ADVANCES IN COMPUTERS, VOL. 17

Semantics and Quantification in Natural Language Question Answering

W. A. WOODS

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

1. Introduction 2
2. Historical Context 4
   2.1 Airlines Flight Schedules 4
   2.2 Answering Questions about ATN Grammars 6
   2.3 The LUNAR System 6
   2.4 TRIPSYS 8
3. Overview 8
   3.1 Structure of the LUNAR System 9
   3.2 Semantics in LUNAR 10
4. The Meaning Representation Language 11
   4.1 Designators 11
   4.2 Propositions 12
   4.3 Commands 12
   4.4 Quantification 12
   4.5 Specification of the MRL Syntax 13
   4.6 Procedural/Declarative Duality 14
   4.7 Opaque Contexts 15
   4.8 Restricted Class Quantification 17
   4.9 Nonstandard Quantifiers 17
   4.10 Functions and Classes 20
   4.11 Unanticipated Requests 20
5. The Semantics of the Notation 21
   5.1 Procedural Semantics 21
   5.2 Enumeration Functions 21
   5.3 Quantified Commands 23
6. Semantic Interpretation 24
   6.1 Complications Due to Quantifiers 26
   6.2 Problems with an Alternative Approach 26
   6.3 The Structure of Semantic Rules 27
   6.4 Relationship of Rules to Syntax 29
   6.5 Organization of the Semantic Interpreter 30
   6.6 Organization of Rules 33
   6.7 The Generation of Quantifiers 35
7. Problems of Interpretation 37
   7.1 The Order of Quantifier Nesting 37
   7.2 Interaction of Negations with Quantifiers 38

ISBN 0-12-012117-4

7.3 Functional Nesting and Quantifier Reversal 39
7.4 Relative Clauses 41
7.5 Other Types of Modifiers 42
7.6 Averages and Quantifiers 44
7.7 Short Scope/Broad Scope Distinctions 45
7.8 Wh Questions 47
8. Post-Interpretive Processing 54
8.1 Smart Quantifiers 55
8.2 Printing Quantifier Dependencies 57
9. An Example 58
10. Loose Ends, Problems, and Future Directions 64
10.1 Approximate Solutions 64
10.2 Modifier Placement 65
10.3 Multiple Uses of Constituents 68
10.4 Ellipsis 69
10.5 Plausibility of Alternative Interpretations 70
10.6 Anaphoric Reference 70
10.7 Ill-Formed Input and Partial Interpretation 71
10.8 Intensional Inference 73
11. Syntactic/Semantic Interactions 75
11.1 The Role of Syntactic Structure 75
11.2 Grammar Induced Phasing of Interpretation 77
11.3 Semantic Interpretation while Parsing 78
11.4 Top-Down versus Bottom-Up Interpretation 79
11.5 Pragmatic Grammars 80
11.6 Semantic Interpretation in the Grammar 81
11.7 Generating Quantifiers while Parsing 84
12. Conclusions 84
References 86

1. Introduction

The history of communication between man and machines has followed a path of increasing provision for the convenience and ease of communication on the part of the human. From raw binary and octal numeric machine languages, through various symbolic assembly, scientific, business and higher level languages, programming languages have increasingly adopted notations that are more natural and meaningful to a human user. The important characteristic of this trend is the elevation of the level at which instructions are specified from the low level details of the machine operations to high level descriptions of the task to be done, leaving out details that can be filled in by the computer. The ideal product of such continued evolution would be a system in which the user specifies what he wants done in a language that is so natural that negligible mental effort is required to recast the specification from the form in which he formulates it to that which the machine requires. The logical choice for such a language is the person's own natural language (which in this paper I will assume to be English).

For a naïve, inexperienced user, almost every transaction with current computer systems requires considerable mental effort deciding how to express the request in the machine's language. Moreover, even for technical specialists who deal with a computer constantly, there is a distinction between the things that they do often and remember well, and many other things that require consulting a manual and/or much conscious thought in order to determine the correct machine "incantation" to achieve the desired effect. Thus, whether a user is experienced or naïve, and whether he is a frequent or occasional user, there arise occasions where he knows what he wants the machine to do and can express it in natural language, but does not know exactly how to express it to the machine. A facility for machine understanding of natural language could greatly improve the efficiency of expression in such situations—both in speed and convenience, and in decreased likelihood of error.

For a number of years, I have been pursuing a long range research objective of making such communication possible between a man and a machine. During this period, my colleagues and I[1] have constructed several natural language question-answering systems and developed a few techniques for solving some of the problems that arise. In this paper, I will present some of those techniques, focusing on the problem of handling natural quantification as it occurs in English. As an organizing principle, I will present the ideas in a roughly historical order, with commentary on the factors leading to the selection of various notations and algorithms, on limitations that have been discovered as a result of experience, and on directions in which solutions lie.

Among the systems that I will use for examples are a flight schedules question-answering system (Woods, 1967, 1968), a system to ask questions about an augmented transition network (ATN) grammar (not previously published), the LUNAR system, which answers questions about the chemical analyses of the Apollo 11 moon rocks (Woods *et al.*, 1972; Woods, 1973b), and a system for natural language trip planning and budget management (Woods *et al.*, 1976).

Some of the techniques used in these systems, especially the use of the ATN grammar formalism (Woods, 1969, 1970, 1973a), have become widely known and are now being used in many different systems and applications. However, other details, including the method of performing semantic interpretation, the treatment of quantification and anaphoric

[1] Principal contributors to one or more of the systems described here include Madeleine Bates, Bertram Bruce, Ronald Kaplan, and Bonnie Nash-Webber (now Webber).

reference, and several other problems, have not been adequately described in accessible publications.

This paper is intended to be a discussion of a set of techniques, the problems they solve, and the relative advantages and disadvantages of several alternative approaches. Because of the length of the presentation, no attempt has been made to survey the field or give an exhaustive comparison of these techniques to those of other researchers. In general, most other systems are not sufficiently formalized at a conceptual level that such comparisons can be made on the basis of published information. In some cases, the mechanisms described here can be taken as models of what is being done in other systems. Certainly, the general notion of computing a representation of the meaning of a phrase from representations of the meanings of its constituents by means of a rule is sufficiently general to model virtually any semantic interpretation process. The details of how most systems handle such problems as the nesting of multiple quantification, however, are difficult to fathom. Hopefully the presentation here and the associated discussion will enable the reader to evaluate for himself, with some degree of discrimination, the capabilities of other systems.

2. Historical Context

2.1 Airlines Flight Schedules

Airlines flight schedules was the focusing context for a gedanken system for semantic interpretation that I developed as my Ph.D. thesis at Harvard University (Woods, 1967). In that thesis, I was concerned with the problem of "semantic interpretation"—making the transition from a syntactic analysis of input questions (such as could be produced by parsing with a formal grammar of English) to a concrete specification of what the computer was to do to answer the question. Prior to that time, this problem had usually been attacked by developing a set of structural conventions for storing answers in the data base and transforming the input questions (frequently by ad hoc procedures) into patterns that could be matched against that data base. Simmons (1965) presents a survey of the state of the art of the field at that time..

In many of the approaches existing at that time, the entire process of semantic interpretation was built on particular assumptions about the structure of the data base. I was searching for a method of semantic interpretation that would be independent of particular assumptions about data base structure and, in particular, would permit a single language understanding system to talk to many different data bases and permit the specification of requests whose answers required the integration of information from several different data bases. In searching for such an approach, I looked more to the philosophy of language and the study of meaning than to data structures and data base design.

The method I developed was essentially an interpretation of Carnap's notion of truth conditions (Carnap, 1964a). I chose to represent those truth conditions by formal procedures that could be executed by a machine. The representation that I used for expressing meanings was at once a notational variant of the standard predicate calculus notation and also a representation of an executable procedure. The ultimate definition of the meanings of expressions in this notation were the procedures that they would execute to determine the truth of propositions, compute the answers to questions, and carry out commands. This notion, which I referred to as "procedural semantics," picks up the chain of semantic specification from the philosophers at the level of abstract truth conditions, and carries it to a formal specification of those truth conditions as procedures in a computer language.

The idea of procedural semantics has since had considerable success as an engineering technique for constructing natural language understanding systems, and has also developed somewhat as a theory of meaning. In my paper "Meaning and Machines" (Woods, 1973c), I discuss some of the more theoretical issues of the adquacy of procedural semantics as a theory of meaning.

The flight schedules application initially served to focus the issues on particular meanings of particular sentences. The application assumed a data base essentially the same as the information contained in the Official Airline Guide (OAG, 1966)—that is, a list of flights, their departure and arrival times from different airports, their flight numbers and airlines, number of stops, whether they serve meals, etc. Specific questions were interpreted as requesting operations to be performed on the tables that make up this data base to compute answers.

The semantic interpretation system presented in my thesis was subsequently implemented for this application with an ATN grammar of English to provide syntax trees for interpretation, but without an actual data base. The system produced formal semantic interpretations for questions such as:

"What flights go from Boston to Washington?"
"Is there a flight to Washington before 8:00 A.M.?"
"Do they serve lunch on the 11:00 A.M. flight to Toronto?"

2.2 Answering Questions about ATN Grammars

To prove the point that the semantic interpretation system used in the flight schedules domain was in fact general for arbitrary data bases and independent of the detailed structure of the data base, immediately after completing that system, I looked for another data base to which I could apply the method. I wanted a data base that had not been designed to satisfy any assumptions about the method of question interpretation to be used. The most convenient such data base that I had at hand was the data structure for the ATN grammar that was being used by the system to parse its input sentences. This data base had a structure that was intended to support the parser, and had not been designed with any forethought to using it as a data base for question answering.

An ATN grammar, viewed as a data base, conceptually consists of a set of named states with arcs connecting them, corresponding to transitions that can be made in the course of parsing. Arcs connecting states are of several kinds depending on what, if anything, they consume from the input string when they are used to make a transition. For example, a word arc consumes a single word from the input, a push arc consumes a constituent phrase of the type pushed for, and a jump arc consumes no input but merely makes a state transition (see Woods, 1970, 1973a, 1975a, for further discussion of ATN grammars). These states and arcs constitute the data base entities about which questions may be asked.

In addition to the entities that actually exist as data objects in the internal structure for the grammar, there are some other important objects that exist conceptually but are not explicit in the grammar. The most important such entity is a path. A path is a sequence of arcs that connect to each other in the order in which they could be taken in the parsing of a sentence. Although paths are implicit in the grammar, they are not explicit in the data structure, i.e., there is no internal data object that can be pointed to in the grammar that corresponds to a path. Nevertheless, one should be able to talk about paths and ask questions about them. The techniques I will describe can handle such entities.

Examples of the kinds of sentences this "grammar information system" could deal with are

"Is there a jump arc from state S/ to S/NP?"
"How many arcs leave state NP/?"
"How many nonlooping paths connect state S/ with S/POP?"
"Show me all arcs entering state S/VP."

2.3 The LUNAR System

The LUNAR system (Woods *et al.*, 1972; Woods, 1973b) was originally developed with support from the NASA Manned Spacecraft Center as a research prototype for a system to enable a lunar geologist to conveniently access, compare, and evaluate the chemical analysis data on lunar rock and soil composition that was accumulating as a result of the Apollo moon missions. The target of the research was to develop a natural language understanding facility sufficiently natural and complete that the task of selecting the wording for a request would require negligible effort for the geologist user.

The application envisaged was a system that would be accessible to geologists anywhere in the country by teletype connections and would enable them to access the NASA data base without having to learn either the programming language in which the system was implemented or the formats and conventions of the data base representations. For example, the geologist should be able to ask questions such as "What is the average concentration of aluminum in high-alkali rocks?" without having to know that aluminum was conventionally represented in the data base as AL2O3, that the high-alkali rocks (also known as "volcanics" or "fine-grained igneous") were conventionally referred to as TYPEAS in the data base, nor any details such as the name of the file on which the data was stored, the names of the fields in the data records, or any of a myriad of other details normally required to use a data base system.

To a substantial extent, such a capability was developed, although never fully put to the test of real operational use. In a demonstration of a preliminary version of the system in 1971 (Woods, 1973b), 78% of the questions asked of the system were understood and answered correctly, and another 12% failed due to trivial clerical errors such as dictionary coding errors in the not fully debugged system. Only 10% of the questions failed because of significant parsing or semantic interpretation problems. Although the requests entered into the system were restricted to questions that were in fact about the contents of the data base, and comparatives (which were not handled at that time) were excluded, the requests were otherwise freely expressed in natural English without any prior instructions as to phrasing and were typed into the system exactly as they were asked.

The LUNAR system allowed a user to ask questions, compute averages and ratios, and make listings of selected subsets of the data. One could also retrieve references from a keyphrase index and make changes to the data base. The system permitted the user to easily compare the measurements of different researchers, compare the concentrations of elements or isotopes in different types of samples or in different phases of a sample, compute averages over various classes of samples, compute ratios of two constituents of a sample, etc., all in straightforward natural English.

Examples of requests understood by the system are

"Give me all lunar samples with magnetite."
"In which samples has apatite been identified?"
"What is the specific activity of Al26 in soil?"
"Analyses of strontium in plagioclase."
"What are the plag analyses for breccias?"
"What is the average concentration of olivine in breccias?"
"What is the average age of the basalts?"
"What is the average potassium/rubidium ratio in basalts?"
"In which breccias is the average concentration of titanium greater than 6 percent?"

2.4 TRIPSYS

TRIPSYS is a system that was developed as the context for a research project in continuous speech understanding (Woods *et al.*, 1976). The overall system of which it was a part was called HWIM (for "Hear What I Mean"). TRIPSYS understands and answers questions about planned and taken trips, travel budgets and their status, costs of various modes of transportation to various places, per diems in various places, conferences and other events for which trips might be taken, people in an organization, the contracts they work on, the travel budgets of those contracts, and a variety of other information that is useful for planning trips and managing travel budgets. It is intended to be a small-scale example of a general management problem. TRIPSYS also permits some natural language entry of information into the data base, and knows how to prompt the user for additional information that was not given voluntarily. Examples of the kinds of requests that TRIPSYS was designed to handle are

"Plan a trip for two people to San Diego to attend the ASA meeting."
"Estimate the cost of that trip."
"Is there any money left in the Speech budget?"

3. Overview

Since the LUNAR system is the most fully developed and most widely known of the above systems, I will use it as the principal focus throughout this paper. A brief overview of the LUNAR system was presented in the 1973 National Computer Conference (Woods, 1973b), and an extensive technical report documenting the system was produced (Woods *et al.*, 1972). However, there has been no generally available document that gives a sufficiently complete picture of the capabilities of the system and how it works. Consequently, I will first give a brief introduction to the structure of the system as a whole, and then proceed to relatively detailed accounts of some of the interpretation problems that were solved. Examples from the other three systems will be used where they are more self-explanatory or more clearly illustrate a principle. Where the other systems differ in structure from the LUNAR system, that will be pointed out.

3.1 Structure of the LUNAR System

The LUNAR system consists of three principal components: a general purpose grammar and parser for a large subset of natural English, a rule-driven semantic interpretation component using pattern $\rightarrow$ action rules for transforming a syntactic representation of an input sentence into a representation of what it means, and a data base retrieval and inference component that stores and manipulates the data base and performs computations on it. The first two components constitute a language understanding component that transforms an input English sentence into a disposable program for carrying out its intent (answering a question or making some change to the data base). The third component executes such programs against the data base to determine the answer to queries and to effect changes in the data base.

The system contains a dictionary of approximately 3500 words, a grammar for a fairly extensive subset of natural English, and two data bases: a table of chemical analyses with 13,000 entries, and a topic index to documents with approximately 10,000 postings. The system also contains facilities for morphological analysis of regularly inflected words, for maintaining a discourse directory of possible antecedents for pronouns and other anaphoric expressions, and for determining how much and what information to display in response to a request.

The grammar used by the parsing component of the system is an augmented transition network (ATN). The ATN grammar model has been relatively well documented elsewhere (Woods, 1970, 1973a), so I will not go into detail here describing it, except to point out that it produces syntactic tree structures comparable to the "deep structures" assigned by a Chomsky type transformational grammar, vintage 1965 (Chomsky, 1965). Likewise, I will not go into much detail describing the inner workings of the data base inference and retrieval component, except to describe the semantics of the formal meaning representation language and discuss some of its advantages. What I will describe here are the problems of semantic interpretation that were handled by the system.

All of the systems mentioned in Section 2 share this same basic structure with the following exceptions:

1) The airline flight schedules problem was implemented up through the parsing and interpretation stage, but was never coupled to a real data base. This system was implemented solely to validate the formal semantic interpretation procedure.

2) The TRIPSYS system does not construct a separate syntactic tree structure to be given to a semantic interpreter, but rather the ATN grammar builds semantic interpretations directly as its output representation.

3.2 Semantics in LUNAR

A semantic specification of a natural language consists of essentially three parts:

a) a meaning representation language (MRL)—a notation for semantic representation for the meanings of sentences,

b) a specification of the semantics of the MRL notation, i.e., a specification of what its expressions mean, and

c) a semantic interpretation procedure, i.e., a procedure to construct the appropriate semantic representations for a given natural language sentence.

Accordingly, the semantic framework of the LUNAR system consists of three parts: a semantic notation in which to represent the meanings of sentences, a specification of the semantics of this notation (by means of formal procedures), and a procedure for assigning representations in the notation to input sentences.

In previous writings on LUNAR, I have referred to the semantic notation as a query language, but I will refer to it here, following a currently more popular terminology as a "meaning representation language" or MRL. To represent expressions in the MRL, I will use the so-called "Cambridge Polish" notation in wich the application of an operator to its arguments is represented with the operator preceding its operands and the entire group surrounded by parentheses. This notation places the operator in a standard position independent of the number of arguments it takes and uses the parentheses to indicate scoping of operators rather than depending on a fixed degree of the operator as in the "ordinary" Polish prefix notation (thus facilitating operators that take a variable number of arguments). Cambridge Polish notation is the notation used for the S-expressions of the programming language LISP (Bobrow *et al.*, 1968), in which LUNAR is implemented.

Occasionally, the notations used for illustration will be slightly simplified from the form actually used in LUNAR to avoid confusion. For example, the DATALINE function used in LUNAR actually takes an additional argument for a data file that is omitted here.

4. The Meaning Representation Language

There are a number of requirements for a meaning representation language, but the most important ones are these:

a) It must be capable of representing precisely, formally, and unambiguously any interpretation that a human reader can place on a sentence.
b) It should facilitate an algorithmic translation from English sentences into their corresponding semantic representations.
c) It should facilitate subsequent intelligent processing of the resulting interpretation.

The LUNAR MRL consists of an extended notational variant of the ordinary predicate calculus notation and contains essentially three kinds of constructions:

- designators, which name or denote objects (or classes of objects) in the database,
- propositions, which correspond to statements that can be either true or false in the data base, and
- commands, which initiate and carry out actions.

4.1 Designators

Designators come in two varieties—individual specifiers and class specifiers. Individual specifiers correspond to proper nouns and variables. For example, S10046 is a designator for a particular sample, OLIV is a designator for a certain mineral (olivine), and X3 can be a variable denoting any type of object in the data base. Class specifiers are used to denote classes of individuals over which quantification can range. They consist of the name of an enumeration function for the class plus possible arguments. For example, (SEQ TYPECS) is a specification of the class of type C rocks (i.e., breccias) and (DATALINE S10046 OVERALL OLIV) is a specification of the set of lines of a table of chemical analyses corresponding to analyses of sample S10046 for the overall concentration of olivine.

4.2 Propositions

Elementary propositions in the MRL are formed from predicates with designators as arguments. Complex propositions are formed from these by use of the logical connectives AND, OR, and NOT and by quantification. For example, (CONTAIN S10046 OLIV) is a proposition formed by substituting designators as arguments to the predicate CONTAIN, and

```
(AND (CONTAIN X3 OLIV) (NOT (CONTAIN X3 PLAG)))
```

is a complex proposition corresponding to the assertion that X3 contains olivine but does not contain plagioclase.

4.3 Commands

Elementary commands consist of the name of a command operator plus arguments. As for propositions, complex commands can be constructed using logical connectives and quantification. For example, TEST is a command operator for testing the truth value of a proposition given as its argument. Thus

```
(TEST (CONTAIN S10046 OLIV))
```

will answer yes or no depending on whether sample S10046 contains olivine. Similarly, PRINTOUT is a command operator which prints out a representation for a designator given as its argument.

4.4 Quantification

An important aspect of the meaning of English sentences that must be captured in any MRL is the use of quantifiers such as "every" and "some." Quantification in the LUNAR MRL is represented in an elaborated version of the traditional predicate calculus notation. An example of an expression in this notation is

```
(FOR EVERY X1 / (SEQ SAMPLES) :
     (CONTAIN X1 OVERALL SILICON) ; (PRINTOUT X1)).
```

This says, "for every object X1 in the set of samples such that X1 contains silicon, print out (the name of) X1."

In general, an instance of a quantified expression takes the form

```
(FOR ⟨quant⟩ X / ⟨class⟩ : (p X) ; (q X))
```

where ⟨quant⟩ is a specific quantifier such as EVERY or SOME, X is the variable of quantification and occurs open in the expressions (p X) and (q X), ⟨class⟩ is a set over which quantification is to range, (p X) is a proposition that restricts the range, and (q X) is the expression being quantified (which may be either a proposition or a command).

For the sake of simplifying some examples, I will generalize the format of the quantification operator so that the restriction operation implied by the ":" can be repeated any number of times (including zero if there is no further restriction on the range), giving rise to forms such as

(FOR ⟨quant⟩ X / ⟨class⟩ ; (q X))

and

(FOR ⟨quant⟩ X / ⟨class⟩ : (p X) : (r X) ; (q X)).

When there is no restriction on the range of quantification, this can also be indicated by using the universally true proposition T, as in

(FOR ⟨quant⟩ X / ⟨class⟩ : T ; (q X)).

4.5 Specification of the MRL Syntax

A formal BNF specification of the LUNAR MRL is given here:

```
⟨expression⟩ = ⟨designator⟩ | ⟨proposition⟩ | ⟨command⟩
⟨designator⟩ = ⟨individual constant⟩ |
               ⟨variable⟩ |
               (⟨function⟩ ⟨expression⟩* )
⟨proposition⟩ = ⟨elementary proposition⟩ |
                ⟨quantified proposition⟩
⟨elementary proposition⟩ = (⟨propositional operator⟩
                           ⟨expression⟩* )
⟨propositional operator⟩ = ⟨predicate⟩ | ⟨logical operator⟩
⟨logical operator⟩ = AND | OR | NOT | IF-THEN . . .
⟨quantified proposition⟩ = (FOR ⟨variable⟩ / ⟨class⟩ ;
                           ⟨proposition⟩)
⟨class⟩ = ⟨elementary class⟩ | ⟨restricted class⟩
⟨elementary class⟩ = ⟨class name⟩ |
                     (⟨class function⟩ ⟨expression⟩* )
⟨restricted class⟩ = ⟨class⟩ : ⟨proposition⟩
⟨command⟩ = ⟨elementary command⟩ | ⟨quantified command⟩
⟨elementary command⟩ = (⟨command operator⟩ ⟨expression⟩* )
⟨quantified command⟩ = (FOR ⟨variable⟩ / ⟨class⟩ ; ⟨command⟩)
```

In addition to the above BNF constraints, each general operator (i.e., function, predicate, logical operator, class function, or command operator) will have particular restrictions on the number and kinds of expressions that it can take as arguments in order to be meaningful. Each operator also specifies which of its arguments it takes literally as given, and which it will evaluate to obtain a referent (see discussion of opaque contexts below).

Predicates, functions, class names, class functions, command operators, and individual constants are all domain-dependent entities which are to be specified for a particular application domain and defined in terms of procedures. In LUNAR, they are defined as LISP subroutines. Individual constants are defined by procedures for producing a reference pointer to the appropriate internal object in the computer's model of the world; functions are defined by procedures for producing a reference pointer to the appropriate value given the values for the arguments; class names and class functions are defined by procedures that (given the appropriate values for arguments) can enumerate the members of their class one at a time; predicates are defined by procedures which, given the values of their arguments, determine a truth value for the corresponding proposition; and command operators are defined by procedures which, given the values of their arguments, can carry out the corresponding commands.

I should point out that the defintion given here for classes and commands are not adequate for a general theory of semantics, but are rather more pragmatic definitions that facilitate question answering and computer response to commands. For a general semantic theory, the requirement for semantic definition of a class is merely a procedure for recognizing a member, and the semantic definition for a command is a procedure for recognizing when it has been carried out. That is, to be said to know the meaning of a command does not require the ability to carry it out, and to know the meaning of a noun does not require an ability to enumerate all members of its extension. The distinction between knowing how, and just knowing whether, marks the difference between pragmatic utility and mere semantic adequacy. The requirements placed on the definitions of the classes and commands in the LUNAR system are thus more stringent than those required for semantic definition alone.

4.6 Procedural/Declarative Duality

The meaning representation language used in LUNAR is intended to serve both as a procedural specification that can be executed to compute an answer or carry out a command, and as a "declarative" representation that can be manipulated as a symbolic object by a theorem prover or other inference system. By virtue of the definition of primitive functions and predicates as LISP functions, the language can be viewed simultaneously as a higher level programming language and as an extension of the predicate calculus. This gives rise to two different possible types of inference for answering questions, corresponding roughly to Carnap's distinction between *intension* and *extension* (Carnap, 1964b). First, because of its definition by means of procedures, a question such as "Does every sample contain silicon?" can be answered *extensionally* (that is, by appeal to the individuals denoted by the class name "samples") by enumerating the individual samples and checking whether silicon has been found in each one. On the other hand, this same question could have been answered *intensionally* (that is, by consideration of its meaning alone without reference to the individuals denoted) by means of the application of inference rules to other (intensional) facts (such as the assertion "Every sample contains some amount of each element"). Thus the expressions in the meaning representation language are capable either of direct execution against the data base (extensional mode) or manipulation by mechanical inference algorithms (intensional mode).

In the LUNAR system, the principal mode of inference is extensional, that is, the direct evaluation of the formal MRL expression as a procedure. However, in certain circumstances, this expression is also manipulated as a symbolic object. Such cases include the construction of descriptions for discourse entities to serve as antecedents for anaphoric expressions and the use of "smart quantifiers" (to be discussed later) for performing more efficient quantification. Extensional inference has a variety of limitations (e.g., it is not possible to prove assertions about infinite sets in extensional mode), but it is a very efficient method for a variety of question-answering applications.

4.7 Opaque Contexts

As mentioned above, the general operators in the meaning representation language are capable of accessing the arguments they are given either literally or after evaluation. Thus, an operator such as ABOUT in an expression like

(ABOUT D70-181 (TRITIUM PRODUCTION))

(meaning "Document D70-181 discusses tritium production") can indicate as part of its definition that, in determining the truth of an assertion, the first argument (D70-181 in this case) is to be evaluated to determine its referent, while the second argument (TRITIUM PRODUCTION) is to be taken unevaluated as an input to the procedure (to be used in some special way as an intensional object—in this case, as a specification of a topic that D70-181 discusses).

This distinction between two types of argument passing is a relatively standard one in some programming languages, frequently referred to as call by value versus call by name. In particular, in the programming language LISP, there are two types of functions (referred to as LAMBDA and NLAMBDA functions), the first of which evaluates all of its arguments and the second of which passes all of its arguments unevaluated to the function (which then specifies in its body which arguments are to be evaluated and what to do with the others).

This ability to pass subordinate expressions literally as intensional objects (to be manipulated in unspecified ways by the operator that gets them) avoids several of the antinomies that have troubled philosophers, such as the nonequivalence of alternative descriptions of the same object in belief contexts. Although belief contexts do not occur in LUNAR, similar problems occur in TRIPSYS, for example, in interpreting the object of the verb "create," where the argument to the verb is essentially a description of a desired object, not an object denoted by the description.

In LUNAR, functions with opaque contexts are also used to define the basic quantification function FOR as well as general purpose counting, averaging, and extremal functions: NUMBER, AVERAGE, MAXIMUM, and MINIMUM. Calls to these functions take the forms:

```
(NUMBER X / ⟨class⟩ : (P X) )
```

"The number of X's in ⟨class⟩ for which (P X) is true."

```
(AVERAGE X / ⟨class⟩ : (P X) ; (F X) )
```

"The average of the values of (F X) over the X's in ⟨class⟩ for which (P X) is true."

```
(AVERAGE X / ⟨class⟩ : (P X) )
```

"The average value of X (a number) over the X's in ⟨class⟩ for which (P X) is true."

```
(MAXIMUM X / ⟨class⟩ : (P X) )
```

"The maximum value of X in the set of X's in ⟨class⟩ for which (P X) is true."

```
(MINIMUM X / ⟨class⟩ : (P X) )
```

"The minimum value of X in the set of X's in ⟨class⟩ for which (P X) is true."

The proposition (P X) in each of these cases has to be taken as an intensional entity rather than a referring expression, since it must be repeatedly evaluated for different values of X.

Opaque context functions are also defined for forming the intensional descriptions of sets and the intensional union of intensionally defined sets:

(SETOF X / ⟨class⟩ : (P X))

"The set of X's in ⟨class⟩ for which (P X) is true."

(UNION X / ⟨class⟩ : (P X) ; (⟨setfn⟩ X))

"The union over the X's in ⟨class⟩ for which (P X) is true of the sets generated by (⟨setfn⟩ X)."

4.8 Restricted Class Quantification

One of the major features of the quantifiers in the LUNAR MRL is the separation of the quantified expression into distinct structural parts: (1) the basic class over which quantification is to range, (2) a set of restrictions on that class, and (3) the main expression being quantified. There are a number of advantages of maintaining these distinctions, one of which is the uniformity of the interpretation procedure over different kinds of noun phrase determiners that it permits. For example, the determiners "some" and "every", when translated into the more customary logical representations, give different main connectives for the expression being quantified. That is, "every man is mortal" becomes (Ax)Man(x)⇒Mortal(x) while "some man is mortal" becomes (Ex)Man(x)&Mortal(x). With the LUNAR format, the choice of determiner affects only the choice of quantifier.

Other advantages to this kind of quantifier are the facilitation of certain kinds of optimization operations on the MRL expressions, and the generation of appropriate antecedents for various anaphoric expressions. Recently, Nash-Webber and Reiter (1977) have pointed out the necessity of making a distinction between the quantification class and the predicated expression if an MRL is to be adequate for handling verb phrase ellipsis and "one"-anaphora.

4.9 Nonstandard Quantifiers

Another advantage of the restricted class quantifier notation is the uniform treatment of a variety of nonstandard quantifiers. For example, LUNAR treats the determiner "the" in a singular noun phrase as a quantifier, selecting the unique object that satisfies its restriction (and complaining if the presupposition that there is a unique such object is not satisfied). This differs from the traditional representation of definite description by means of the iota operator, which constructs a complex designator for a constituent rather than a governing quantifier. In the traditional notation, the sentence "The man I see is mortal," would be represented something like

MORTAL(i(x) : MAN(x) & SEE(I,x)).

In the LUNAR MRL it would be

(FOR THE X / MAN : (SEE I X) ; (MORTAL X)).

Quantifiers such as "many" and "most," whose meaning requires knowledge of the size of the class over which quantification ranges (as well as the size of the class for which the quantified proposition is true) can be adequately handled by this notation since the range of quantification is specifically mentioned. These quantifiers were not implemented in LUNAR, however.

Among the nonstandard quantifiers handled by LUNAR are numerical determiners (both cardinal and ordinal) and comparative determiners. Ordinal quantifiers ("the third X such that P") are handled by a special quantifier (ORDINAL n) that can be used in the ⟨quant⟩ slot of the quantifier form. In general this ordinal quantifier should take another parameter that names the ordering function to be used, or at least require a preferred ordering function to be implied by context. The ordering of the members of the class used by LUNAR is the order of their enumeration by the enumeration function that defines the class (see Section 5.2).

Numerical quantification and comparative quantification are handled with a general facility for applying numeric predicates to a parameter *N* in the FOR function that counts the number of successful members of the range of quantification that have been found. Examples are (GREATER *N* ⟨number⟩), (EQUAL *N* ⟨number⟩), or even (PRIME *N*) (i.e., *N* is a prime number).

The interpretation of general numeric predicates as quantifiers is that if any number *N* satisfying the predicate can be found such that *N* members of the restricted class satisfy the quantified proposition (or successfully complete a quantified command), then the quantified proposition is true (or a quantified command is considered completed). In the implementation, the current value of *N* is tested as each successful member of the restricted class is found, until either the count *N* satisfies the numeric predicate or there are no more members in the class.

The numeric predicate quantifier can be used directly to handle comparative determiners such as "at least" and "more than," and can be used in a negated quantification to handle "at most" and "fewer than." The procedure for testing such quantifiers can return a value as soon as a sufficient number of the class have been found, without necessarily determining the exact number of successful members. The numerical determiner "exactly ⟨*n*⟩" is handled in LUNAR by the generalized counting function NUMBER embedded in an equality statement. (It could also be handled by a conjunction of "at least" and "not more than," but that would not execute as efficiently.)

The LUNAR MRL also permits a generic quantifier GEN, which is assigned to noun phrases with plural inflection and no determiner. Such noun phrases sometimes behave like universal quantification and sometimes like existential quantification. In LUNAR, unless some higher operator indicates that it should be interpreted otherwise, a generic quantifier is evaluated exactly like EVERY.

Examples of types of quantification in LUNAR are

```
(FOR EVERY X / CLASS : (P X) ; (Q X))
```

"Every X in CLASS that satisfies P also satisfies Q."

```
(FOR SOME X / CLASS : (P X) ; (Q X))
```

"Some X in CLASS that satisfies P also satisfies Q."

```
(FOR GEN X / CLASS : (P X) ; (Q X))
```

"A generic X in CLASS that satisfies P will also satisfy Q."

```
(FOR THE X / CLASS : (P X) ; (Q X))
```

"The single X in CLASS that satisfies P also satisfies Q."

```
(FOR (ORDINAL 3) X / CLASS : (P X) ; (Q X))
```

"The third X in CLASS that satisfies P also satisfies Q."

```
(FOR (GREATER N 3) X / CLASS : (P X) ; (Q X))
```

"More than 3 X's in CLASS that satisfy P also satisfy Q."

```
(FOR (EQUAL N 3) X / CLASS : (P X) ; (Q X))
```

"At least 3 X's in CLASS that satisfy P also satisfy Q."

```
(NOT (FOR (EQUAL N 3) X / CLASS : (P X) ; (Q X)))
```

"Fewer than 3 X's in CLASS satisfy P and also satisfy Q."

```
(EQUAL 3 (NUMBER X / CLASS : (P X) : (Q X) ))
```

"Exactly 3 X's in CLASS satisfy P and also satisfy Q."

4.10 Functions and Classes

Another of the attractive features of the LUNAR MRL is the way that quantification over classes, single and multiple valued functions, and the attachment of restrictive modifiers are all handled uniformly, both individually and in combination, by the quantification operators. Specifically, a noun phrase consisting of a function applied to arguments is represented in the same way as a noun phrase whose head is a class over which quantification is to range. For example "The departure time of flight 557 is 3:00" can be represented as

```
(FOR THE X / (DEPARTURE-TIME FLIGHT-557) : T ;
    (EQUAL X 3:00))
```

(where T is the universally true proposition, signifying here that there are no further restrictions on the range of quantification). This permits exactly the same mechanisms for handling the various determiners and modifiers to apply to both functionally determined objects and quantification over classes.

This uniformity of treatment becomes especially significant when the function is not single valued and when the class of values is being quantified over or restricted by additional modifiers as in

```
(FOR EVERY X / (DATALINE S10046 OVERALL SIO2) :
    T ; (PRINTOUT X))
```

and

```
(FOR THE X / (DATALINE S10046 OVERALL SIO2) :
    (REF* X D70-181) ; (PRINTOUT X))
```

where (DATALINE ⟨sample⟩ ⟨phase⟩ ⟨constituent⟩) is the function used in LUNAR to enumerate measurements in its chemical analysis table and (REF* ⟨table entry⟩ ⟨document⟩) is a relation between a measurement and the journal article it was reported in.

4.11 Unanticipated Requests

The structure of the meaning representation language, when coupled with general techniques for semantic interpretation, enable the user to make very explicit requests with a wide range of diversity within a natural framework. As a consequence of the modular composition of MRL expressions, it is possible for the user to combine the basic predicates and functions of the retrieval component in ways that were not specifically anticipated by the system designer. For example, one can make requests such as "List the minerals", "What are the major elements?", "How many minerals are there?", etc. Although these questions might not be sufficiently useful to merit special effort to handle them, they fall out of the mechanism for semantic interpretation in a natural way with no additional effort required. If the system knows how to enumerate the possible samples for one purpose, it can do so for other purposes as well. Furthermore, anything that the system can enumerate, it can count. Thus, the decomposition of the retrieval operations into basic units of quantifications, predicates, and functions provides a very flexible and powerful facility for expressing requests.

5. The Semantics of the Notation

5.1 Procedural Semantics

As mentioned before, the semantic specification of a natural language requires not only a semantic notation for representing the meanings of sentences, but also a specification of the semantics of the notation. As discussed previously, this is done in LUNAR by relating the notation to procedures that can be executed. For each of the predicate names that can be used in specifying semantic representations, LUNAR requires a procedure or subroutine that will determine the truth of the predicate for given values of its arguments. Similarly, for each of the functions that can be used, there must be a procedure that computes the value of that function for given values of its arguments. Likewise, each of the class specifiers for the FOR function requires a subroutine that enumerates the members of the class.

The FOR function itself is also defined by a subroutine, as are the logical operators AND, OR, and NOT, the general counting and averaging functions NUMBER and AVERAGE, and the basic command functions TEST and PRINTOUT. Thus any well-formed expression in the language is a composition of functions that have procedural definitions in the retrieval component and are therefore themselves well-defined procedures capable of execution on the data base. In the LUNAR system, the definition of all of these procedures is done in LISP, and the notation of the meaning representation language is so chosen that its expressions are executable LISP programs. These function definitions and the data base on which they operate constitute the retrieval component of the system.

5.2 Enumeration Functions

One of the engineering features of the LUNAR retrieval component that makes the quantification operators both efficient and versatile is the definition of quantification classes by means of enumeration functions. These are functions that compute one member of the class at a time and can be called repeatedly to obtain successive members. Enumeration functions take an enumeration index argument which is used as a restart pointer to keep track of the state of the enumeration. Whenever FOR calls an enumeration function to obtain a member of a class, it gives it an enumeration index (initially T), and each time the enumeration function returns a value, it also returns a new value of the index to be used as a restart pointer to get the next member. This pointer is frequently an inherent part of the computation and involves negligible overhead to construct. For example, in enumerating integers, the previous integer suffices, while in enumerating members of an existing list, the pointer to the rest of the list already exists.

The enumeration function formulation of the classes used in quantification frees the FOR function from explicit dependence on the structure of the data base; the values returned by the enumeration function may be searched for in tables, computed dynamically, or merely successively accessed from a precomputed list. Enumeration functions also enable the quantifiers to operate on potentially infinite classes and on classes of objects that do not necessarily exist prior to the decision of the quantifier to enumerate them. For example, in an expression such as

```
(FOR SOME X / INTEGER : (LESSP X 10) ; (PRIME X) )
```

("some integer less than 10 is a prime"), a general enumeration procedure for integers can be used to construct successive integers by addition, without having to assume that all the integers of interest exist in the computer's memory ahead of time. Thus, the treatment of this kind of quantification fits naturally within LUNAR's general quantification mechanism without having to be treated as a special case.

In the grammar information system application, an enumeration function for paths computes representations for paths through the grammar, so that paths can be talked about even though there are no explicit entities in the internal grammar representation that correspond to paths. (See the discussion on "smart" quantifiers below for a further discussion of the problems of quantifying over such entities.)

An enumeration function can indicate termination of the class in one of two ways: either by returning NIL, indicating that there are no more members, or by returning a value with a NIL restart pointer, indicating that the current value is the last one. This latter can save one extra call to the enumeration function if the information is available at the time the last value is returned (e.g., for single valued functions). This avoids what would otherwise be an inefficiency in treating multiple- and single-valued functions the same way.

In LUNAR, a general purpose enumeration function SEQ can be used to enumerate any precomputed list, and a similar function SEQL can be used to enumerate singletons. For example,

```
(FOR EVERY X1 / (SEQ TYPECS) : T ; (PRINTOUT X1))
```

is an expression that will printout the sample numbers for all of the samples that are type C rocks.

Functionally determined objects and classes, as well as fixed classes, are implemented as enumeration functions, taking an enumeration index as well as their other arguments and computing successive members of their class one at a time. In particular, intensional operators such as AVERAGE, NUMBER, SETOF, and UNION are defined as enumeration functions and also use enumeration functions for their class arguments. Thus quantification over classes, computation of single-valued functions, and quantification over the values of multiple-valued functions are all handled uniformly, without special distinctions having to be made.

5.3 Quantified Commands

As mentioned earlier, both propositions and commands can be quantified. Thus one can issue commands such as

```
(FOR (EQ N 5) X / SAMPLES : (CONTAIN X SIO2) ; (PRINTOUT X))
```

("Print out five samples that contain silicon"). The basic commands in such expressions are to be iterated according to the specifications of the quantifier. However, it is possible for such commands to fail due to a violation of presuppositions or of necessary conditons. For example, in the above case, there might not be as many as five samples that contain silicon. In order for the system to be aware of such cases, each command in the system is defined to return a value that is non-null if the command has been successfully executed and NIL otherwise. Given this convention, the FOR operator will automatically return T if such an iterated command has been successfully completed and NIL otherwise.

There are other variations of this technique that could be useful but were not implemented in LUNAR, such as returning comments when a command failed indicating the kind of failure. In LUNAR, such comments were sometimes printed to the user directly by the procedure that failed, but the system itself had no opportunity to "see" those comments and take some action of its own in response to them (such as trying some other way to achieve the same end).

In LUNAR, interpretations of commands are given directly to the retrieval component for evaluation, although in a more intelligent system, as in humans, the decision to carry out a command once it is understood would not necessa:ily automatically follow.

6. Semantic Interpretation

Having now specified the notation in which the meanings of English sentences are to be represented and specifying the meanings of expressions in that notation, we are now left with the specification of the process whereby meanings are assigned to sentences. This process is referred to as semantic interpretation, and in LUNAR it is driven by a set of formal semantic interpretation rules. For example, the interpretation of the sentence "S10046 contains silicon," to which the parser would assign the syntactic structure

```
S DCL
  NP   NPR S10046
  AUX  TNS PRESENT
  VP   V CONTAIN
       NP NPR SILICON
```

is determined by a rule that applies to a sentence when the subject is a sample, the object is a chemical element, oxide, or isotope, and the verb is "have" or "contain." This rule specifies that such a sentence is to be interpreted as an instance of the schema (CONTAIN x y), where x is to be replaced by the interpretation of the subject noun phrase of the sentence, and y is to be replaced by the interpretation of the object.

This information about conditions on possible arguments and substitutions of subordinate interpretations into "slots" in the schema is represented in LUNAR by means of the pattern → action rule

```
[S:CONTAIN
     (S.NP (MEM 1 SAMPLE))
     (S.V (OR (EQU 1 HAVE)
              (EQU 1 CONTAIN))
     (S.OBJ (MEM 1 (ELEMENT OXIDE ISOTOPE)))
     →(QUOTE (CONTAIN (# 1 1) (# 3 1))) ].
```

The name of the rule is S:CONTAIN. The left-hand side, or pattern part, of the rule consists of three templates that match fragments of syntactic structure. The first template requires that the sentence being interpreted have a subject noun phrase that is a member of the semantic class SAMPLE; the second requires that the verb be either "have" or "contain;" and the third requires a direct object that is either a chemical element, an oxide, or an isotope.

The right-hand side, or action part, of the rule follows the right arrow and specifies that the interpretation of this node is to be formed by inserting the interpretations of the subject and object constituents into the schema (CONTAIN (# 1 1) (# 3 1)), where the expressions (# m n) mark the "slots" in the schema where subordinate interpretation are to be inserted. The detailed structure of such rules is described in Section 6.3. (Note that the predicate CONTAIN is the name of a procedure in the retrieval component, and it is only by the "accident" of mnemonic design that its name happens to be the same as the English word "contain" in the sentence that we have interpreted.)

The process of semantic interpretation can conveniently be thought of as a process that applies to parse trees produced by a parser to assign semantic interpretations to nodes in the tree. In LUNAR and the other systems above, except for TRIPSYS, this is how the interpretations are produced. (In TRIPSYS, they are produced directly by the parser without an intermediate syntax tree representation.) The basic interpretation process is a recursive procedure that assigns an interpretation to a node of the tree as a function of its syntactic structure and the interpretations of its constituents.

The interpretations of complex constituents are thus built up modularly by a recursive process that determines the interpretation of a node by inserting the interpretations of certain constituent nodes into open slots in a schema. The schema to be used is determined by rules that look at a limited portion of the tree. At the bottom level of the tree (i.e., the leaves of the tree), the interpretation schemata are literal representations without open slots, specifying the appropriate elementary interpretations of basic atomic constituents (e.g., proper names).

In LUNAR, the semantic interpretation procedure is implemented in such a way that the interpretation of nodes can be initiated in any order. If the interpretation of a node requires the interpretation of a constituent that has not yet been interpreted, then the interpretation of that constituent is performed before that of the higher node is completed. Thus, it is possible to perform the entire semantic interpretation by calling for the interpretation of the top node (the sentence as a whole). This is the normal mode in which the interpreter is operated in LUNAR. I will discuss later (Sections 11.3 and 11.4) some experiments in which this mechanism is used for "bottom-up" interpretation.

6.1 Complications Due to Quantifiers

In the above example, the interpretation of the sentence is obtained by inserting the interpretations of the proper noun phrases "S10046" and "silicon" (in LUNAR these are "S10046" and "SIO2," respectively) into the open slots of the right-hand side schema to obtain

(CONTAIN S10046 SIO2).

When faced with the possibility of a quantified noun phrase, however, the problem becomes somewhat more complex. If the initial sentence were "Every sample contains silicon," then one would like to produce the interpretation

(FOR EVERY X / SAMPLE ; (CONTAIN X SIO2)).

That is, one would like to create a variable to fill the "container" slot of the schema for the main verb, and then generate a quantifier governing that variable to be attached above the predicate CONTAIN. As we shall see, the LUNAR semantic interpretation system specifically provides for the generation and appropriate attachment of such quantifiers.

6.2 Problems with an Alternative Approach

Because of the complications discussed above, one might ask whether there is some other way to handle quantification without generating quantifiers that are extracted from their noun phrase and attached as dominant operators governing the clause in which the original noun phrase was embedded. One might, instead, attempt to interpret the quantified noun phrase as some kind of a set that the verb of the clause takes as its argument, and require the definition of the verb to include the iteration of its basic predicate over the members of the class. For example, one might want a representation for the above example something like

(CONTAIN (SET X / SAMPLE : T) SIO2)

with the predicate CONTAIN defined to check whether its first argument is a set and if so, check each of the members of that set.

However, if one were to take this approach, some way would be needed to distinguish giving CONTAIN a set argument over which it should do universal quantification from one in which it should do existential quantification. One would similarly have to be able to give it arguments for the various nonstandard quantifiers discussed above, such as numerical quantifiers and quantifiers like "most." Moreover, the same thing would have to be done separately for the second argument to

CONTAIN as well as the first (i.e., the chemical element as well as the sample), and one would have to make sure that all combinations of quantifiers in the two argument positions worked correctly. Essentially one would have to duplicate the entire quantificational mechanism discussed above as part of the defining procedure for the meaning of the predicate CONTAIN. Moreover, one would then have to duplicate this code separately for each other predicate and command in the system. Even if one managed to share most of the code by packaging it as subroutines, this is still an inelegant way of handling the problem.

Even if one went to the trouble just outlined, there are still logical inadequacies, since there is no way with the proposed method to specify the differences in meaning that correspond to the different relative scopes of two quantifiers (e.g., "Every sample contains some element" versus "There is some element that every sample contains"). Likewise, there is no mechanism to indicate the relative scopes of quantifiers and sentential operators such as negation ("Not every sample contains silicon" versus "Every sample contains no silicon"). It appears, therefore, that treating quantifiers effectively as higher operators is essential to correct interpretation in general.

6.3 The Structure of Semantic Rules

As discussed above, in determining the meaning of a construction, two types of information are used: syntactic information about sentence construction and semantic information about constituents. For example, in interpreting the above example, it is both the syntactic structure of the sentence (subject = S10046; verb = "contain;" object = silicon) plus the semantic fact that S10046 is a sample and silicon is a chemical element that determine the interpretation. Syntactic information about a construction is tested by matching tree fragments such as those indicated below against the mode being interpreted:

S.NP	= S NP (1)	(subject of a sentence)
S.V	= S VP V (1)	(main verb of a sentence)
S.OBJ	= S VP NP (1)	(direct object of a sentence)
S.PP	= S VP PP PREP (1)	(preposition and object
	NP (2)	modifying a verb phrase)
NP.ADJ	= NP ADJ (2)	(adjective modifying a noun phrase).

Fragment S.NP matches a sentence if it has a subject and also associates the number 1 with the subject noun phrase. S.PP matches a sentence that contains a prepositional phrase modifying the verb phrase and associates the numbers 1 and 2 with the preposition and its object, respectively.

The numbered nodes can be referred to in the left-hand sides of rules for checking semantic conditions, and they are used in the right-hand sides for specifying the interpretation of the construction. These tree structure fragments can be named mnemonically as above for readability.

The basic element of the left-hand side of a rule is a *template* consisting of tree fragments plus additional semantic conditions on the numbered nodes of the fragment. For example, the template (S.NP (MEM 1 SAMPLE)) matches a sentence if its subject is semantically marked as a sample. The pattern part of a rule consists of a sequence of templates, and the action of the rule specifies how the interpretation of the sentence is to be constructed from the interpretations of the nodes that match the numbered nodes of the templates.

Occasionally, some of the elements that are required to construct an interpretation may be found in one of several alternative places in a construction. For example, the constituent to be measured in an analysis can occur either as a prenomial adjective ("a silicon analysis") or as a post-nominal prepositional phrase ("an analysis of silicon"). To handle this case, basic templates corresponding to the alternative ways the necessary element can be found can be grouped together with an OR operator to form a disjunctive template that is satisfied if any of its disjunct templates are. For example,

```
(OR  (NP.ADJ (MEM 2 ELEMENT))
     (NP.PP (AND (EQU 1 OF)
                 (MEM 2 ELEMENT)))).
```

Also occasionally, two rules will be distinguished by the fact that one applies when a given constituent is present and the other will require it to be absent. In order to write the second rule so that it will not match in circumstances where it is not intended, a basic template can be embedded in a negation operator NOT to produce a negated template that is satisfied if its embedded template fails to match and is not satisfied when its embedded template succeeds. For example,

```
(NOT (NP.ADJ (EQU 2 MODAL))).
```

In general, the left-hand side of a rule consists of a sequence of templates (basic, disjunctive, or negated).

6.3.1 Right-Hand Sides

The right-hand sides (or actions) of semantic rules are schemata into which the interpretations of embedded constituents are inserted before the resulting form is evaluated to give a semantic interpretation. The places, or "slots," in the right-hand sides where subordinate interpretations are to be inserted are indicated by expressions called REFs, which begin with the atom # and contain one or two numbers and an optional "TYPEFLAG." The numbers indicate the node in the tree whose interpretation is to be inserted by naming first the sequence number of a template of the rule, and then the number of the corresponding node in the tree fragment of that template. Thus the reference (# 2 1) represents the interpretation of the node that matches node 1 of the 2nd template of the rule. In addition, the single number 0 can also be used to reference the current node, as in (# 0 TYPEFLAG).

The TYPEFLAG element, if present, indicates how the subordinate node is to be interpreted. For example, in LUNAR there is a distinction between interpreting a node normally and interpreting it as a topic description. Thus (# 0 TOPIC) represents the interpretation of the current node as a topic description. There are a variety of types of interpretation used for various purposes in the rules of the system. The absence of a specific TYPEFLAG in a REF indicates that the interpretation is to be done in the normal mode for the type of node that it matches.

6.3.2 Right-Hand Side Evaluation

In many cases, the semantic interpretation to be attached to a node can be constructed by merely inserting the appropriate constituent interpretations into the open slots in a fixed schema. However, occasionally, more than this is required and some procedure needs to be executed to modify or transform the resulting instantiated schema. To provide for this, the semantic interpreter treats right-hand sides of rules as expressions to be evaluated to determine the appropriate interpretation. For rules in which the desired final form can be given literally, the right-hand side schema is embedded in the operator QUOTE which simply returns its argument unchanged. This is the case in the example above. In special cases, right-hand side operators can do fairly complex things, such as searching a discourse directory for antecedents for anaphoric expressions and computing intensional unions of sets. In the usual case, however, the operator is either QUOTE or one of the two operators PRED and QUANT that handle quantifier passing (discussed below).

6.4 Relationship of Rules to Syntax

In many programming languages and some attempts to specify natural language semantics, semantic rules are paired directly with syntactic phrase structure rules so that a single compact pairing specifies both the syntactic structure of a constituent and its interpretation. This type of specification is clean and straightforward and works well for artificial languages that can be defined by context-free or almost context-free grammars. For interpreting natural language sentences, whose structure is less isomorphic to the kind of logical meaning representation that one would like to derive, it is less convenient, although not impossible. Specifically, with the more complex grammars for natural language, e.g., ATN's and transformational grammars, the simple notion of a syntactic rule with which to pair a semantic rule becomes less clear. Consequently, the rules in the LUNAR system are not paired with the syntactic rules, nor are they constrained to look only at the immediate constituents of a phrase. In general they can look arbitrarily far down into the phrase they are interpreting, picking up interpretations of subordinate constituents at any level, and looking at various syntactic aspects of the structure they are interpreting, as well as the semantic interpretations of constituents. The rules are invoked not by virtue of applying a given syntactic rule, but by means of rule indexing strategies described below.

6.5 Organization of the Semantic Interpreter

The overall operation of the semantic interpreter is as follows: A top level routine calls the recursive function INTERP looking at the top level of the parse tree. Thereafter, INTERP attempts to match semantic rules against the specified node of the tree, and the right-hand sides of matching rules specify the interpretation to be given to the node. The possibility of semantic ambiguity is recognized, and therefore the routine INTERP produces a list of possible interpretations (usually a singleton, however). Each interpretation consists of two parts: a node interpretation (called the SEM of the node) and a quantifier "collar" (called the QUANT of the node). The QUANT is a schema for higher operators (such as quantification) that is to dominate any interpretation in which the SEM is inserted (used for quantifier passing—see Section 6.7). Thus the result of a call to INTERP for a given node P is a list of SEM-QUANT pairs, one for each possible interpretation of the node.

6.5.1 Context-Dependent Interpretation

The function INTERP takes two arguments—the construction to be interpreted and a TYPEFLAG that indicates how to interpret it. The TYPEFLAG mechanism is intended to allow a constituent to be interpreted differently depending on the higher level structure within which it is embedded. The TYPEFLAG permits a higher level schema to pass down information to indicate how it wants a constituent interpreted. For example, some verbs can specify that they want a noun phrase interpreted as a set rather than as a quantification over individuals. The TYPEFLAG mechanisms is also used to control the successive phases of interpretation of noun phrases and clauses (discussed below).

When interpreting a node, INTERP first calls a function HEAD to determine the head of the construction and then calls a function RULES to determine the list of semantic rules to be used (which depends, in general, on the type of node, its head word, and the value of TYPEFLAG). It then dispatches control to a routine MATCHER to try to match the rules. If no interpretations are found, then, depending on the TYPEFLAG and various mode settings, INTERP either returns a default interpretation T, goes into a break with a comment that the node is uninterpretable (permitting a systems programmer to debug rules), or returns NIL indicating that the node has no interpretations for the indicated TYPEFLAG.

6.5.2 Phased Interpretation

In general, there are two types of constituents in a sentence that receive interpretations—clauses and noun phrases. The former receive interpretations that are usually predications or commands, while the latter are usually designators. The interpretation of these two different kinds of phrase are slightly different, but also remarkably similar. In each case there is a governing "head" word; the verb in the case of a clause, and the head noun in the case of the noun phrase. The interpretation of a phrase is principally determined by the head word (noun or verb) of the construction. However, there are also other parts of a construction that determine aspects of its interpretation independent of the head word. These in turn break down into two further classes: (1) modifying phrases (which themselves have dominating head words) that augment or alter meaning of the head, and (2) function words that determine governing operators of the interpretation that are independent of the head word and its modifiers. In the case of clauses, these latter include the interpretation of tense and aspect and various qualifying operators such as negative particles. In the case of noun phrases, these include the interpretation of articles and quantifiers and the inflected case and number of the head noun.

As a consequence of these distinctions, the semantic interpretation of a construction generally consists of three kinds of operations: determining any governing operators that are independent of the head word, determining the basic interpretation of the head, and interpreting any modifiers that may be present. In LUNAR, these three kinds of interpretation are governed by three different classes of rules that operate in three phases.

The phases are controlled by the rules themselves by using multiple calls to the interpreter with different TYPEFLAGS.

The above description is not the only way such phasing could be achieved. For example, it would be possible to gain the same phasing of interpretation by virtue of the structures assigned to the input by the parser (see Section 11.2) or by embedding the phasing in the control structure of the interpreter. In the original flight schedules and grammar information implementations, this phasing was embedded in the control structure of the interpreter. Placing the phasing under the control of the rules themselves in LUNAR provided more flexibility. In TRIPSYS, the equivalent of such phasing is integrated, along with the semantic interpretation, into the parsing process.

In general, the interpretation of a construction is initially called for with TYPEFLAG NIL. This first interpretation may in turn involve successive calls for interpretation of the same node with other TYPEFLAGs to obtain subsequent phases of interpretation. For example, clauses are initially interpreted with TYPEFLAG NIL, and the rules invoked are a general set of rules called PRERULES that look for negative articles, tense marking, conjunctions, etc., to determine any governing operators that should surround the interpretation of the verb. Whichever of these rules matches will then call for another interpretation of the same construction with an appropriate TYPEFLAG. The basic interpretation of the verb is done by a call with TYPEFLAG SRULES, which invokes a set of rules stored on the property list of the verb (or reachable from the entry for that verb by chaining up a generalization hierarchy). For example, in interpreting the sentence "S10046 doesn't contain silicon", the initial PRERULE PR-NEG matches with a right-hand side

(PRED (NOT (# 0 SRULES))).

The SRULE S:CONTAIN discussed above then matches, producing eventually (CONTAIN S10046 SIO2), which is then embedded in the PR-NEG schema to produce the final interpretation

(NOT (CONTAIN S10046 SIO2)).

Ordinary noun phrases are usually interpreted by an initial phase that interprets the determiner and number, a second phase that interprets the head noun and any arguments that it may take (i.e., as a function), and a third phase that interprets other adjectival and prepositional phrase modifiers and relative clauses.

6.5.3 Proper Nouns and Mass Terms

In addition to the rules discussed above for ordinary noun phrases, there are two special classes of noun phrases—proper nouns and mass terms—that have their own rules. Proper nouns are the direct names of individuals in the data base. Their identifiers in the data base, which are not necessarily identical to their normal English orthography, are indicated in the dictionary entry for the English form. Mass terms are the names of substances like silicon and hydrogen. Proper nouns are represented in the LUNAR syntactic representations as special cases of noun phrases by a rule equivalent to NP → NPR, while mass terms are represented as ordinary noun phrases with determiner NIL and number SG.

In general, the interpretation of mass terms requires a special treatment of quantifiers, similar to but different from the ordinary quantifiers that deal with count nouns (e.g., "some silicon" means an amount of stuff, while "some sample" means an individual sample). In the LUNAR system, however, mass terms are used only in a few specialized senses in which they are almost equivalent to proper nouns naming a substance.

6.6 Organization of Rules

As mentioned above, the semantic rules for interpreting sentences are usually governed by the verb of the sentence. That is, out of the entire set of semantic rules, only a relatively small number of them can possibly apply to a given sentence because of the verb mentioned in the rule. Similarly, the rules that interpret noun phrases are governed by the head noun of the noun phrase. For this reason, most semantic rules in LUNAR are indexed according to the heads of the constructions to which they could apply, and recorded in the dictionary entry for the head words. Specifically, associated with each verb is a set of "SRULES" for interpreting that verb in various contexts, and associated with each noun is a set of "NRULES" for interpreting various occurrences of that noun. In addition, associated with each noun are a set of "RRULES" for interpreting various restrictive modifiers that may be applied to that noun. Each rule essentially characterizes a syntactic/semantic environment in which a word can occur, and specifies its interpretation in that environment. The templates of a rule thus describe the necessary and sufficient constituents and semantic restrictions for a word to be meaningful.

In addition to indexing rules directly in the dictionary entry for a given word, certain rules that apply generally to a class of words are indexed in an inheritance hierarchy (frequently called an "is-a" hierarchy in semantic network notations) so that they can be recorded once at the appropriate level of generality. Specifically, each work in the dictionary has a property called MARKERS which contains a list of classes of which it is a member (or subclass), i.e., classes with which this word has an "is-a" relationship. Each of these classes also has a dictionary entry that may contain SRULES, NRULES, and RRULES. The set of rules used by the interpreter for any given phrase is obtained by scanning up these chains of inheritance and gathering up the rules that are found. These accesses are quite shallow in LUNAR but would be used more heavily in a less limited topic domain.

In situations in which the set of rules does not depend on the head of the construction, the rules to be used are taken from a global list determined by the value of TYPEFLAG and the type of the constituent being interpreted. For example, in interpreting the determiner structure of a noun phrase, a global list of DRULES is used.

6.6.1 Rule Trees

Whether indexed by the head words of constructions or taken from global lists, rules to be tried are organized into a tree structure that can make rule matching conditional on the success or failure of previous rules. A rule tree specifies the order in which rules are to be tried and after each rule indicates whether a different tree of rules is to be tried next, depending on the success or failure of previous rules. The format for a rule tree is basically a list of rules (or rule groups—see multiple matches below) in the order they are to be tried. However, after any given element in this list, a new rule tree can be inserted to be used if any of the rules preceding it have succeeded. If no rules preceding it have succeeded, then the inserted tree is skipped and rules continue to be taken from the rules that follow it in the list. For example, the tree (R1 R2 (R4 R5) R3 R4 R5) indicates that R1 and R2 are to be tried in that order and if either of them succeed, the subsequent rules to be tried are R4 and R5. If neither R1 nor R2 succeed, then the remaining list R3, R4, R5 is to be tried next. This example illustrates how a rule tree can be used to skip around rules that are to be omitted if previous rules have succeeded.

The most usual cases of rule trees in LUNAR are simple lists (i.e., no branching in the tree), and lists of rules with inserted empty trees (i.e., the empty list NIL) serving as "barriers" to stop the attempted matching of rules once a successful rule has been found.

6.6.2 Multiple Matches

Since the templates of a rule may match a node in several ways, and since several rules may simultaneously match a single node, it is necessary to indicate how the interpretation of a node is to be constructed in such a case. To provide this information, the lists of rules at each level of a rule tree can be organized into groups, with each group indicating how (or whether) simultaneous matches by different rules are to be combined. The format of a rule group is a list of rules (or other groups) preceded by an operator specifying the mode for combining simultaneous matches. Outside the scopes of rule groups, the mode to be used is specified by a default value determined by TYPEFLAG and the type of node being interpreted. Possible modes are AND (which combines multiple matches with an AND, i.e., treats multiple matches as finding different parts of a single conjoined meaning), OR (which combines multiple matches with an OR), SPLIT (which keeps multiple matches separate as semantic ambiguities), and FAIL (which prohibits multiple matches, i.e., complains if it finds any).

To illustrate the behavior of rule groups in rule trees, a rule list of the form (A B NIL C (OR D E)) with default mode AND indicates that if either of the rules A or B is successful, then no further matches are tried (NIL is a barrier); otherwise, rules C, D, and E are tried. If both D and E match, then the results are OR'ed together, and if C matches together with D or E or both, it is AND'ed to the results of the OR group.

The modes (AND, OR, SPLIT, and FAIL) also apply to multiple matches of a single rule. A rule may either specify the mode for multiple matches as its first element prior to the list of templates, or else it will be governed by the rule group or default mode setting at the time it is matched.

6.7 The Generation of Quantifiers

As mentioned above, the LUNAR interpretation system specifically provides for the generation and appropriate attachment of quantifiers governing the interpretations it produces. Central to this capability is the division of the interpretation of a constituent into two parts: a SEM that is to be inserted into the appropriate slot of the schema for some higher constituent, and a QUANT that serves as a "collar" of higher operators that is to be passed up to some higher level of the tree (around which the collar will be "worn"). A quantifier to be attached to some higher constituent is represented as a schema, which itself contains a slot into which the interpretation of that higher constituent is to be inserted. This slot (the "hole" in the collar) is indicated by a marker DLT.

In the unquantified example sentence considered in Section 6.1, the SEM of the subject noun phrase is simply S10046, and the QUANT is the "empty" collar DLT. The quantifier schema in the second example would be represented as

(FOR EVERY X / SAMPLE ; DLT).

6.7.1 *Steps in Interpretation*

The general procedure for interpreting a construction is

a) Match an interpretation rule against the construction, subject to the control of the rule tree.

b) If it matches, then determine from the right-hand side of the rule the set of constituent nodes that need to be interpreted.

c) Call for the interpretation of all of the constituents required, associate their SEMs with the slots in the schema that they are to fill, and gather up all of the QUANTs that are generated by those interpretations. Call a function SORTQUANT to determine the order in which those quantifiers (if there are several) should be nested.

d) Depending on an operator in the right-hand side of rule, either attach the quantifiers so generated around the outside of the current schema, or pass them further up the tree as the QUANT of the resulting interpretation.

e) If multiple matches are to be combined with an AND or OR, it is their SEMs that are so combined. Their QUANTs are nested one inside the other to produce the QUANT of the result.

6.7.2 *Quantifier Passing Operators*

There are three principal operators for use in the right-hand sides of rules to determine the behavior of quantifier passing up the tree. These are the operators PRED, QUOTE, and QUANT. The first indicates that the schema it contains is a predication that will accept quantifiers from below; it causes any quantifiers that arise from constituent interpretations to be attached around the current schema to become part of the resulting SEM. The QUANT associated with such an interpretation will be the empty QUANT DLT. The operator QUANT, on the other hand, indicates that the schema it contains is itself a quantifier schema, and that the result of its instantiation is to be passed up the tree (together with other quantifiers that may have resulted from constituent interpretations) as the QUANT of the interpretation. The SEM associated with such an interpretation is the variable name that is being governed by the quantifier. The operator QUOTE is used around a schema that is transparent to quantifier passing, so that any quantifiers that accumulate from constituent interpretations are simply aggregated together and passed on up the tree as the QUANT of the interpretation. The SEM of such an interpretation is simply the instantiated schema inside the QUOTE.

In the LUNAR implementation, a function SEMSUB, which substitutes the SEMs of lower interpretations into the right-hand sides of rules, maintains a variable QUANT to accumulate the nesting of quantifiers returned from the lower interpretations. Then, after making the substitutions, the right-hand side of the rule is evaluated to determine the SEM-QUANT pair to be returned. The result of the evaluation is the desired SEM of the pair, and the value of QUANT (which may have been changed as a side effect of the evaluation) is the QUANT of the pair. The operators PRED and QUANT in the right-hand sides of rules manipulate the variable QUANT to grab and insert quantifiers.

7. Problems of Interpretation

7.1 The Order of Quantifier Nesting

In the general quantification schema

(FOR ⟨quant⟩ X / ⟨class⟩ : (p X) ; (q X))

both the expressions (p X) and (q X) can themselves be quantified expressions. Sentences containing several quantified noun phrases result in expressions with a nesting of quantifiers dominating the interpretation of the main clause. For example, the sentence "Every sample contains some element" has a representation

```
(FOR EVERY X / SAMPLE ;
   (FOR SOME Y / ELEMENT ;
      (CONTAIN X Y) ) ).
```

Alternative interpretations of a sentence corresponding to different orderings of the quantifiers correspond to different relative nestings of the quantifier operations. For example, the above sentence has an unlikely interpretation in which there is a particular element that is contained in every sample. The representation of this interpretation is

```
(FOR SOME Y / ELEMENT ;
   (FOR EVERY X / SAMPLE ;
      (CONTAIN X Y) ) ).
```

Thus, in interpreting a sentence, it is necessary to decide the appropriate order of nesting of quantifiers to be used. In general, this ordering is the left-to-right order of occurrence of the quantifiers in the sentence, but this is not universally so (for example, when a function is applied to a quantified noun phrase—see functional nesting below). In situations where the order of quantifiers is not otherwise determined, LUNAR assumes the left-to-right order of occurrence in the sentence.

7.2 Interaction of Negations with Quantifiers

The construction of an interpretation system that will handle sentences containing single instances of a quantification or simple negation without quantification is not difficult. What is difficult is to make it correctly handle sentences containing arbitrary combinations of quantifiers and negatives. The interpretation mechanism of LUNAR handles such constructions fairly well. Consider the sentence "Every sample does not contain silicon." This sentence is potentially ambiguous between two interpretations:

(NOT (FOR EVERY X / SAMPLE ; (CONTAIN X SIO2)))

and

(FOR EVERY X / SAMPLE ; (NOT (CONTAIN X SIO2))).

The difference lies in the relative scopes of the quantifer and the negative.

One interpretation of the above sentence is handled in LUNAR by the interaction of the rules already presented. The interpretation of the PRE-RULE PR-NEG, discussed in Section 6.5.2, has the right-hand side (PRED (NOT (# 0 SRULES))), whose governing operator indicates that it grabs quantifiers from below. The interpretation of the noun phrase "every sample" produces the quantifier "collar":

(FOR EVERY X / SAMPLE : T ; DLT)

which is passed up as the QUANT together with the SEM X. The right-hand side of S:CONTAIN is embedded in the operator QUOTE, which is transparent to quantifiers, producing the SEM (CONTAIN X SIO2) and passing on the same QUANT. The top level rule PR-NEG now executes its instantiated right-hand side:

(PRED (NOT (CONTAIN X SIO2)))

which grabs the quantifier to produce the interpretation:

(FOR EVERY X / SAMPLE : T ; (NOT CONTAIN X SIO2))).

The alternative interpretation of the above sentence can be obtained by an alternative PRERULE for sentential negatives whose right-hand side is

(BUILDQ (NOT #) (PRED (# 0 SRULES)))

where BUILDQ is an operator whose first argument is a literal schema into which it inserts the values of its remaining arguments. In this case, the PRED expression produces

(FOR EVERY X / SAMPLE : T ; (CONTAIN X SIO2))

and the BUILDQ produces

(NOT (FOR EVERY X / SAMPLE : T ; (CONTAIN X SIO2))).

If these two negative rules both existed in the list PRERULES, then the LUNAR interpreter when interpreting a negative sentence would find them both and would produce both interpretations. In the case where no quantifier is returned by the subordinate SRULES interpretation, then both rules would produce the same interpretation and the duplicate could be eliminated. In the case where a quantifier is returned, then the two interpretations would be different and a genuine ambiguity would have been found, resulting in a request by the system to the user to indicate which of the two interpretations he intended.

However, if one decides to legislate that only one of the two possible scope choices should be perceived by the system, then only the corresponding rule for negation should be included in the PRERULES list. This is the choice that was taken in the demonstration LUNAR system. Since the interpretation of the negative operator outside the scope of the quantifier can be unambiguously expressed using locutions such as "Not every sample contains silicon," LUNAR's rules treat sentential negation as falling inside any quantifiers (as expressed by the PR-NEG rule discussed previously). Rules for interpreting determiners such as "not every" can easily be written to produce quantifier expressions such as

(NOT (FOR EVERY X / ⟨class⟩ ; DLT))

to give interpretations in which the negative operator is outermost.

7.3 Functional Nesting and Quantifier Reversal

As previously mentioned, an interesting example of quantifier nesting occurs when an argument to a function is quantified. As an example, consider the flight schedules request, "List the departure times from Boston of every American Airlines flight that goes from Boston to Chicago." This sentence has a bizarre interpretation in which there is one time at which every American Airlines flight from Boston to Chicago departs. However, the normal interpretation requires taking the subordinate quantifier "every flight" and raising it above the quantifier of the higher noun phrase "the departure time." Such nesting of quantifiers is required when the range of quantification of one of them (in this case, the departure times) contains a variable governed by the other (in this case, the flights).

In the logical representation of the meaning of such sentences, the higher quantifier must be the one that governs the variable on which the other depends. This logical dependency is exactly the reversal of the "syntactic dependency" in the parse tree, where the argument to the function is contained within (i.e., "dependent" on) the phrase the function heads. The LUNAR system facility for interpreting such constructions automatically gets the preferred interpretation, since the quantifiers from subordinate constituents are accumulated and nested before the quantifier for a given noun phrase is inserted into the quantifier collar.

To illustrate the process in detail, consider the interpretation of the above example. In the processing of the constituents of the noun phrase whose head is "departure time," the quantifier

```
(FOR EVERY X2 / FLIGHT : (EQUAL (OWNER X2) AMERICAN) ;
                         DLT)
```

is returned from the interpretation of the "flight" noun phrase (which gets the SEM X2). The temporary QUANT accumulator in the function SEMSUB (discussed in Section 6.7), at this point contains the single "empty" quantifier collar DLT. This is now modified by substituting the returned quantifier for the DLT, resulting in the QUANT accumulator now containing the returned quantifier

```
(FOR EVERY X2 / FLIGHT : (EQUAL (OWNER X2) AMERICAN) ;
                         DLT)
```

(with its DLT now marking the "hole" in the collar).

When all of the subordinate constituents have been interpreted, and their SEM's have been inserted into the right-hand side schema of the rule interpreting the "departure time" noun phrase, the resulting instantiated schema will be

```
(QUANT (FOR THE X1 / (DTIME X2 BOSTON) : T ; DLT) ).
```

This is then evaluated, again resulting in the DLT in the temporary QUANT accumulator being replaced with this new quantifier (thus inserting the definite quantification THE inside the scope of the universal quantifier EVERY that is already there). The result of this interpretation is to return the SEM-QUANT pair consiting of the SEM X1 and the QUANT

```
(FOR EVERY X2 / FLIGHT : (EQUAL (OWNER X2) AMERICAN) ;
    (FOR THE X1 / (DTIME X2 BOSTON) : T ; DLT )).
```

The right-hand side for the next higher rule (the one that interprets the command "list x") contains a PRED operator, so that when its instantiated schema

```
(PRED (PRINTOUT X1))
```

is executed, it will grab the quantifier collar from below to produce the interpretation

```
(FOR EVERY X2 / FLIGHT : (EQUAL (OWNER X2) AMERICAN) ;
    (FOR THE X1 / (DTIME X2 BOSTON) : T ;
        (PRINTOUT X1) )).
```

7.4 Relative Clauses

One of the features of the LUNAR system that makes it relatively powerful in the range of questions it can handle is its general treatment of relative clause modifiers. This gives it a natural ability to handle many questions that would be awkward or impossible to pose to many data management systems. Relative clauses permit arbitrary predicate restrictions to be imposed on the range of quantification of some iterative search. The way in which relative clauses are interpreted is quite simple within LUNAR's general semantic interpretation framework. It is done by a general RRULE R:REL, which is implicitly included in the RRULES for any noun phrase.

The rule R:REL will match a noun phrase if it finds a relative clause structure modifying the phrase. On each such relative clause, it will execute a function RELTAG that will find the node in the relative clause corresponding to the relative pronoun ("which" or "that"), and will mark this found node with the same variable X that is being used for the noun phrase that the relative clause modifies. This pronoun will then behave as if it had already been interpreted and assigned that variable as its SEM. The semantic interpreter will then be called on the relative clause node, just like any other sentence being interpreted, and the result will be a predicate with a free occurrence of the variable X. This resulting predicate is then taken, together with any other RRULE predicates obtained from adjectival and prepositional phrase modifiers, to form the restriction on the range of quantification of the modified noun phrase.

One consequence of a relative clause being interpreted as a subordinate S node (in fact, a consequence of any subordinate S node interpretation) is that, since the PRERULES used in interpreting the subordinate S node all have PRED operators in their right-hand sides, any quantifiers produced by noun phrases inside the relative clause will be grabbed by the relative clause itself and not passed up to the main clause. This rules out interpretations of sentences like "List the samples that contain every major element" in anomalous ways such as

```
(FOR EVERY X / MAJORELT : T ;
    (FOR EVERY Y / SAMPLE : (CONTAIN Y X) ;
        (PRINTOUT Y) ))
```

(i.e., "For every major element list the samples that contain it") instead of the correct

```
(FOR EVERY Y / SAMPLE :
    (FOR EVERY X / MAJORELT : T ; (CONTAIN Y X)) ;
    (PRINTOUT Y) ).
```

Except in certain opaque context situations, this seems to be the preferred interpretation. As in other cases, however, although LUNAR's interpretation system is capable of producing alternative interpretations for some other criteria to choose between, the demonstration prototype instead uses rules that determine just those interpretations that seem to be most likely in its domain.

7.5 Other Types of Modifiers

In addition to relative clauses, there are other kinds of constructions in English that function as predicates to restrict the range of quantification. These include most adjectives and prepositional phrases. They are interpreted by RRULES that match the appropriate structures in a noun phrase and produce a predicate with free variable X (which will be instantiated with the variable of quantification for the noun phrase being interpreted). I will call such modifiers *predicators* since they function as predicates to restrict the range of quantification. Examples of predicators are modifiers like "recent" and "about olivine twinning" in phrases like "recent articles about olivine twinning". The interpretation of this phrase would produce the quantifier

```
(FOR GEN X / DOCUMENT :
    (AND (RECENT X) (ABOUT X (OLIVINE TWINNING))) ; DLT ).
```

Note that not all adjectives and prepositional phrases are interpreted as just described. Many fill special roles determined by the head noun, essentially serving as arguments to a function. For example, in a noun phrase such as "the silicon concentration in S10046," the adjective "silicon" is specifying the value of one of the arguments to the function "concentration," rather than serving as an independent predicate that the concentration must satisfy. (That is, this phrase is not equivalent to "the concentration in S10046 which is silicon," which does not make sense). Similarly, the prepositional phrase "in S10046" is filling the same kind of argument role, and is not an independent modifier. I will call this class of modifiers *role fillers*.

In some cases, there are modifiers that could either be treated as restricting predicates or as filling argument roles in a function, depending on the enumeration function that is being used to represent the meaning of the head noun. For example, a modifier like "to Chicago" in "flights to Chicago" could either be interpreted as an independent predicate (ARRIVE X CHICAGO) modifying the flight, or as an argument to a specialized flight enumeration function FLIGHT-TO which enumerates flights to a given destination. In the flight schedules application, the former interpretation was taken, although later query optimization rules (see smart quantifiers, below) were able to transform the resulting MRL expression to a form equivalent to the latter to gain efficiency.

In general English, there are cases in which it seems moot whether one should treat a given phrase as filling an argument role or as a restricting predicate. However, there are also clear cases where the head noun is definitely a function and cannot stand alone without some argument being either explicitly present or inferable from context. In these cases such modifiers are clearly role fillers. On the other hand, the diversity of possible modifiers makes it unlikely that all adjectives and prepositional phrases could be interpretable as role fillers in any general or economical fashion. Thus, the distinction between predicators and role fillers seems to be necessary.

There is another use of a modifier that neither fills an argument role nor stands as an independent predicate, but rather changes the interpretation of the head noun. An example is "modal" in "modal olivine analyses." This adjective does not describe a kind of olivine, but rather a kind of analysis that is different from the normal interpretation one would make of the head "analysis" by itself. Such modifiers might be called *specializers* since they induce a special interpretation on the head noun. Note that these distinctions in types of modification refer to the role of modifier plays in a given construction, not to anything inherent in the modifier itself.

The sentence "List modal olivine analyses for lunar samples that contain silicon" contains a mixture of the different kinds of modifiers. The presence of the specializer adjective "modal" blocks the application of the normal NRULE N:ANALYSIS (it has a NOT template that checks for it), and it enables a different rule N:MODAL-ANALYSIS instead. The adjective "olivine" and the prepositional phrase are both interpreted by REFs in the right-hand side of this rule to fill argument slots in the enumeration function DATALINE. There are no predicators modifying "analyses," but there is a potential predicator "lunar" modifying "samples" and a restrictive relative clause also modifying samples. In LUNAR, the apparently restrictive modifier "lunar" modifying a word like "samples" is instead interpreted as a specializer that does not make a difference, since LUNAR knows of no other kind of sample. However, this is clearly not a limitation of the formalism.

The relative clause modifying "samples" is interpreted as described above to produce the predicate

```
(CONTAIN X2 SIO2).
```

The interpretation of the noun phrase "lunar samples that contain silicon" thus consists of the SEM X2 and the QUANT

```
(FOR GEN X2 / SAMPLE : (CONTAIN X2 SIO2) ; DLT ).
```

This SEM-QUANT pair is returned to the process interpreting the noun phrase "modal olivine analyses for ... ," which in turn produces a SEM X1 and a QUANT

```
(FOR GEN X2 / SAMPLE : (CONTAIN X2 SIO2) ;
    (FOR GEN X1 / (DATALINE X2 OVERALL OLIV) : T ;
        DLT)).
```

This is returned to the rule interpreting the main verb "list," whose right-hand side produces the SEM (PRINTOUT X1) with the same QUANT as above. This process returns to the PRERULE for positive imperative sentences, where the quantifiers are grabbed to produce the interpretation

```
(FOR GEN X2 / SAMPLE : (CONTAIN X2 SIO2) ;
    (FOR GEN X1 / (DATALINE X2 OVERALL OLIV) : T ;
        (PRINTOUT X1) )).
```

7.6 Averages and Quantifiers

An interesting class of quantifier interaction problems occurs with certain operators such as "average," "sum," and "number." In a sentence such as "What is the average silicon concentration in breccias?" it is clear that the generic "breccias" is not to be interpreted as a universal quantifier dominating the average computation, but rather the average is to be performed over the set of breccias. A potential way of interpreting such phrases would be to treat average as a specializer adjective which, when applied to a noun like "concentration," produces a specialized enumeration function that computes the average. This special interpretation rule, would then interpret the class being averaged over in a special mode as a role filler for one of the arguments to the AVERAGE-CONCENTRATION function. However, this approach would lack generality, since it would require a separate interpretation rule and a separate AVERAGE-X function for every averageable measurement X. Instead, one would like to treat average as a general operator that can apply to anything averageable. Doing this, and making it interact correctly with various quantifiers is handled in the LUNAR system by a mechanism of some elegance and generality. I will describe here the interpretation of averages; the interpretations of sums and other such operators are similar.

Note that there are two superficial forms in which the average operator is used: one is a simple adjective modifying a noun ("the average concentration. . ."), and one is as a noun referring to a function that is explicitly applied to an argument ("the average of concentrations . . ."). LUNAR's grammar standardizes this variation by transforming the first kind of structure into the second (effectively inserting an "of ... PL" into the sentence). As a result, average always occurs in syntactic tree structures as the head noun of a noun phrase with a dependent prepositional phrase whose object has a "NIL ... PL" determiner structure and represents the set of quantities to be averaged.

In interpreting such noun phrases, the NRULE invoked by a head noun "average" or "mean" calls for the interpretation of the set being averaged with the special TYPEFLAG SET. This will result in that node's being interpreted with a special DRULE D:SETOF, which will construct an intensional set representation for the set being averaged. The data base function AVERAGE knows how to use such an intensional set to enumerate members and compute the average. The NRULE for "average" is

```
[N:AVERAGE
    (NP.N (MEM 1 (MEAN AVERAGE)))
    (NP.PP (MEM 2 (QUANTITY)))
    → (QUOTE (SEQL (AVERAGE X / (# 2 2 SET) ))) ].
```

7.7 Short Scope/Broad Scope Distinctions

Another interesting aspect of quantifier nesting is a fairly well-known distinction between so called short-scope and broad-scope interpretation quantifiers. For example, Bohnert and Backer (1967) present an account of the differences between "every" and "any" and between "some" and "a" in contexts such as the antecedents of if–then statements by giving "any" and "some" the broadest possible scope and "every" and "a" the narrowest. For example, using the LUNAR MRL notation,

If any soldier stays home, there is no war

```
(FOR EVERY x / soldier ; (IF (home x)
                          THEN (not war))
```

If every soldier stays home, there is no war

```
(IF (FOR EVERY x / soldier ; (home x))
    THEN (not war))
```

If some soldier stays home, there is no war

```
(FOR SOME x / soldier ; (IF (home x)
                         THEN (not war)))
```

If a soldier stays home, there is no war

```
(IF (FOR SOME x / soldier ; (home x))
    THEN (not war)).
```

The scope rules of Bohnert and Backer are enforced rules of an artificial language that approximates English and are not, unfortunately, distinctions that are always followed in ordinary English. In ordinary English, only a few such distinctions are made consistently, while in other cases the scoping of quantifiers appears to be determined by which is most plausible (see discussion of plausibility evaluation in Section 10.5).

In LUNAR, a slightly different form of this short/broad scope distinction arose in the interaction of operators like average with universal quantifiers. For example, the sentence "List the average concentration of silicon in breccias" clearly means to average over all breccias, while "List the average concentration of silicon in each breccia" clearly means to compute a separate average for each breccia. (In general, there are multiple measurements to average even for a single sample.) The sentences "List the average concentration of silicon in every breccia," and "List the average concentration of silicon in all breccias" are less clear, but it seems to be that the average over all breccias is slightly preferred in these cases. At any rate, the treatment of quantifiers needs to be able to handle the fact that there are two possible relative scopings of the average operator with universal quantifiers, and the fact that the choice is determined at least for the determiner "each" and for the "generic" or NIL-PL determiner.

LUNAR handles these scope distinctions for the "average" operator by a general mechanism that applies to any operator that takes a set as its argument. As discussed above, the right-hand side of the N:AVERAGE rule calls for the interpretation of the node representing the set being averaged over with TYPEFLAG SET. This causes a DRULE D:SET OF to be used for interpreting that node. The right-hand side of D:SETOF is

```
    (SETGEN (SETOF X / (# 0 NRULES) : (# 0 RRULES) ))
```

where SETGEN is a function that grabs certain quantifiers coming from subordinate interpretations and turns them into UNION operations instead. The generic quantifier is grabbed by this function and interpreted as a union. However, the quantifier EACH is not grabbed by SETGEN but is passed on up as a dominating quantifier. Thus, the sentence "What is the average concentration of silicon in breccias?" becomes

```
 (FOR THE X4 / (SEQL (AVERAGE X5 /
     (UNION X7 / (SEQ TYPECS) : T ;
         (SETOF X6 / (DATALINE X7 OVERALL SIO2) : T)))) : T ;
     (PRINTOUT X4) )
```

(i.e., the average is computed over the set formed by the union over all type C rocks X7 of the sets of measurements of SIO2 in the individual X7's). On the other hand, "What is the average concentration of silicon in each breccia?" becomes

```
(FOR EACH X12 / (SEQ TYPECS) : T ;
    (FOR THE X9 / (SEQL (AVERAGE X10 /
        (SETOF X11 / (DATALINE X12 OVERALL SIO2) : T ))) : T ;
           (PRINTOUT X9) ))
```

(i.e., a separate average is computed for each type C rock X12).

7.8 Wh Questions

In addition to simple yes/no questions and imperative commands to print the results of computations, LUNAR handles several kinds of so-called wh questions. Examples are "What is the concentration of silicon in S10046?", "Which samples contain silicon?", and "How many samples are there?" These fall into two classes: those in which an interrogative pronoun stands in the place of an entire noun phrase, as in the first example, and those in which an interrogative determiner introduces an otherwise normal noun phrase. In both cases, the noun phrase containing the interrogative word is usually brought to the front of the sentence from the position that it might otherwise occupy in normal declarative word order, but this is not always the case.

7.8.1 *Interrogative Determiners*

The natural representation of the interrogative determiners would seem to be to treat them just like any other determiner and represent a sentence such as the second example above as

```
S Q
  NP   DET WHQ
       N SAMPLE
       NU PL
  AUX  TNS PRESENT
  VP   V CONTAIN
       NP NPR SILICON
```

The interpretation procedure we have described seems to work quite well on this structure using a DRULE that matches the interrogative noun phrase and generates the quantifier

```
(FOR EVERY X / (# 0 NRULES) : (AND (# 0 RRULES) DLT) ;
(PRINTOUT X)).
```

Note that the DLT in the quantifier (where the interpretation of the main clause is to be inserted) is part of the restriction on the range, and the quantified operator is a command to print out the answer. The structure of the quantifier in this case seems somewhat unusual, but the effect is correct and the operation is a reasonably natural one given the capabilities of the semantic interpreter.

However, when we try to apply this kind of analysis to conjoined sentences, such as "What samples contain silicon and do not contain sodium?", the standard kind of deep structure assigned by a transformational grammar to conjoined sentences is not compatible with this interpretation. The usual reversal of the conjunction reduction transformations in a transformational grammar would produce a structure something like

```
S AND
    S Q
      NP    DET WHQ
            N SAMPLE
            NU PL
      AUX   TNS PRESENT
      VP    V CONTAIN
            NP NPR SILICON
    S Q
      NEG
      NP    DET WHQ
            N SAMPLE
            NU PL
      AUX   TNS PRESENT
      VP    V CONTAIN
            NP NPR SODIUM.
```

This structure corresponds to the conjunction of the two questions "What samples contain silicon?" and "What samples do not contain sodium?", which is the interpretation that it would receive by the LUNAR rules with the above DRULE for wh-determiners. However, this is not what the original conjoined question means; the intended question is asking for samples that simultaneously contain silicon and not sodium.

In order to handle such sentences, it is necessary to distinguish some constituent that corresponds to the conjunction of the two predicates "contain silicon" and "not contain sodium," which is itself a constituent of a higher level "what samples" operator. To handle such constructions correctly for both conjoined and nonconjoined constructions, LUNAR's ATN grammar of English was modified to assign a different structure to wh-determiner questions than the one that is assigned to other determiners. These sentences are analyzed as a special type of sentence, a noun phrase question (NPQ), in which the top level structure of the syntactic representation is that of a noun phrase, and the matrix sentence occurs as a special kind of subsidiary relative clause. For example, the sentence

"Which samples contain silicon?" is represented syntactically as

```
S NPQ
  NP   DET WHICHQ
       N SAMPLE
       NU PL
       S QREL
         NP   DET WHR
              N SAMPLE
              NU PL
         AUX  TNS PRESENT
         VP   V CONTAIN
              NP DET NIL
                 N SILICON
                 NU SG.
```

This structure provides an embedded S node inside the higher level question, whose interpretation is a predicate with free variable bound in the question operator above. This embedded S node can be conjoined freely with other S nodes, while remaining under the scope of a single question operator. In this case, the appropriate DRULE (for a wh-determiner in a plural NPQ utterance) is simply

```
[D: WHQ-PL
   (NP.DET (AND (MEM 1 WHQ) (EQU 2 PL)))
   →
   (QUANT (FOR EVERY X / (# 0 NRULES) :
      (# 0 RRULES) ; (PRINTOUT X))) ].
```

Since the matrix sentence has been inserted as a relative clause in the syntactic structure assigned by the grammar, it will be interpreted by the RRULE R:REL in the subordinate interpretation (# 0 RRULES). A similar rule for interpreting singular noun phrases ("which sample contains. . .") produces a quantifier with ⟨quant⟩ = THE, instead of EVERY, thus capturing the presupposition that there should be a single answer.

All of the interrogative determiners, "which," "what," and "how many" are treated in the above fashion. The right-hand side of the "how many'' rule is

```
(FOR THE X / (NUMBER X / (# 0 NRULES) : (# 0 RRULES)) ;
    (PRINTOUT X)).
```

Here again, the interpretation of the matrix sentence is picked up in the call (# 0 RRULES). (The use of the same variable name in two different scopes does not cause any logical problems here, so no provision was made in LUNAR to create more than one variable for a given noun phrase.)

7.8.2 Interrogative Pronouns

A general treatment of the interrogative pronouns would require modifications of the assigned syntactic structures similar to the ones discussed above for interrogative determiners in order to handle conjunctions correctly. That is, sentences such as ''What turns generic quantifiers into set unions and passes 'each' quantifiers through to a higher level?'' seem to require an embeded S node to serve as a conjoined proposition inside a single ''what'' operator. However, it is far more common for conjoined questions with interrogative pronouns to be interpreted as a conjunction of two separate questions. This is especially true for conjoined ''what is ...'' questions. For example, ''What is the concentration of silicon in S10046 and the concentration of rubidium in S10084?'' is clearly not asking for a single number that happens to be the value of the concentration in both cases.

The LUNAR system contains rules for handling interrogative pronouns only in the special case of ''what is. . .'' questions. In this special case, conjoined questions fall into two classes, both of which seem to be handled correctly without special provisions in the grammar. In questions where the questioned noun phrase contains an explicit relative clause, that clause will contain an S node where conjunctions can be made and LUNAR's current techniques will treat this as one question with a conjoined restriction (e.g., ''What is the sample that contains less than 15% silicon and contains more than 5% nickel?''). On the other hand, when there is no explicit relative clause, LUNAR will interpret such questions as a conjunction of separate questions (e.g., ''What is the concentration of silicon in S10046 and the concentration of rubidium in S10084?'').

The conventional structure assigned to ''what is. . .'' sentences by a transformational grammar represents the surface object as the deep subject, with a deep verb ''be'' and predicate complement corresponding to the interrogative pronoun "what." For example, in LUNAR the question "What is the concentration of silicon in S10046?" becomes

```
S Q
   NP    DET THE
         N CONCENTRATION
         NU SG
         PP  PREP OF
             NP DET NIL
                N SILICON
                NU SG
         PP  PREP IN
             NP NPR S10046
   AUX   TNS PRESENT
   VP    V BE
         NP DET WHQ
            N THING
            NU SG/PL
```

A special SRULE for the verb "be" with complement "WHQ THING SG/PL" handles this case with a right-hand side schema:

```
(QUOTE (PRINTOUT (# 1 1)))
```

where the REF (# 1 1) refers to the subject noun phrase.

A somewhat more general treatment of the interrogative pronoun "what" would involve a DRULE whose right-hand side was

```
(FOR EVERY X / THING : DLT ; (PRINTOUT X) ).
```

Where the interpretation of the matrix sentence is to be inserted as a restriction, on the range of quantification and the overall interpretation is a command to print out the values that satisfy it. (THING in this case is meant to stand for the universal class.) One would not want to apply this rule in general to the simple "What is ..." questions as above, since it would result in an interpretation that was less efficient (i.e., would enumerate all possible things and try to filter out the answer with an equality predicate). For example, "what is the concentration of silicon in S10046" would be interpreted

```
(FOR THE X / (DATALINE S10046 OVERALL SIO2) : T ;
    (FOR EVERY Y / THING : (EQUAL X Y) ;
        (PRINTOUT Y) ))
```

instead of

```
(FOR THE X / (DATALINE S10046 OVERALL SIO2) : T ;
    (PRINTOUT X)).
```

Thus, one would still want to keep the special "what is ..." rule and LUNAR would only use the general rule in cases where the "what is..." rule did not apply. (When the "what is ..." rule does apply, it does not even call for the interpretation of the "what" noun phrase that it has matched, so the general rule would not be invoked.)

Alternatively, one could use the general rule for all cases and then perform post-interpretive query optimization (see Section 8) to transform instances of filtering with equality predicates to a more efficient form that eliminates the unnecessary quantification.

7.8.3 Other Kinds of Wh Questions

Note that LUNAR interprets "what is ..." questions only as a request for the value of some function or the result of some search or computation, and not as requesting a definition or explanation. For example if LUNAR is asked "what is a sample" it will respond with an example (e.g., "S10046"), and if it is asked "what is S10046," it will respond "S10046." LUNAR is not aware of the internal structure of the defining procedures for its terms, nor does it have any intensional description of what samples are, so it has no way of answering the first type of question. There is no difficulty, however, in defining another rule for "what is ..." to apply to proper nouns and produce an interpretation with an operator NAME-CLASS (instead of PRINTOUT) that will print the class of an individual instead of its name. "What is S10046?" would then be interpreted as (NAME-CLASS S10046), which would answer "a sample."

Getting LUNAR to say something more complete about how S10046 differs from other samples, such as "a sample that contains a large olivine inclusion," is another matter. Among other problems, this would begin to tread into the area of pragmatics, where considerations such as the user's probable intent in asking the question and appropriateness of response in a particular context, as well as semantic considerations of meaning, become an issue (see Section 11.5). All of this is well beyond the scope of systems like LUNAR. However, deciding what semantic representation to assign as the intent of such a question is not nearly as difficult as deciding what the defining procedure for some of the possible intents should be. LUNAR's mechanisms are suitable for generating the alternative possible semantic representstions.

8. Post-Interpretive Processing

As mentioned before, the LUNAR meaning representation language has been designed both as a representation of executable procedures and as a symbolic structure that can be manipulated as an intensional object. Although every expression in the LUNAR MRL has an explicit semantics defined by its straightforward execution as a procedure, that procedure is frequently not the best one to execute to answer a question or carry out a command. For example, in the flight schedules applications, the literal interpretation of the expression

```
(FOR EVERY X / FLIGHT : (CONNECT X BOSTON CHICAGO) ;
    (PRINTOUT X))
```

is to enumerate all of the flights known to the system, filtering out the ones that do not go from Boston to Chicago, and printing out the rest. However, in a reasonable data base for this domain, there would be various indexes into the flights, breaking them down by destination city and city of origin. If such an index exists, then a specialized enumeration function FLIGHT-FROM-TO could be defined for using the index to enumerate only flights from a given city to another. In this case, the above request could be represented as

```
(FOR EVERY X / (FLIGHT-FROM-TO BOSTON CHICAGO) : T ;
    (PRINTOUT X)),
```

which would be much more efficient to execute.

Given the possibility of using specialized enumeration functions, one can then either write special interpretation rules to use the more specific enumeration function in the cases where it is appropriate, or one can perform some intensional manipulations on the interpretation assigned by the original rules to transform it into an equivalent expression that is more efficient to execute. The first approach was used in the original flight schedules system. An approach similar to the latter was used in the grammar information system, and to some extent in LUNAR, by using "smart" quantifiers (see below). Recently, Reiter (1977) has presented a systematic treatment of a class of query optimizations in systems like LUNAR that interface to a relational data base.

Other post-interpretive operations on the MRL expression are performed in LUNAR to analyze the quantifiers and make entries in a discourse directory for potential antecedents of anaphoric expressions. Subsequently, definite descriptions and pronouns can make reference to this directory to select antecedents. I will not go into the treatment of anaphoric expressions in this paper other than to say that the search for the antecedent is invoked by an operator ANTEQUANT in the right-hand side of the DRULES that interpret anaphoric noun phrases. In general, this results in the generation of a quantifier, usually a copy of the one that was associated with the antecedent. Occasionally, the antecedent will itself fall in the scope of a higher quantifier on which it depends, in which case such governing quantifiers will also be copied and incorporated into the current interpretation. Some of the characteristics of LUNAR's treatment of anaphora are covered in Nash-Webber (1976) and woods *et al.* (1972).

8.1 Smart Quantifiers

In the grammar information system, a notation of "smart" quantifier was introduced, which rather than blindly executing the quantification procedure obtained from semantic interpretation, made an effort to determine if there was a more specific enumeration function that could be used to obtain an equivalent answer. In general, the restriction on the range of quantification determines a subclass of the class over which quantification is ranging. If one can find a specialized enumeration function that enumerates a subclass of the original class but is still guaranteed to include any of the members that would have passed the original restriction, then that subclass enumeration function can be used in place of the original.

In the grammar information system, tables of specialized enumeration functions, together with sufficient conditions for their use, were stored associated with each basic class over which quantification could range. A resolution theorem prover a la Robinson (1965) was then used to determine whether the restriction of a given quantification implied one of the sufficient conditions for a more specialized class enumeration function. If so, the more specialized function was used. Unlike most applications of resolution theorem proving, the inferences required in this case are all very short, and since the purpose of the inference is to improve the efficiency of the quantification, a natural bound can be set on the amount of time the theorem prover should spend before the attempt should be given up and the original enumeration function used.

In general, sufficiency conditions for specialized enumeration functions are parameterized with open variables to be instantiated during the proof of the sufficiency condition and then used as parameters for the specialized enumeration function. The resolution theorem proving strategies have a nice feature of providing such instantiated parameters as a result of their proofs; e.g., by using a mechanism such as the "answer" predicate of Green (1969).

Smart quantifiers were intended in general to be capable of other operations, such as estimating the cost of a computation from the sizes of the classes being quantified over and the depth of quantifier nesting (and warning the user if the cost might be excessive), saving the results of inner loop quantifications where they could be reused, interchanging the scopes of quantification to bring things that do not change outside a loop, etc. The capabilities actually implemented, however, are much more limited.

8.1.1 Path Enumeration in ATN's

Smart quantifiers were essential for efficiency in the grammar information system's enumeration of paths through its ATN. The system contained a variety of specialized path enumeration functions: one for paths between a given pair of states, one for paths leaving a given state, one for paths arriving at a given state, one for paths irrespective of end states, and versions of all of these for looping and nonlooping paths. Each specialized enumeration function was associated with a parameterized sufficiency condition for its use. For example, the function for nonlooping paths leaving a given state had a table entry equivalent to

(PATHSEQ Y T) if (AND (NOLOOP X) (START X Y))

where X refers to the variable of the class being quantified over, Y is a parameter to be instantiated, and (PATHSEQ Y T) is the enumeration function to be used if the sufficiency condition is satisfied.

Thus, if a quantification over paths had a restriction such as (AND (CONNECT-PATH X S/ S/VP) (NOLOOP X)) and the theorem prover had axioms such as (CONNECT-PATH X Y Z)$\Rightarrow$(START X Y), then the theorem prover would infer that the sufficiency condition (AND (NOLOOP X) (START X Y)) is satisfied with Y equal to S/ and therefore the specialized enumeration function (PATHSEQ S/ T) can be used.

Notice that the order of conjuncts in the restriction is irrelevant, and the restriction need only imply the sufficiency condition not match it exactly. In the above, there are still conditions in the restriction that will have to be checked as a filter on the output of the specialized enumeration function to make sure that the end of the path is at state S/VP. In general, it would be nice to remove from the restriction that portion that is already guaranteed to be satisfied by the new enumeration function, but that is easier said than done. In the grammar information system the original restriction was kept and used unchanged.

8.1.2 Document Retrieval in LUNAR

In the LUNAR system, a special case of smart quantifiers, without a general theorem prover, is used to handle enumeration of documents about a topic. When the FOR function determines that the class of objects being enumerated is DOCUMENT, it looks for a predicate (ABOUT X TOPIC) in the restriction (possibly in the scope of a conjunction but not under a negative). It then uses this topic as a parameter to an inverted file accessing routine which retrieves documents about a given topic.

8.2 Printing Quantifier Dependencies

The LUNAR MRL permits the natural expression of fairly complex requests such as "What is the average aluminum concentration in each of the type c rocks?" The interpretation of this request would be

```
(FOR EVERY X / (SEQ TYPECS) : T ;
   (FOR THE Y / (AVERAGE Z / (DATALINE X OVERALL AL2O3))
      : T ; (PRINTOUT Y) )).
```

If the PRINTOUT command does nothing more than print out a representation for the value of its argument, the result of this command will be nothing more than a list of numbers, with no indication of which number goes with which of the rocks. Needless to say, this is usually not what the user expected.

For special classes of objects, say concentrations, a pseudo-solution to this problem would be to adopt a strategy of always printing out all conceivable dependencies for that object (e.g., the sample, phase, and element associated with that concentration). This would be sufficient to indicate what dependencies each answer had on values of arguments, but would take no account of which of those dependencies was currently varying and which were fixed by the request. Moreover, this approach would not work in the above case, since the objects being printed are the results of a general purpose numerical averaging function, which does not necessarily have any dependencies, depending on what is being averaged and what classes are being averaged over.

LUNAR contains a general solution to this quantifier dependency problem that is achieved by making the PRINTOUT command an opaque operator that processes its argument in a semi-intelligent way as an intensional object. PRINTOUT examines its argument for the occurrence of free variables. If the argument is itself a variable, it looks up the corresponding governing quantifier in the discourse directory (the same directory used for antecedents of anaphoric expressions) and checks that quantifier for occurrences of free variables. If it finds free variables in either place, it means that the object it is about to print has a dependency on those variables. In that case it prints out the current values of those variables along with the value that it is about to print out. In the case of the example above, the variable Y has the corresponding class specification (DATALINE X OVERALL SIO2) with restriction T, and is thus dependent on the variable X, which is ranging over the rocks. As a result, the printout from this request would look like

```
S10018 12.48 PCT
S10019 12.80 PCT
S10021 12.82 PCT
       :
```

This mechanism works for arbitrary nesting of any number of quantifiers.

9. An Example

As an example of the overall operation of the semantic interpreter to review and illustrate the preceding discussions, consider the sentence

"What is the average modal plagioclase concentration for lunar samples that contain rubidium?"

This sentence has the following syntactic structure assigned to it by the grammar:

```
S Q
  NP   DET THE
       N AVERAGE
       NU SG
       PP  PREP OF
               NP DET  NIL
                  ADJ  MODAL
                  ADJ  N PLAGIOCLASE
              N CONCENTRATION
              NU   PL
              PP   PREP FOR
                   NP DET NIL
                      ADJ LUNAR
                      N SAMPLE
                      NU PL
                      S  REL
                         NP    DET WHR
                               N SAMPLE
                               NU PL
                         AUX   TNS PRESENT
                         VP    V CONTAIN
                               NP DET NIL
                                  N RUBIDIUM
    AUX  TNS PRESENT              NU SG
    VP   V BE
    NP   DET WHQ
         N THING
         NU SG/PL.
```

Semantic interpretation begins with a call to INTERP looking at the topmost S node with TYPEFLAG NIL. The function RULES looking at an S node with TYPEFLAG NIL returns the global rule tree PRE-RULES. These rules look for such things as yes/no question markers, sentential negations, etc. In this case, a rule PR6 matches and right-hand side, (PRED (# 0 SRULES)), specifies a call to INTERP for the same node with TYPEFLAG SRULES.

The function RULES looking at the S node with TYPEFLAG SRULES returns a rule tree which it gets from the dictionary entry for the head of the sentence (the verb BE), and in this case a rule S: BE-WHAT matches. Its right-hand side is

(PRED (PRINTOUT (# 1 1)))

specifying a schema into which the interpretation of the subject noun phrase is to be inserted.

The semantic interpreter now begins to look at the subject noun phrase with TYPEFLAG NIL. In this case, RULES is smart enough to check the determiner THE and return the rule tree:

(D:THE-SG2 NIL D:THE-SG NIL D:THE-PL)

of which, the rule D:THE-SG matches successfully. The right-hand side of this rule is

(QUANT (FOR THE X / (# 0 NRULES) : (# 0 RRULES) ; DLT))

which specifies that a quantifier is to be constructed by substituting in the indicated places the interpretations of this same node with TYPE-FLAGs NRULES and RRULES.

The call to interpret the subject noun phrase with TYPEFLAG NRULES finds a list of NRULES in the dictionary entry for the word "average," consisting of the single rule N: AVERAGE. This rule, which we presented previously in Section 7.6, has a right-hand side

(QUOTE (SEQL (AVERAGE X / (# 1 1 SET))))

which calls for the interpretation of the "concentration" noun phrase with TYPEFLAG SET. The call to interpret the "average" node with TYPEFLAG RRULES, which will be done later, will result in the empty restriction T.

The call to interpret the "concentration" noun phrase with TYPE-FLAG SET uses a list of rules (D:SETOF NIL D:NOT-SET) where D:SETOF, which has been discussed previously in Section 7.7, checks for a determiner and number consistent with a set interpretation (i.e., determiner THE or NIL and number PL) and D:NOT-SET will match anything else. In this case, D:SETOF matches, with right-hand side

(SETGEN (SETOF X / (# 0 NRULES) : (# 0 RRULES)))

and calls for the interpretation of the same node with TYPEFLAGs NRULES and RRULES. The call with NRULES finds a matching rule N:MODAL-CONC after failing to match N:CONCENTRATION because of the presence of the adjective MODAL, which is rejected by a negated template. N:MODAL-CONC is used to interpret modal concentrations of minerals in samples as a whole, and has the form

```
[N:MODAL-CONC
  (NP.N (MEM 1 (CONCENTRATION)))
  (OR (NP.PP (MEM 2 (SAMPLE)))
      (NP.PP.PP (MEM 2 (SAMPLE)))
      (DEFAULT (2 NP (DET EVERY)
                     (N SAMPLE)
                     (NU SG))))
  (OR (NP.PP (MEM 2 (PHASE MINERAL ELEMENT
                    OXIDE ISOTOPE)))
  (NP.ADJ#2 (MEM 2 (PHASE MINERAL ELEMENT
                    OXIDE ISOTOPE))))
  → (QUOTE (DATALINE (# 2 2) OVERALL (# 3 2))) ].
```

(DEFAULT is a special kind of template that always succeeds and that makes explicit bindings for use in the right-hand side. In the above case, if the "concentration" noun phrase had not mentioned a sample, then the default "every sample" would be assumed.)

N: MODAL-CONC in turn calls for the interpretations of the "sample" noun phrase and the constituent "rubidium." In interpreting the "sample" noun phrase, it again goes through the initial cycle of DRULES selected by TYPEFLAG NIL looking at a noun phrase, in this case finding a matching rule D: NIL whose right-hand side is

```
(QUANT (FOR GEN X / (# 0 NRULES) : (# 0 RRULES) ; DLT ))
```

This in turn invokes an NRULES interpretation of the same phrase which uses the rule tree (N:TYPEA N:TYPEB N:TYPEC N:TYPED NIL N: SAMPLE) that looks first for any of the specific kinds of samples that might be referred to, and failing any of these, tries the general rule N:SAMPLE. N:SAMPLE checks for the head "sample" with an optional adjective "lunar" or the complete phrase "lunar material" and has a right-hand side

```
(QUOTE (SEQ SAMPLES))
```

where SEQ is the general enumeration function for known lists, and SAMPLES is a list of all the samples in the data base.

The RRULES interpretation uses the rule tree ((AND R:SAMPLE-WITH R:SAMPLE-WITH-COMP R:QREL R:REL R:PP R:ADJ)), which contains a single AND group of rules, all of which are to be tried and the results of any successful matches conjoined. The rule R: REL matches the relative clause, tagging the relative pronoun with the variable of interpretation X13 and then calling for the interpretation of the relative clause via the right-hand side

(PRED (# 1 1)).

The interpretation of the relative clause, like that of the main clause begins with a set of PRERULES, of which a rule PR6 matches with right-hand side

(PRED (# 0 SRULES)).

This again calls for the interpretation of the same node with TYPEFLAG SRULES. This interpretation finds the rule S:CONTAIN (presented earlier in Section 6), whose right-hand side calls for the interpretation of its subject noun phrase (which it finds already interpreted with the variable of quantification from above) and its object noun phrase "rubidium." The latter is interpreted by a rule D:MASS, whose right-hand side looks up the word "rubidium" in the dictionary to get its standard data base representation RB (from a property name TABFORM) and produces the interpretation (QUOTE RB). As a SEM-QUANT pair, this is

((QUOTE RB) DLT).

This interpretation, together with that of the relative pronoun is returned to the process interpreting the "contain" clause, where they produce (after substitution and right-hand side evaluation) the SEM-QUANT pair

((CONTAIN X13 (QUOTE RB)) DLT).

This same SEM-QUANT pair is return unchanged by the R:REL rule and since that is the only matching RRULE, no conjoining needs to be done to obtain the result of the RRULES interpretation of the "sample" noun phrase. Inserting this and the NRULES interpretation into the right-hand side of D:NIL, and executing, produces the SEM-QUANT pair

```
(X13 (FOR GEN X13 / (SEQ SAMPLES) :
   (CONTAIN X13 (QUOTE RB)) ; DLT ))
```

where the right-hand side evaluation of the QUANT operator has embedded the quantifier in the QUANT accumulator and returned the SEM X13.

We now return to the NRULES interpretation of the "concentration" noun phrase, whose right-hand side called for the above interpretation and now calls for the interpretation of "plagioclase." Again, the D:MASS rule applies, looking up the TABFORM of the word in the dictionary and resulting in the SEM-QUANT pair

((QUOTE PLAG) DLT).

The substitution of these two into the right-hand side of the rule N:MODAL-CONC (and evaluating) produces the SEM-QUANT pair:

```
((DATALINE X13 OVERALL (QUOTE PLAG))
    (FOR GEN X13 / (SEQ SAMPLES) :
        (CONTAIN X13 (QUOTE RB)) ; DLT ))
```

where the quantifier from below is still being passed up.

The RRULES interpretation of the "concentration" noun phrase produces T, since there are no predicating modifiers, and the insertion of these two into the right-hand side of the rule D:SETOF produces

```
(SETGEN (SETOF X12 / (DATALINE X13 OVERALL
                        (QUOTE PLAG)) : T ))
```

while the quantifier accumulator QUANT contains the collar

```
(FOR GEN X13 / (SEQ SAMPLES) : (CONTAIN X13 (QUOTE RB)) ;
                        DLT).
```

The execution of the function SETGEN grabs the generic quantifier from the register QUANT, leaving QUANT set to DLT, and produces the SEM

```
(UNION X13 / (SEQ SAMPLES) : (CONTAIN X13 (QUOTE RB)) ;
 (SETOF X12 / (DATALINE X13 OVERALL (QUOTE PLAG)) : T )).
```

The quantification over samples has now been turned into a union of sets of data lines over a set of samples.

The resulting SEM and QUANT are returned to the process that is interpreting the "average" phrase, where the insertion into the right-hand side of the rule N:AVERAGE and subsequent evaluation yields the SEM-QUANT pair

```
((SEQL (AVERAGE X11 / (UNION X13 /
    (SEQ SAMPLES) : (CONTAIN X13 (QUOTE RB)) ;
        (SETOF X12 / (DATALINE X13 OVERALL
                        (QUOTE PLAG)) : T )))) DLT).
```

Interpretation of the "average" phrase with TYPEFLAG RRULES produces the SEM-QUANT pair (T DLT), and the insertion of this and the above into the right-hand side of the DRULE D:THE-SG and evaluating yields the SEM-QUANT pair

```
(X11 (FOR THE X11 / (SEQL (AVERAGE X11 / (UNION X13 /
 (SEQ SAMPLES) : (CONTAIN X13 (QUOTE RB)) ;
  (SETOF X12 / (DATALINE X13 OVERALL (QUOTE PLAG)) : T))))
    : T ; DLT)).
```

This is returned to the SRULE S:BE-WHAT where the SEM X11 is embedded in the right-hand side to produce:

```
(PRED (PRINTOUT X11)).
```

Evaluating this expression grabs the quantifier to produce the new SEM, which the next higher rule, PR6, passes on unchanged as the final interpretation:

```
(FOR THE X11 / (SEQL (AVERAGE X11 / (UNION X13 /
  (SEQ SAMPLES) : (CONTAIN X13 (QUOTE RB)) ;
    (SETOF X12 / (DATALINE X13 OVERALL (QUOTE PLAG)) : T))))
      : T ; (PRINTOUT X11)).
```

10. Loose Ends, Problems, and Future Directions

The techniques that I have described make a good start in handling the semantic interpretation of quantification in natural English—especially in the interaction of quantifiers with each other, with negatives, and with operators like "average." However, problems remain. Some reflect LUNAR's status as an intermediate benchmark in an intended ongoing project. Others reflect the presence of some difficult problems that LUNAR would eventually have had to come up against. In the remaining sections, I will discuss some of the limitations of LUNAR's techniques, problems left unfaced, and trends and directions for future work in this area.

10.1 Approximate Solutions

One characteristic of some of the techniques used in LUNAR and many other systems is that they are only approximate solutions. A good example of an approximate solution to a problem is illustrated by LUNAR's use of the head word of a constituent as the sole source of features for the testing of semantic conditions in the left-hand sides of rules. To be generally adequate, it seems that semantic tests should be applied to the *interpretation* of a phrase, not just its syntactic structure (and especially not just its head). Some of the problems with the approximate approach became apparent when LUNAR first began to handle conjoined phrases. For example, it's simple semantic tests were no longer adequate when, instead of a single noun phrase of type X, a conjunction was encountered. This was due to a prior decision that the head of a conjoined phrase should be the conjunction operator (e.g., AND), since a constituent should have a unique head and there is no other unique candidate in a coordinate conjunction. However, since a conjunction operator would never have the semantic features expected by a rule, selectional restrictions applied to the head would not work.

A possible solution to this problem is to define the semantic features of a conjoined phrase to be the intersection of the features of its individual conjuncts. This has the attractive feature of enforcing some of the well-known parallelism constraints on conjunctions in English (i.e., conjoined constituents should be of like kind or similar in some respect). However, this solution is again only an approximation of what is required to fully model parallelism constraints. For example, it does not consider factors of size or complexity of the conjuncts. Further experience with such a model will almost certainly uncover still more problems.

Another example where obtaining the features from the head alone is inadequate involves noun phrases in which an adjective modifying the head contributes essential information (e.g., obtaining a feature +TOY from the phrase "toy gun"). In general, semantic selectional restrictions seem to require intensional models of potential referents rather than just syntactic structures. (In fact, their applying to such models is really the only justification for calling such constraints "semantic.") In my paper "Meaning and Machines" (Woods, 1973c), I discuss more fully the necessity for invoking models of semantic reference for correctly dealing with such restrictions.

More seriously, the whole treatment of selectional restrictions as prerequisites for meaningfulness is not quite correct, and the details of making selectional restrictions work correctly in various contexts such as modal sentences (especially assertions of impossibility) are far from worked out. For example, there is nothing wrong with the assertion "Rocks cannot love people" even if there seems to be something odd about "the rock loved John." Again, Woods (1973c) discusses such problems more fully.

10.2 Modifier Placement

Another area in which LUNAR's solution to a problem was less than general is in the interpretation of modifiers that are syntactically ambiguous as to what they modify. For example, in the sentence "Give me the average analysis of breccias for all major elements," there are at least three syntactic possibilities for the modifier "for all major elements" (it can modify the phrases headed by "breccias," "analysis," or "give"). In this case, our understanding of the semantics of the situation tells us that it modifies "analysis," since one can *analyze* a sample for an element, while "breccias for all major elements" does not make sense.

Without a semantic understanding of the situation, the computer has no criteria to select which of these three cases to use.

One of the roles that one might like the syntactic component to play in a language understanding system would be to make the appropriate grouping of a movable modifier with the phrase it modifies, so that the subsequent semantic interpretation rules will find the constituent where they would like it to be. However, since there is not always enough information available to the parser to make this decision on the basis of syntactic information alone, this would mean requiring the parser to generate all of the alternatives, from which the semantic interpreter would then make the choice. This in turn would mean that the interpreter would have to spend effort typing to interpret a wrong parsing, only to have to throw it away and start over again on a new one. It would be better for the parser to call upon semantic knowledge earlier in the process, while it is still trying to enumerate the alternative possible locations for the movable modifier. The question it would ask at this point would simply be whether a given phrase can take the kind of modifier in question, rather than a complete attempt to interpret each possibility.

10.2.1 Selective Modifier Placement

In general, the ATN grammars used in LUNAR tend to minimize the amount of unnecessary case analysis of alternative possible parsings by keeping common parts of different alternatives merged until the point in the sentence is reached where they make different predictions. At such a point, the choice between alternatives is frequently determined by having only one of their predictions satisfied. However, one place where this kind of factoring does not significantly constrain the branching of possibilities is at the end of a constituent where the grammar permits optional additional modifiers (e.g., prepositional phrase modifiers at the end of a noun phrase, as in the above example). Here, the alternatives of continuing to pick up modifiers at the same level and popping to a higher level have to be considered separately. If when the alternative of popping a constituent is chosen and the construction at the higher level can also take the same kind of modifier as the lower constituent, then a real ambiguity will result unless some restriction makes the modifier compatible with only one of the alternatives.

The LUNAR parser contains a facility called "selective modifier placement" for dealing with such "movable modifiers." When this facility is enabled, each time a movable modifier is constructed, the parser returns to the level that pushed for it to see if the configuration that caused the push could also have popped to a higher level and, if so, whether that higher level could also have pushed for the same thing. It repeats this process until it has gathered up all of the levels that could possibly (syntactically) use the modifier. It then asks semantic questions to rank order the possibilities, choosing the most likely one, and generating alternatives for the others. In a classic example, "I saw the man in the park with a telescope," the phrase "in the park" could modify either "man" or "see," and "with a telescope" could modify either "park," "man," or "see" (with the possible exception, depending on your dialect, of forbidding "with a telescope" from modifying "man" if "in the park" is interpreted as modifying "see"). The selective modifier placement facility chooses the interpretation "see with a telescope" and "man in the park" when given information that one can see with an optical instrument. Woods (1973a) describes this facility for selective modifier placement more fully.

10.2.2 Using Misplaced Modifiers

Although the selective modifier placement facility in LUNAR's parser is probably very close to the right solution to this problem of movable modifiers, the mechanism as implemented requires the semantic information that it uses to be organized in a slightly different form from that used in the semantic interpretation rules. Rather than duplicate the information, LUNAR's demonstration prototype used a different approach. In this sytem, the grammar determined an initial placement of such modifiers based solely on what prepositions a given head noun could take as modifiers. Subject to this constraint, the movable modifier was parsed as modifying the nearest preceding constituent (i.e., as deep in the parse tree as premitted by the constraint). Subsequently during interpretation, if the semantic interpreter failed to find a needed constituent at the level it wanted it, it would look for it attached to more deeply embedded levels in the tree.

If this procedure for looking for misplaced modifiers had been handled by a general mechanism for looking for misplaced constituents subject to appropriate syntactic and semantic guidance, it would provide an alternative approach of comparable generality to selective modifier placement, raising an interesting set of questions as to the relative advantages of the two approaches. In the demonstration prototype, however, it was handled by the simple expedient of using disjunctive templates in the rules to look for a constituent in each of the places where it might occur. Each rule thus had to be individually tailored to look for its needed constituents wherever they might occur. Problems were also present in making sure that all modifiers were used by some rule and avoiding duplicate use of the same modifier more than once.

A number of such decisions were made in LUNAR for the expedient of getting it working, and are not necessarily of theoretical interest. This particular one is mentioned here because of its suggestion of a possible way to handle a problem, and also to illustrate the difference between solving a problem in general and patching a system up to handle a few cases.

10.3 Multiple Uses of Constituents

Alluded to above in the discussion of LUNAR's method of looking for misplaced modifiers was the potential for several different rules to use the same constituent for different purposes. In general, one expects a given modifier to have only one function in a sentence. However, this is not always the case. For example, an interesting characteristic of the "average" operator is the special use of a prepositional phrase with the preposition "over," which usurps one of the arguments of the function being averaged. Specifically, in "the average concentration of silicon over the breccias," the prepositional phrase "over the breccias" is clearly an argument to the average function, specifying the class of objects over which the average is to be computed. However, it is also redundantly specifying the variable that will fill the constituent slot of the concentration schema, even though it does not have any of the prepositions that would normally specify this slot. The semantic interpretation framework that the LUNAR system embodies does not anticipate the simultaneous use of a constituent as a part of two different operators in this fashion (although the implemented mechanism does not forbid it).

The rules in the implemented LUNAR system deal with this problem (as opposed to solving it) by permitting the prepositional phrase with "over" to modify concentration rather than average. This choice was made because the average operator is interpretable without a specific "over" modifier, whereas the concentration is not interpretable without a constituent whose concentration is being measured. However, this "solution" leaves us without any constraint that "over" can only occur with averages. Consequently, phrases such as "the concentration of silicon over S10046" would be acceptable. Such lack of constraint is generally not a serious problem in very restricted topic domains and with relatively simple sentences, because users are unlikely to use one of the unacceptable constructions. However, as the complexity of the language increases, especially with the introduction of constructions such as reduced relative clauses and conjunction reduction, the possibility increases that some of these unacceptable sequences may be posed as partial parsings of an otherwise acceptable sentence, and can either result in unintended parsings or long excursions into spurious garden path interpretations.

This kind of ad hoc "solution" to the "average...over..." problem is typical of the compromises made in many natural language systems, and is brought up here to illustrate the wrong way to attack a problem. It contrasts strongly with the kinds of general techniques that typify LUNAR's solutions to other problems.

10.4 Ellipsis

Possibly the correct solution to the problem of "average...over..." is one that handles a general class of ellipsis—those cases where an argument is omitted because it can be inferred from information available elsewhere in a sentence. In this account, the "over" phrase would be an argument to "average" and the subordinate "concentration" phrase would have an ellipsed specification of the constituent being measured.

A similar problem with ellipsis occurs in the flight schedules context, where sentences such as

List the departure time from Boston of every TWA flight to Chicago.

would be interpreted literally as asking for the Boston departure times of all TWA flights that go to Chicago, regardless of whether they even go through Boston. To express the intended request without ellipsis, the user would have to say

List the departure time from Boston of every TWA flight *from Boston* to Chicago.

As I pointed out in my thesis (Woods, 1967), the information in the semantic rules provides the necessary information for the first step in treating such ellipsis—the recognition that something is missing. Capitalizing on this, however, requires a rule-matching component that is able to find and remember the closest matching rule when no rule matches fully, and to provide specifications of the missing pieces to be used by some search routine that tries to recover the ellipsis. This latter routine would have to examine the rest of the structure of the sentence, and perhaps some of the discourse history, to determine if there are appropriate contextually specified fillers to use. Research problems associated with such ellipsis have to do with the resolution of alternative possible fillers that meet the description, finding potential fillers that are not explicitly mentioned elsewhere but must be inferred, and characterizing the regions of the surrounding context that can legitimately provide antecedents for ellipsis (e.g., can they be extracted out of subordinate relative clauses that do not dominate the occurrence of the ellipsis?).

10.5 Plausibility of Alternative Interpretations

In general, the correct way to handle many of the potential ambiguities that arise in English seems to be to construct representations of alternative interpretations, or alternative parts of interpretations, and evaluate the alternatives for their relative plausibility. LUNAR does not contain such a facility. Instead, it makes the best effort it can to resolve ambiguities, given what it knows about general rules for preferred parsings, criteria for preferred interpretations, and specific semantic selectional restrictions for nouns and verbs. LUNAR does quite well within these constraints in handling a wide variety of constructions. This is successful largely because of the limited nature of the subject matter and consequent implicit constraints on the kinds of questions and statements that are sensible. However, a variety of phenomena seem to require a more general plausibility evaluator to choose between alternatives. If one had such an evaluator of relative plausibility, the mechanisms used in LUNAR would be adequate to generate the necessary alternatives.

10.6 Anaphoric Reference

Anaphoric reference is another problem area in which LUNAR's treatment does not embody a sufficiently general solution. Every time an interpretation is constructed, LUNAR makes entries in a discourse directory for each constituent that may be subsequently referred to anaphorically. Each entry consists of the original syntactic structure of a phrase, plus a slightly modified form of its semantic interpretation. In response to an anaphoric expression such as "it" and "that sample," LUNAR searches this directory for the most recent possible antecedent and reuses its previous interpretation.

LUNAR's anaphoric reference facility is fairly sophisticated, including the possibility to refer to an object that is dependent on another quantified object, in which case it will bring forward both quantifiers into the interpretation of the new sentence (e.g., "What is the silicon content of each volcanic sample?" "What is its magnesium concentration?"). It also handles certain cases of anaphora where only part of the intensional description of a previous phrase is reused (e.g., "What is the concentration of silicon in breccias?" "What is it in volcanics?"). However, this facility contains a number of loose ends. One of the most serious is that only the phrases typed in by the user are available for anaphoric reference, while the potential antecedents implied by the responses of the system are not (responses were usually not expressed in English, and in any case were not entered into the discourse directory). Anaphoric reference in general contains some very deep problems, some of which are revealed in LUNAR. Nash-Webber (1976, 1977), Nash-Webber and Reiter (1977), and Webber (1978) discuss these problems in detail.

10.7 Ill-Formed Input and Partial Interpretation

One of the problems that face a real user of a natural language understanding system is that not everything that he tries to say to the system is understandable to it. LUNAR tried to cope with this problem by having a grammar sufficiently comprehensive that it would understand everything a lunar geologist might ask about its data base. The system actually came fairly close to doing that. In other systems, such as the SOPHIE system of Brown and Burton (1975), this has been achieved even more completely. In a limited topic domain, this can be done by systematically extending the range of the system's understanding every time a sentence is encountered that is not understood, until eventually a virtual closure is obtained. Unfortunately, in less topic-specific systems, it is more difficult to reach this kind of closure, and in such cases it would be desirable for the system to provide a user with some partial analysis of his request to at least help him develop a model of what the machine does and does not understand.

LUNAR contains no facility for such partial understanding, although it does have a rudimentary facility to comment about modifiers that it does not understand in an otherwise understandable sentence and to notify the user of a phrase that it does not understand in a sentence that it has managed to parse but cannot interpret. Given the size of its vocabulary and the extensiveness of its grammar, there are large classes of sentences that LUNAR can parse but not understand. For these, LUNAR will at least inform the user of the first phrase that it encounters that it cannot understand. However, it cannot respond to questions about its range of understanding or be of much help to the user in finding out whether (and, if so, how) one can rephrase a request to make it understandable. More seriously, if a sentence fails to parse (a less common occurrence, but not unusual), LUNAR provides only the cryptic information that it could not parse the input. The reason for this is as follows.

If the user has used words that are not in its dictionary, LUNAR of course informs him of this fact and the problem is clear. If, however, the user has used known words in a way that does not parse, all LUNAR knows is that it has tried all of its possible ways to parse the input and none of them succeeded. In general, the parser has followed a large number of alternative parsing paths, each of which has gotten some distance through the input sentence before reaching an inconsistency. LUNAR in fact keeps track of each blocked path, and even knows which one of them has gotten the farthest through the sentence. However, experience has shown that there is no reason to expect this longest partial parse path to be correct. In general, the mistake has occurred at some earlier point, after which the grammar has continued to fit words into its false hypothesis for some unknown distance before an inconsistency arises. Beyond simply printing out the words used in this longest path (letting the user guess what grammatical characteristic of his sentence was unknown to the computer) there is no obvious solution to this problem. In this respect, a language with a deterministic grammar has an advantage over natural English, since there will only be one such parse path. In that case, when the parser blocks, there is no question about which path was best.

Note that there is no problem here in handling any particular case or anticipated situation. Arbitrary classes of grammatical violations can be anticipated and entered into the grammar (usually with an associated penalty to keep them from interfering with completely grammatical interpretations). Such sentences will no longer be a problem. What we are concerned with here requires a system with an understanding of its own understanding, and an ability to converse with a user about the meaning and use of words and constructions. Such a system would be highly desirable, but is far from realization at present. The grammar information system discussed above, which knows about its own grammar and can talk about states and transitions in the grammar, is a long way from being able to help a user in this situation.

One technique from the HWIM speech understanding system (Woods *et al.*, 1976) that could help in such a situation is to find maximally consistent islands in the word string using a bidirectional ATN parser that can parse any fragment of a correct sentence from the middle out. One could then search in the regions where such islands abut or overlap for possible transitions that could connect the two.

A special case of the ungrammatical sentence problem is the case of a mistyped word. If the misspelling results in an unknown word, then the problem is simple; when LUNAR informs the user of an unknown word, it also gives him the opportunity to change it and continue. However, if the misspelling results in another legal word, then the system is likely to go into the state discussed above, where all parsing paths fail and there is little the system can say about what went wrong. In this case, the user can probably find his mistake by checking the sentence he has typed, but sometimes a mistake will be subtle and overlooked. Again, some of the techniques from the HWIM system could be used here. Specifically, HWIM's dictionary look-up is such that it finds all words that are sufficiently similar to the input acoustics and provides multiple alternatives with differing scores, depending on how well they agree with the input. An identical technique can enumerate possible known words that could have misspellings corresponding to the typed input, with scores depending on the likelihoods of those misspellings. These alternatives would then sit on a shelf to be tried if no parsing using the words as typed were found.

10.8 Intensional Inference

As discussed previously, the LUNAR prototype deals only with extensional inferences, answering questions with quantifiers by explicitly enumerating the members of the range and testing propositions for individual members. LUNAR contains a good set of techniques for such inference, such as the use of general enumeration functions and smart quantifiers. However, although this is a very efficient mode of inference, it is not appropriate for many types of questions. The ability to deal with more complex types of data entities, even such specialized things as descriptions of shape and textural features of the lunar samples, will require the use of intensional inference procedures. For this reason, LUNAR's MRL was designed to be compatible with both intensional and extensional inference. Intensional inference is necessary for any type of question whose answer requires inference from general facts, rather than mere retrieval or aggregation of low-level observations. In particular, it is necessary in any system that is to accept input of new information in anything other than a rigid stylized format.

Although LUNAR contained some rudimentary facilities for adding new lines to its chemical analysis data base and for editing such entries, it contained no facility for understanding, storing, or subsequently using general facts and information. For example, a sentence such as "All samples contain silicon" is interpreted by LUNAR as an assertion to be tested and either affirmed or denied. It is not stored as a fact to be used subsequently. However, there is nothing in LUNAR's design that prohibits such storage of facts. In particular, a simple PRERULE for declarative sentences with a right-hand side (PRED (STORE (# 0 SRULES))) could generate interpretations that would store facts in an intensional data base (where STORE is assumed to be a function that stores facts in an intensional data base).

The function STORE could interface to any mechanical inference system to store its argument as an axiom or rule. For example, with a resolution theorem proving system such as Green's QA3 (Green, 1969), STORE could transform its argument from its given (extended) predicate calculus form into clause form and enter the resulting clauses into an indexed data base of axioms. TEST could then be extended to try inferring the truth of its argument proposition from such axioms either prior to, or after, attempting to answer the question extensionally. TEST could in fact be made smart enough to decide which mode of inference to try first on the basis of characteristics of the proposition being tested. Moreover, procedures defining individual predicates and functions could also call the inference component directly. For example, the predicate ABOUT that relates documents to topics could call the inference facility to determine whether a document is about a given topic due to one of its stored topics subsuming or being subsumed by the one in question.

The incorporation of intensional inference into the LUNAR framework is thus a simple matter of writing a few interfacing functions to add axioms to, and call for inferences from, some mechanical inference facility (assuming one has the necessary inference system). The problems of constructing such an inference facility to efficiently handle the kinds of inferences that would generally be required is not trivial, but that is another problem beyond the scope of this paper. A number of other natural language systems have capabilities for natural language input of facts (e.g., Winograd, 1972), but few have very powerful inference facilities for their subsequent use.

Among the shifts in emphasis that would probably be made in a semantic interpretation system to permit extensive intensional inference would be increasing attention to the notational structure of intensional entities to make them more amenable to inspection by various computer programs (as opposed to being perspicuous to a human). The effectiveness of the MRL used in LUNAR derives from its overall way of decomposing meanings into constituent parts, but is not particularly sensitive to notational variations that preserve this decomposition. When such MRL expressions are used as data objects by intensional processors, internal notational changes may be desired to facilitate such things as indexing facts and rules, relating more general facts to more specific ones, and making the inspection of MRL expressions as data objects more efficient for the processes that operate on them. In particular, one might want to represent the MRL expressions in some network form such as that described in Woods (1975b) to make them accessible by associative retrieval.

However, whatever notational variations one might want to adopt for increasing the efficiency of intensional processing, it should not be necessary, and is certaintly not desirable, to sacrifice the fundamental understanding of the semantics of the notation and the kinds of structural decompositions of meanings that have been evolved in LUNAR and her sister systems.

11. Syntactic/Semantic Interactions

A very important question, for which LUNAR's techniques are clearly not the general answer, has to do with the relative roles of syntactic and semantic information in sentence understanding. Since this is an issue of considerable complexity and confusion, I will devote the remainder of this paper to discussing the issues as I currently understand them.

The question of how syntax and semantics should interact is one that has been approached in a variety of ways. Even the systems discussed above contain representatives of two extreme approaches. LUNAR exemplifies one extreme: it produces a complete syntactic representation which is only then given to a semantic interpretation component for interpretation. TRIPSYS, on the other hand, combines the entire process of parsing and semantic interpretation in a grammar that produces semantic interpretations directly without any intermediate syntactic representation.

Before proceeding further in this discussion, let me first review the role of syntactic information in the process of interpretation.

11.1 The Role of Syntactic Structure

The role of a syntactic parsing in the overall process of interpreting the meaning of sentences includes answering such questions as "What is the subject noun phrase?", "What is the main verb of the clause?", "What determiner is used in this noun phrase?", etc.—all of this is necessary input information for the semantic interpretation decisions. Parsing is necessary to answer these questions because, in general, the answers cannot be determined by mere local tests in the input string (such as looking at the following or preceding word). Instead, such answers must be tentatively hypothesized and then checked out by discovering whether the given hypothesis is consistent with some complete analysis of the sentence. (The existence of "garden path" sentences whose initial portion temporarily misleads a reader into a false expectation about the meaning are convincing evidence that such decisions cannot be made locally.)

Occasionally, the interpretation of a sentence depends on which of several alternative possible parsings of the sentence the user intends (i.e., the sentence is ambiguous). In this case the parser must perform the case analysis required to separate the alternative possibilities so they can be considered individually. A syntactic parse tree, as used in LUNAR and similar systems, represents a concise total description that answers all questions about the grouping and interrelationships among words for a particular hypothesized parsing of a sentence. As such, it represents an example of what R. Bobrow (Bobrow and Brown, 1975) calls a "contingent knowledge structure," an intermediate knowledge structure that is synthesized from an input to summarize fundamental information from which a large class of related questions can then be efficiently inferred. In general, there is an advantage to using a separate parsing phase to discover and concisely represent these syntactic relationships, since many different semantic rules may ask essentially the same questions. One would not want to duplicate the processing necessary to answer them repeatedly from scratch.

In addition to providing a concise description of the interrelationships among words, the parse trees can serve an additional role by providing levels of grouping that will control the semantic interpretation process, assigning nodes to each of the phrases that behave as modular constituents of the overall semantic interpretation. The semantic interpreter then walks this tree structure, assigning interpretations to the nodes corresponding to phrases that the parser has grouped together. The syntax trees assigned by the grammar thus serve as a control structure for the semantic interpretation.

For historical reasons, LUNAR's grammar constructed syntactic representations as close as possible to those that were advocated at the time by transformational linguists as deep structures for English sentences (Stockwell *et al.*, 1968). The complex patterns of semantic rules in LUNAR and the multiple-phase interpretation are partly mechanisms that were designed to provide additional control information that was not present in those tree structures. An alternative approach could have been to modify the syntactic structures to gain the same effect (see below). The approach that was taken provides maximum flexibility for applying a set of semantic interpretation rules to an existing grammar. It also provides a good pedagogical device for describing interpretation rules and strategies, independent of the various syntactic details that stand between the actual surface word strings and the parse structures assigned by the grammar. However, the use of such powerful rules introduces a cost in execution time that would not be required by a system that adapted the grammar more to the requirements of semantic interpretation.

11.2 Grammar Induced Phasing of Interpretation

As mentioned above, most of the control of multiple phase interpretation that is done in LUNAR by means of successive calls to the interpreter with different TYPEFLAGS could be handled by having the parser assign a separate node for each of the phases of interpretation. If this were done, the phasing of interpretation would be governed entirely by the structure of the tree. For example, one could have designed a grammar to assign a structure to negated sentences that looks something like

```
S DCL
  NEG
  S     NP NPR S10046
        VP V CONTAIN
        NP DET NIL
           N SILICON
           NU SG
```

instead of

```
S DCL
  NEG
  NP    NPR S10046
  VP    V CONTAIN
        NP DET NIL
           N SILICON
           NU SG.
```

In such a structure, there is a node in the tree structure to receive the interpretation of the constituent unnegated sentence, and thus the separate phasing of the PRERULES and the SRULES used in LUNAR would be determined by the structure of the tree. Similarly, noun phrases could be structured something like

```
NP DET  THE
   NU   SG
   NOM  NOM ADJ  N SILICON
            NOM  N CONCENTRATION
        PP  PREP IN
            NP NPR S10046
```

instead of the structure

```
NP DET  THE
   ADJ  N SILICON
   N    CONCENTRATION
   NU   SG
   PP  PREP IN
       NP NPR S10046
```

which is used in the LUNAR grammar. In such a structure, the nested NOM phrases would receive the interpretation of the head noun plus modifiers by picking up modifiers one at a time.

It is not immediately obvious, given LUNAR's separation of syntactic and semantic operations, which of the two ways of introducing the phasing is most efficient. Introducing phasing via syntax requires it to be done without the benefit of some of the information that is available at interpretation time, so that there is the potential of having to generate alternative syntactic representations for the interpreter to later choose between. On the other hand, doing it with the semantic interpretation rules requires extra machinery in the interpreter (but does not seem to introduce much extra run-time computation).

One might argue for the first kind of structure in the above examples on syntactic grounds alone. If this is done, then the efficiency issue just discussed is simply one more argument. If it turns out that the preferred structure for linguistic reasons is also the most efficient for interpretation, that would be a nice result. Whether this is true or not, however, is not clear to me at present.

11.3 Semantic Interpretation while Parsing

The previous discussion illustrates some of the disadvantages of the separation of parsing and semantic interpretation phases in the LUNAR system. The discussion of placement of movable modifiers illustrates another. In general, there are a variety of places during parsing where the use of semantic information can provide guidance that is otherwise not available, thus limiting the number of alternative hypothetical parse paths considered by the parser. It has frequently been argued that performing semantic interpretation during parsing is more efficient than performing it later by virtue of this pruning of parse paths. However, the issue is not quite as simple as this argument makes it appear. Against this savings, one must weigh the cost of doing semantic interpretation on partial parse paths that will eventually fail for syntactic reasons. Which of the two approaches is superior in this respect depends on (1) the relative costs of doing semantic versus syntactic tests and (2) which of these two sources of knowledge provides the most constraint. Both of these factors will vary from one system to another, depending on the fluency of their grammars and the scope of their semantics.

At one point, a switch was inserted in the LUNAR grammar that would call for the immediate interpretation of any newly formed constituent rather than wait for a complete parse tree to be formed. This turned out not to have an efficiency advantage. In fact, sentences took longer to process (i.e., parse and interpret). This was due in part to the fact that LUNAR's grammar did a good job of selecting the right parse without semantic guidance. In such circumstances, semantic interpretations do not help to reject incorrect paths. Instead, they merely introduce an extra cost due to interpretations performed on partial parse paths that later fail. Moreover, given LUNAR's rules, there are constituents for which special interpretations are required by higher constructions (e.g., with TYPEFLAG SET or TOPIC). Since bottom-up interpretation may not know how a higher construction will want to interpret a given constituent, it must either make as assumption (which may usually be right, but occasionally will have to be changed), or else make all possible interpretations. Either case will require more interpretation than waiting for a complete tree to be formed and then doing only the interpretation required. All of these considerations make semantic interpretation during parsing less desirable unless some positive benefit of early semantic guidance outweighs these costs.

11.4 Top-Down versus Bottom-Up Interpretation

In the experiment described above, in which LUNAR was modified to perform bottom-up interpretation during parsing, the dilemma of handling context-dependent interpretations was raised. In those experiments, the default assumption was made to interpret every noun phrase with TYPEFLAG NIL during the bottom-up phase. In cases where a higher construction required some other interpretation, reinterpretation was called for at that point in the usual top-down mode. Since LUNAR maintains a record of previous interpretations that have been done on a node to avoid repeating an interpretation, it was possible to efficiently use interpretations that were made bottom-up when they happened to be the kind required, while performaing new ones if needed.

An alternative approach to this problem of bottom-up interpretation in context is to make a default interpretation that preserves enough information so that it can be modified to fit unexpected contexts without actually having to redo the interpretation. This would be similar to the kind of thing that SETGEN (in the right-hand side of the D:SET rule) does to the quantifiers it picks up to turn them into UNIONs. In the HERMES grammar (Ash *et al.*, 1977), R. Bobrow uses this approach, which he calls "coercion" (intuitively, forcing the interpretation of a constituent to be the kind that is expected). In this case, when the higher construction wants the interpretation of a constituent in some mode other than the one that has been already done, it asks whether the existing one can be coerced into the kind that it wants rather than trying to reinterpret the original phrase.

Many of these questions of top-down versus bottom-up interpretation, syntax-only parsing before semantic interpretation or vice versa (or both together), do not have clear cut answers. In general, there is a tension between doing work on a given portion of a sentence in a way that is context free (so that the work can be shared by different alternative hypotheses at a higher level) and doing it in the context of a specific hypothesis (so that the most leverage can be gained from that hypothesis to prune the alternatives at the lower level). It is not yet clear whether one of the extremes or some intermediate position is optimal.

11.5 Pragmatic Grammars

One thing that should be borne in mind when discussing the role of grammars is that it is not necessary that the grammar characterize exactly those sentences that a grammarian would consider correct. The formal grammar used by a system can characterize sentences as the user would be likely to say them, including sentences that a grammarian might call ungrammatical. For example, LUNAR accepts isolated noun phrases as acceptable utterances, implicitly governed by an operator "give me."

In the classical division of problems of meaning into the areas of syntax, semantics, and pragmatics, the latter term is used to denote those aspects of meaning determined not by general semantic rules, but as aspects of the current situation, one's knowledge of the speaker, etc. For example, in situations of irony, a speaker says exactly the opposite of what he means. Likewise, certain apparent questions should in fact be interpreted as commands or as other requests (e.g., "Do you have the time?" is usually a "polite" way of asking "What time is it?"). Moreover, certain ungrammatical utterances nevertheless have a meaning that can be inferred from context. In general, the ultimate product of language understanding is the pragmatic interpretation of the utterance in context. This interpretation, while not necessarily requiring a syntactically and semantically correct input sentence, nevertheless depends on an understanding of normal syntax and semantics.

In LUNAR, there is no systematic treatment of pragmantic issues, although in some cases, pragmatic considerations as well as semantic ones were used in formulating its interpretation rules. For example, the rule that interprets the head "analysis," when it finds no specification of the elements to be measured, makes a default assumption that the major elements are intended. This is due to the pragmatic fact that (according to our geologist informant) this is what a geologist would want to see if he made such a request, not because that is what the request actually means. In this way, LUNAR can handle a small number of anticipated pragmantic situations directly in its rules.

In TRIPSYS, a small step toward including pragmatics in the grammar was taken. The TRIPSYS grammar takes into account not only semantic information such as class membership and selectional restrictions of words, but also pragmatic information. This includes factual world knowledge such as what cities are in which states, actual first and last names of people, and discourse history information, such as whether appropriate referents exist for anaphoric expressions. The TRIPSYS system is only beginning to explore these issues, and has not begun to develop a general system for pragmatic interpretation. Much more work remains to be done in this area, and interest in it seems to be building as our mastery of the more basic syntactic and semantic issues matures.

The "pragmatic" grammar of TRIPSYS is only one exploration of a philosophy of combined syntactic and semantic grammars that has arisen independently in several places. Other similar uses of ATN or ATN-like grammars combining syntactic and semantic (and possibly pragmatic) information are the "Semantic Grammars" of Burton (1976), the "Performance Grammars" of Robinson (1975), the SHRDLU system of Winograd (1972), and the HERMES grammar of R. Bobrow (Ash *et al.*, 1977).

11.6 Semantic Interpretation in the Grammar

In separating parsing and semantic interpretation into two separate processes (whether performed concurrently or in separate phases), LUNAR gains several advantages and also several disadvantages. On the positive side, one obtains a syntactic characterization of a sizable subset of English that is independent of a specific topic domain and hence transferable to other applications. All of the domain-specific information is contained in the dictionaries and the semantic interpretation rules. On the other hand, there is a conceptual expense in determining what syntactic structure to use for many of the less standard constructions. One would like such structures to be somehow motivated by linguistic principles and yet, at the same time, have them facilitate subsequent interpretation. In many cases, the desired interpretation is more clear to the grammar designer than is a suitable syntactic representation. In a number of situations, such as those discussed previously for handling wh-questions with conjunction reduction and for handling averages, I have found it desirable to change what had initially seemed a suitable syntactic representation in order to facilitate subsequent semantic interpretation. If semantic interpretations were to be produced directly by the grammar instead of using an intermediate syntactic representation, then such problems would be avoided.

The integration of semantic interpretation rules into the grammar could be done in a number of ways, one of which would be to develop a rule compiler that would use the templates of rules such as LUNAR's to determine where in the grammar to insert the rule. Another would be to write the interpretation rules into the grammar in the first place. This latter is the approach that is taken in the TRIPSYS system. It seems clearly an appropriate thing to do for such rules as the PRERULES for sentences and the DRULES for noun phrases, where the principal information used is largely syntactic. For the equivalent of SRULES, NRULES, and RRULES, writing specific rules into the grammar would make the grammar itself more topic-specific than one might like. However, writing generalized rules that apply to large classes of words, using information from their dictionary entries for word-specific information such as case frames, selectional restrictions, permitted prepositions, and corresponding MRL translations, should produce a grammar that is relatively topic-independent. This is the approach taken by Robinson (1975) and by R. Bobrow (Ash *et al.*, 1977).

Integrating semantic interpretation with a grammar is not an obvious overall improvement, since by doing so one gives up features as well as gaining them. For example, as discussed earlier the "advantage" of using semantic interpretation to prune parse paths is not always realized. However, there are some other efficiencies of the combined syntactic/semantic grammars that have nothing to do with pruning. One of these is the avoidance of pattern-matching.

One of the costs of the separate semantic interpretation phase used in LUNAR is the cost of pattern-matching the rules. Much of this effort is redundant since the various pieces of information that are accessed by the rules were mostly available in registers during the parsing process. From here they were packaged up by actions in the grammar into the parse tree structures that are passed on to the interpreter. The pattern-matching in the interpreter recovers these bindings so that the right-hand side of the rule can use them. If the right-hand side schema of the rule could be executed while these bindings were still available during the parsing process, considerable computation could be avoided. Moreover, much of the syntactic information that is checked in the rules is implicitly available in the states of the grammar by virtue of the fact that the parser has reached that state (and more of that information could be put into the states if desired). Thus, in many cases, much of the testing that goes on in the pattern-matching of rules would be avoided if the right-hand side of the rule, paired with whatever semantic tests are required, were inserted as an action at the appropriate points in the grammar.

For example, at certain points in the parsing, the grammar would know that it had enough information to construct the basic quantifier implied by the determiner and number of a noun phrase. At a later point, it would know all of the various modifiers that are being applied to the head noun. As the necessary pieces arrive, the interpretation can be constructed incrementally.

The effectiveness of this kind of combined parser/interpreter depends partly on the discovery that the kinds of associations of REFs to constituent nodes that are made by LUNAR's rules are usually references to direct constituents of the node being interpreted. Thus, they correspond closely to the constituents that are being held in the registers by the ATN grammar during its parsing. The original semantic rule format was designed to compensate for rather large potential mismatches between the structure that a grammar assigns and the structure that the interpreter would like to have (since it was intended to be a general facility applicable to any reasonable grammar). When a grammar is specifically designed to support the kinds of structures required by the interpreter, this very general "impedance matching" capability of the rules is not required.

Thus, when fully integrated with the parsing process in an ATN grammar, the process of semantic interpretation requires fewer computation steps than when it is done later in a separate phase. This clearly has a bearing on the previous discussion of the relative costs of syntactic and semantic processing. Other advantages of this kind of integrated parsing and interpretation process is that the single nondeterminism mechanism already present in the parser can be used to handle alternative interpretations of a given syntactic structure, without requiring a separate facility for finding and handling multiple rule matches. This not only eliminates extra machinery from the system, but appears to be more efficient. It also permits a more flexible interaction between the ranking of alternative syntactic choices and the ranking of alternative choices in semantic interpretation.

A disadvantage of this integrated approach is that the combined syntactic/semantic grammar is much more domain-specific and less transportable unless clear principles for separating domain-specific from general knowledge are followed. Moreover, the fact that a given semantic constituent can be found in different places by different arcs in the grammar seems to require separate consideration of the same semantic operations at different places in the grammar.

11.7 Generating Quantifiers while Parsing

The generation of separate SEM's and QUANT's when performing interpretation while parsing appears to complicate the integration of the semantic interpretation into the grammar, but in fact is not difficult. One can stipulate that any constituent parsed will return a structure that contains both a SEM and a QUANT as currently assigned by the INTERP function in LUNAR. The parsing at the next higher level in the grammar will then accumulate the separate QUANTs from each of the constituents that it consumes, give them to a SORTQUANT function to determine the order of nesting, and construct the interpretation of the phrase being parsed out of the SEM's of the constituent phrases. All of the quantifier passing operations described previously can be carried out during the parsing with little difficulty.

One advantage of this procedure is that the job of SORTQUANT is simplified by the fact that the quantifiers will be given to it in surface structure order rather than in some order determined by the deep structure assigned by the grammar. LUNAR's SORTQUANT function has to essentially reconstruct surface word order.

12. Conclusions

The LUNAR prototype marks a significant step in the direction of fluent natural language understanding. Within the range of its data base, the system permits a scientist to ask questions and request computations in his own natural English in much the same form as they arise to him (or at least in much the same form that he would use to communicate them to another human being). However, although the LUNAR prototype exhibits many desired qualities, it is still far from fully achieving its goal. The knowledge that the current system contains about the use of English and the corresponding meanings of words and phrases is very limited outside the range of those English constructions that pertain to the system's data base of chemical analysis data. This data base has a very simple structure; indeed it was chosen as an initial data base because its structure was simple and straightforward. For less restricted applications, such systems will require much greater sophistication in both the linguistic processing and the underlying semantic representations and inference mechanisms.

In this paper, I have presented some of the solutions that were developed in LUNAR (and several related systems) for handling a variety of problems in semantic interpretation, especially in the interpretation of quantifiers. These include a meaning representation language (MRL) that facilitates the uniform interpretation of a wide variety of linguistic constructions, the formalization of meanings in terms of procedures that define truth conditions and carry out actions, efficient techniques for performing extensional inference, techniques for organizing and applying semantic rules to construct meaning representations, and techniques for generating higher quantifiers during interpretation. These latter include methods for determining the appropriate relative scopes of quantifiers and their interactions with negation, and for handling their interactions with operators such as "average." Other techniques are described for post-interpretive query optimization and for displaying quantifier dependencies in output.

I have also discussed a number of future directions for research in natural language understanding, including some questions of the proper relationship between syntax and semantics, the partial understanding of "ungrammatical" sentences, and the role of pragmatics. In the first area especially, I have discussed a number of advantages and disadvantages of performing semantic interpretation during the parsing process, and some aspects of the problem of separating domain specific from general knowledge.

As discussed in several places in the paper, there are a variety of loose ends and open problems still to be solved in the areas of parsing and semantic interpretation. However, even in the four systems discussed here, it is apparent that as the system becomes more ambitious and extensive in its scope of knowledge, the need for pragmatic considerations in selecting interpretations becomes increasingly important. I believe that, as a result of increasing understanding of the syntactic and semantic issues derived from explorations such as the LUNAR system, the field of computational linguistics is now reaching a sufficient degree of sophistication to make progress in a more general treatment of pragmatic issues. In doing so, it will become much more concerned with general issues of plausible inference and natural deduction, moving the field of language understanding in the direction of some of the other traditional areas of artificial intelligence research, such as mechanical inference and problem solving.

Acknowledgments

Work described in this paper has been supported in part by the following contracts and grants: National Science Foundation Grant GS-2301; NASA Contract No. NAS9-1115; ARPA Contracts N00014-75-C-0533, N00014-77-C-0378; and ONR Contract N00014-77-C-0371.

References

Ash, W., Bobrow, R., Grignetti, M., and Hartley, A. (1977). "Intelligent On-Line Assistant and Tutor System," Final Tech. Rep., Rep. No. 3607. Bolt Beranek and Newman, Cambridge, Massachusetts.

Bobrow, D. G., Murphy, D. P., and Teitelman, W. (1968). "The BBN-LISP System," BBN Rep. No. 1677. Bolt Beranek and Newman, Cambridge, Massachusetts.

Bobrow, R. J., and Brown, J. S. (1975). Systematic understanding: Synthesis, analysis, and contingent knowledge in specialized understanding systems. *In* "Representation and Understanding: Studies in Cognitive Science" (D. Bobrow and A. Collins, eds.), pp. 103–129. Academic Press, New York.

Bohnert, H. G., and Backer, P. O. (1967). "Automatic English-to-Logic Translation in a Simplified Model. A Study in the Logic of Grammar," IBM Res. Pap. RC-1744. IBM Research, Yorktown Heights, New York.

Brown, J. S., and Burton, R. R. (1975). Multiple representations of knowledge of tutorial reasoning. *In* "Representation and Understanding: Studies in Cognitive Science" (D. Bobrow and A. Collins, eds.), pp. 311–349. Academic Press, New York.

Burton, R. (1976). "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems," Rep. No. 3453. Bolt Beranek and Newman, Cambridge, Massachusetts.

Carnap, R. (1964a). Foundations of logic and mathematics. *In* "The Structure of Language: Readings in the Philosophy of Language" (J. Fodor and J. Katz, eds.), pp. 419–436. Prentice-Hall, Englewood Cliffs, New Jersey.

Carnap, R. (1964b). "Meaning and Necessity." Univ. of Chicago Press, Chicago, Illinois.

Chomsky, N. (1965). "Aspects of the Theory of Syntax." MIT Press, Cambridge, Massachusetts.

Green, S. (1969). "The Application of Theorem Proving to Question-Answering Systems," Tech. Rep. CS 138. Stanford University Artificial Intelligence Project, Stanford, California.

Nash-Webber, B. L. (1976). Semantic interpretation revisited. Presented at *1976 Int. Conf. Comput. Linguist. (COLING-76), Ottawa.* (Available as BBN Rep. No. 3335. Bolt Beranek and Newman, Cambridge, Massachusetts.)

Nash-Webber, B. L. (1977). Inference in an approach to discourse anaphora. *Proc 8th Ann. Meet. North Eastern Linguist. Soc. (NELS-8)* (K. Ross, ed.), pp. 123–140. University of Massachusetts, Amherst. (also as Tech. Rep. No. 77. Center for the Study of Reading, University of Illinois, Urbana.)

Nash-Webber, B. L., and Reiter, R. (1977). Anaphora and logical form: On formal meaning representations for English. *Proc. Int. J. Conf. Artif. Intell., 5th, MIT, Cambridge, Mass.* pp. 121–131. (Also as Tech. Rep. No. 36. Center for the Study of Reading, University of Illinois, Urbana and Bolt Beranek and Newman, Cambridge, Massachusetts.)

OAG (1966). "Official Airline Guide," Quick Reference North American Edition. Standard reference of the Air Traffic Conference of America.

Reiter, R. (1977). "An Approach to Deductive Question-Answering," Rep. No. 3649. Bolt Beranek and Newman, Cambridge, Massachusetts.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. ACM* **12,** 23–41.

Robinson, J. J. (1975). Performance grammars. *In* "Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium" (D. R. Reddy, ed.), pp. 401–427. Academic Press, New York.

Simmons, R. F. (1965). Answering English questions by computer: A survey. *Commun. ACM* **8**(1), 53–70.

Stockwell, R. P., Schacter, P., and Partee, B. H. (1968). "Integration of Transformational Theories on English Syntax," Rep. ESD-TR-68-419. Electronic Systems Division, L. G. Hanscom Field, Bedford, Massachusetts.

Webber, B. L. (1978). A formal approach to discourse anaphora. Ph.D. Thesis, Harvard University, Cambridge, Massachusetts.

Winograd, T. (1972). "Understanding Natural Language." Academic Press, New York.

Woods, W. A. (1967). "Semantics for a Question-Answering System," Rep. NSF-19. Harvard University Computation Laboratory, Cambridge, Massachusetts. (Available from NTIS as PB-176-548.)

Woods, W. A. (1968). Procedural semantics for a question-answering machine. *AFIPS Natl. Comput. Conf. Expo., Conf. Proc.* **33,** 457–471.

Woods, W. A. (1969). "Augmented Transition Networks for Natural Language Analysis," Rep. No. CS-1. Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts. (Available from NTIS as Microfiche PB-203-527.)

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Commun. ACM* **13,** 591–602.

Woods, W. A. (1973a). An experimental parsing system for transition network grammars. *In* "Natural Language Processing" (R. Rustin, ed.), pp. 111–154. Algorithmics Press, New York.

Woods, W. A. (1973b). Progress in natural language understanding: An application to LUNAR geology. *AFIPS Natl. Comput. Conf. Expo., Conf. Proc.* **42,** 441–450.

Woods, W. A. (1973c). Meaning and machines. *In* "Computational and Mathematical Linguistics" (A. Zampolli, ed.), pp. 769–792. Leo S. Olschki, Florence.

Woods, W. A. (1975a). Syntax, semantics, and speech. *In* "Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium" (D. R. Reddy, ed.), pp. 345–400. Academic Press, New York.

Woods, W. A. (1975b). What's in a link: Foundations for semantic networks. *In* "Representation and Understanding: Studies in Cognitive Science" (D. Bobrow and A. Collins, eds.), pp. 35–82. Academic Press, New York.

Woods, W. A., Kaplan, R. M., and Nash-Webber, B. (1972). "The Lunar Sciences Natural Language Information System: Final Report," BBN Rep. No. 2378. Bolt Beranek and Newman, Cambridge, Massachusetts.

Woods, W. A. , Bates, M., Brown, G., Bruce, B., Cook, C., Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J., and Zue, V. (1976). "Speech Understanding Systems—Final Report, 30 October 1974 to 29 October 1976," BBN Rep. No. 3438, Vols. I–V. Bolt Beranek and Newman, Cambridge, Massachusetts.