Semantic Grammars

A *semantic grammar* [Burton, 1976; Hendrix *et al.*, 1978; Hendrix and Lewis, 1981] is a context-free grammar in which the choice of nonterminals and production rules is governed by semantic as well as syntactic function. In addition, there is usually a semantic action associated with each grammar rule. The result of parsing and applying all the associated semantic actions is the meaning of the sentence. This close coupling of semantic actions to grammar rules works because the grammar rules themselves are designed around key semantic concepts.

```
S → what is FILE-PROPERTY of FILE?
        {query FILE.FILE-PROPERTY}
S → I want to ACTION
        {command ACTION}
FILE-PROPERTY → the FILE-PROP
        {FILE-PROP}
FILE-PROP → extension | protection | creation date | owner
        {value}
FILE → FILE-NAME | FILE1
        {value}
FILE1 → USER's FILE2
        {FILE2.owner: USER}
FILE1 → FILE2
        {FILE2}
FILE2 → EXT file
        {instance: file-struct
         extension: EXT}
EXT → .init | .txt | .lsp | .for | .ps | .mss
        value
ACTION → print FILE
        {instance: printing
         object: FILE}
ACTION → print FILE on PRINTER
        {instance: printing
         object: FILE
         printer: PRINTER}
USER → Bill | Susan
        {value}
```

Figure 15.10: A Semantic Grammar

An example of a fragment of a semantic grammar is shown in Figure 15.10. This grammar defines part of a simple interface to an operating system. Shown in braces under each rule is the semantic action that is taken when the rule is applied. The term "value" is used to refer to the value that is matched by the right-hand side of the rule. The dotted notation $x.y$ should be read as the y attribute of the unit x. The result of a successful parse using this grammar will be either a command or a query.

A semantic grammar can be used by a parsing system in exactly the same ways in which a strictly syntactic grammar could be used. Several existing systems that have used semantic grammars have been built around an ATN parsing system, since it offers a great deal of flexibility.

Figure 15.11 shows the result of applying this semantic grammar to the sentence

I want to print Bill's .init file.

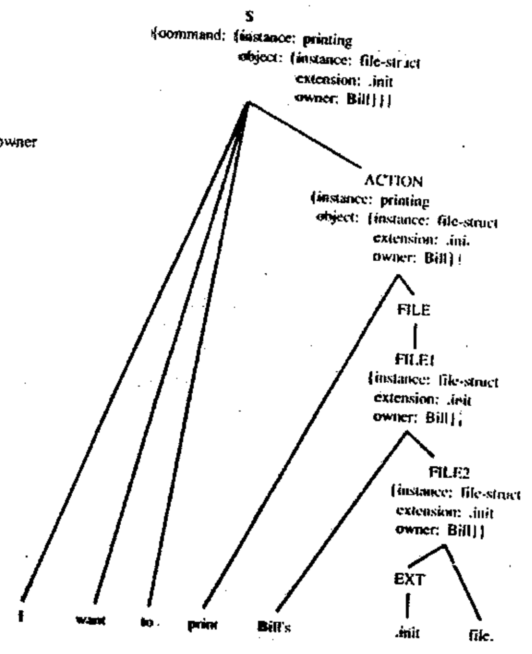
Semantic grammar

Semantic as well as syntactic rules & nonterminals
Semantic actions

Meaning = parse result + result of actions

Eg. operating system interface

```
S → what is FILE-PROPERTY of FILE?
        {query FILE.FILE-PROPERTY}
S → I want to ACTION
        {command ACTION}
FILE-PROPERTY → the FILE-PROP
        {FILE-PROP}
FILE-PROP → extension | protection | creation date | owner
        {value}
FILE → FILE-NAME | FILE1
        {value}
FILE1 → USER's FILE2
        {FILE2.owner: USER}
FILE1 → FILE2
        {FILE2}
FILE2 → EXT file
        {instance: file-struct
         extension: EXT}
EXT → .init | .txt | .lsp | .for | .ps | .mss
        value
ACTION → print FILE
        {instance: printing
         object: FILE}
ACTION → print FILE on PRINTER
        {instance: printing
         object: FILE
         printer: PRINTER}
USER → Bill | Susan
        {value}
```



Can be parsed e.g. with ATN

Advantages

result is immediately usable
  good for interfaces

ambiguities can be avoided using semantic info

syntax can be overridden (some)

but

ad hoc rules; anything goes

lots of rules

slow

Not a general model,
but good for limited interfaces