CONSTRUCT: IN SEARCH OF A THEORY OF MEANING

by

R. L. Smith, N. W. Smith, and F. L. Rawson

TECHNICAL REPORT NO. 238

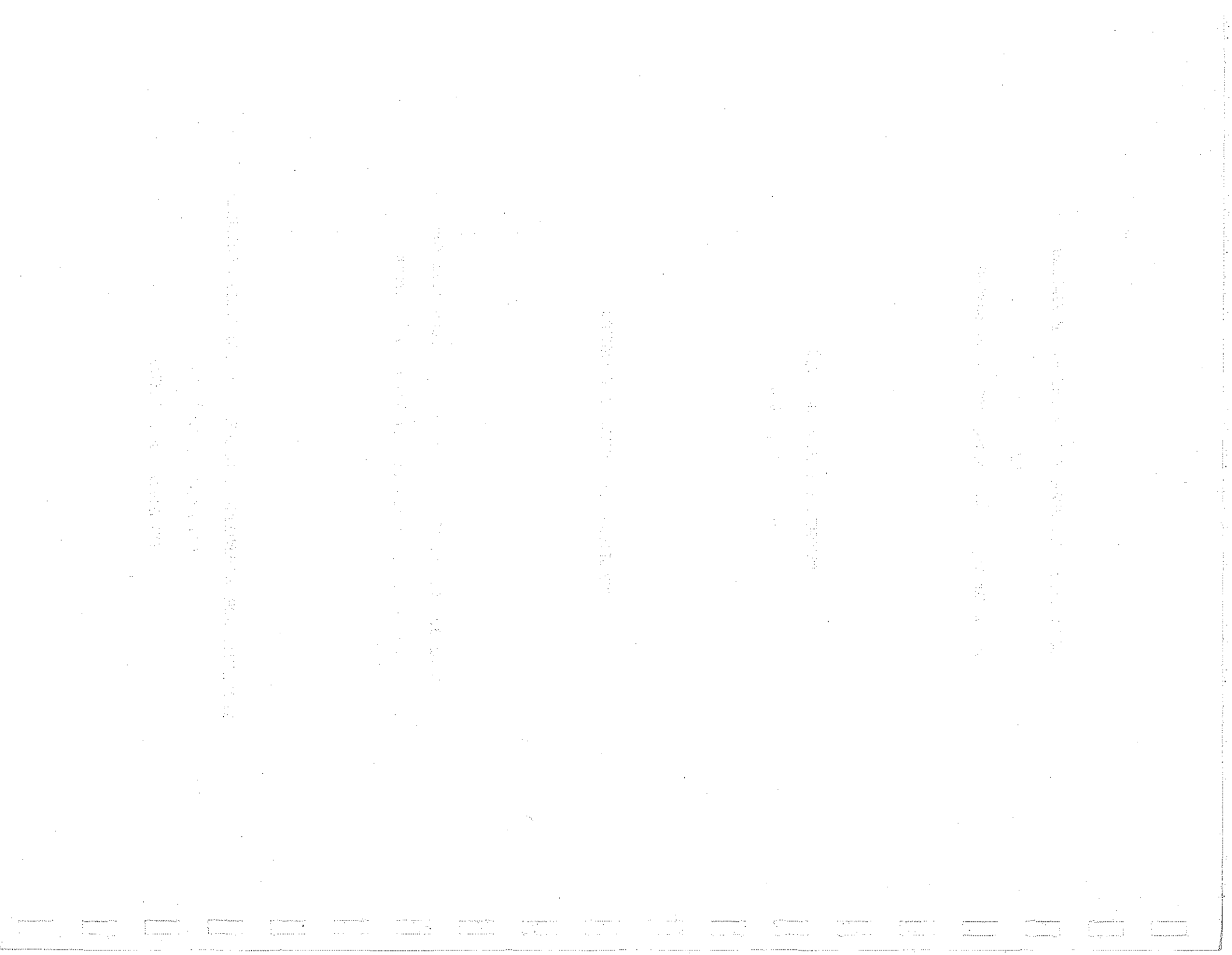
October 25, 1974

PSYCHOLOGY AND EDUCATION SERIES

INSTITUTE FOR MATHEMATICAL STUDIES IN THE SOCIAL SCIENCES

STANFORD UNIVERSITY

STANFORD, CALIFORNIA



CONSTRUCT:
In Search of a Theory of Meaning*

R. L. Smith
IMSSS, Stanford, Ca.

N. W. Smith
SUMEX, Stanford, Ca.

F. L. Rawson
IBM, San Jose, Ca.

Abstract: CONSTRUCT is a question-answering system for elementary mathematical language using natural language input. The primary goal of the system is to reach a better understanding of the relationship between the syntactic and semantic components of natural language. The *meaning conditions* for sentences are given in terms of constructive set theory. The notion of a *semantic transformation* is introduced, and it is suggested how to analyze this notion in terms of program schemata.

**Acknowledgements.* We would like to express our gratitude to Patrick Suppes, who defined our goals, criticized our ideas and implementation, and made available to us the computer facilities of the Institute for Mathematical Studies in the Social Sciences (IMSSS) at Stanford.

The opinions expressed herein are those of the authors and not of the United States Government or the International Business Machines Corporation.

Support for this work was obtained from the National Science Foundation under Grant EC 443X4. This work was done while all three authors were at IMSSS.

1 Introduction

In the history of computational linguistics, the first significant question asked was how to analyze syntax. Programs that were developed using only syntactic analysis were unsuccessful at actually understanding natural language and consequently the emphasis shifted to semantics. Subsequent programs using primarily semantic methods met with a beginning level of success. Since Winograd [22], in an effort to further refine this success, the basic question guiding artificial intelligence work has been sharpened to the question of how to best represent semantic information (this has been called "the representation problem"). (1) In its extreme form, the current emphasis on semantics excludes any consideration of syntax. A typical view is that of Schank [17], who claims about his system that "syntax has been denigrated to the status of something to use when all else fails".

Most of the recent discussion concentrates on the holistic nature of language and, rather than denying syntax a role, stresses the interaction of syntactic and semantic features of language and cautions against the possibility of making hard and fast divisions into "syntactic" and "semantic" problems. In actual programs, this view is implemented by the elimination of distinct phases of analysis corresponding to syntax and semantics. Instead both syntactic and semantic procedures are available which can call each other, and the results of each call determine how the analysis will proceed. Thus the result of both the extreme and the more common view of the relation between these two components of language is the exclusion of any serious discussion of the general systematic role that syntax plays in language understanding. An even more disturbing trend that has appeared in recent discussions is the assumption that no logically coherent theory of meaning for natural language is even possible. (2)

Certainly the early failures in mechanical translation and other types of purely syntactic analysis have shown that the traditional conception of syntax is weak; and efforts to use such logical techniques as resolution theorem proving have shown them to be inadequate for solving the problems of natural language inference. These failures are far enough behind us that we can perhaps restore some balance into the discussion.

We make two claims:

(1) Syntax is important to the understanding of natural language.

---

(1) The "semantic information" question is currently being phrased as a debate between *procedural* and *declarative* semantics.

(2) See for example Colby and Enea [2], Wilks [20].

(2) A logically coherent theory of meaning is not only possible, but necessary if computational linguistics is to proceed.

We shall, in the body of the paper, elaborate these claims, present a computer system (CONSTRUCT) that was designed to support these claims, and then suggest what kind of mathematical analysis is necessary to make progress in the area.

1.1 The Role of Syntax

The "denigration of syntax" seems to be associated with the belief that ordinary language is sufficiently less orderly and "rule-like" (3) than is required for formal analysis. On the other hand, Richard Montague [9] has suggested that there is no important theoretical difference between formal and natural languages. Despite the popularity of the informal approach, we think that Montague's position will be vindicated.

What is needed is a richer view of syntax that accounts for the appearance of disorder. We suggest that the correct analysis of the syntax of a language provides a method of discovering the computational control structure of the sentences in the language -- to use computer metaphors, the "program" that must be "executed" in order to answer a question, execute a command, or evaluate the truth value of a declarative. In order to answer the natural language questions asked of it, the computer must execute a program to compute the answer. Thus, the "semantics" of a sentence is a program generated by the "syntax".

The execution of such a "program" will in general involve steps that modify the program itself. These steps correspond to places where the syntax fails to be as orderly as first-order logic. We describe these situations as *semantic transformations*.

1.2 Towards a Theory of Meaning

The claim that English is best thought of as a "formal language" is motivated by the success in model theory, a part of formal logic. Alfred Tarski [19] characterized truth for the formal languages by saying that "a true sentence is one which says that the state of affairs is so and so, and the state of affairs indeed is so and so". (Pp. 153-155 of [19]) The programme of carrying this characterization of meaning through to natural languages involves giving a model structure that represents the world under discourse, and showing how to map from sentences (or sequences of sentences) to conditions of this model structure.

---

(3) See Wilks [20] for a discussion of this view.

2 The CONSTRUCT System

Before we discuss the above theories in greater detail, let us describe the CONSTRUCT system. CONSTRUCT is a flexible modular system for natural language processing. The CONSTRUCT program itself which is written in SAIL serves as the executive component of the system. It contains a built-in scanner and parser as well as all the routines for interfacing and coordinating the other modules of the system and also provides extensive facilities for on-line monitoring of the system. Depending on the actual modules used the CONSTRUCT system can serve many purposes and handle many diverse fragments of natural language. The implementation discussed in this paper is a question-answering system for elementary mathematics. (For more details on this system see N. Smith [18].)

In addition to the built-in scanner and parser, the current system is composed of 1) a dictionary of the words in the vocabulary and their lexical categories, 2) a special dictionary called the TRANSL dictionary of strings of words designated for special preprocessing, 3) a context free grammar which has 4) a semantic function associated with each rule of the grammar and 5) an evaluation program written in LISP.

2.1 Preprocessing

The operation of the system can be viewed as a four-stage process. The first stage is preprocessing which is conducted under the direction of the scanner. CONSTRUCT's scanner is similar to scanners found in compilers. It uses a table of break characters to separate the input into words. Next certain strings of words are eliminated or replaced. The TRANSL file contains a complete list of the strings to be substituted for and the substitution strings. Currently four types of substitutions are made: 1) abbreviations are TRANSL'd to the full word form, 2) synonyms are TRANSL'd to the most commonly used one of the group, 3) common phrases which have a meaning not strictly determined by the combination of the individual words are TRANSL'd to a single word representation, and 4) noise words are eliminated.

One important function of the TRANSL component is to provide pattern-recognition capabilities. Colby and Enea [2] use question-introducers as an example to explain their pattern recognition technique. Certain polite phrases that cannot be analyzed literally are used to introduce questions. Essentially the problem is to fill a syntactic slot with an unanalyzed phrase. Their rules for this are:

```
RULES OF SENTENCE =
   <QUESTION-INTRODUCER>: Q <NOUN-PHRASE>:N
                              ← (IS :N '*'?'* );
RULES OF QUESTION-INTRODUCER =
   COULD YOU TELL ME ←,
   WOULD YOU TELL ME ←,
   PLEASE TELL ME ←,
```

Our system handles question-introducers by including each such phrase in the TRANSL and processing the questions with the grammatical rules:

```
RULE1:  Q → QUESTION-INTRODUCER NP
  EX1:  Do you know the sum of 2 and 4?

RULE2:  Q → QUESTION-INTRODUCER INTERROGATIVE NP LINKVERB
  EX2:  Do you know what the factors of 12 are?
```

It is interesting to note that their system which is designed with pattern recognition as the paramount concern requires as many rules to deal with the question-introducer as ours does. We need one line in the TRANSL for each of the phrases as they need one rule for each, and we also need one grammatical rule for each sufficiently different construction of the question following the phrase. Our RULE1 is analogous to their rule and they would need to add RULE2 before they could handle EX2.

Use of a separate component for pattern recognition has four definite advantages. The first advantage is the speed and efficiency gained by an initial preprocessing phase which makes possible a shorter grammar. A second advantage to a separate module containing all the strings is the ease with which changes can be made to the strings and to the grammar itself. The third advantage is in the flexibility of the system. To change subject areas we can use the same basic grammar with a different TRANSL file while their grammar contains many rules dependent on the particular vocabulary of the subject area involved. Finally, the preprocessing approach has the advantage of clarity. The output of the scanner shows precisely which substitutions were made and how the input was standardized. In a typical pattern-recognition grammar, this information is buried in the parse tree.

After all of these substitutions have been made, the scanner uses the dictionary to form a new representation of the input by replacing each word by its lexical category. It is this representation which is passed to the parser.

2.2 Syntactic and Semantic Analysis

The second and third stages of processing proceed in parallel. The second stage is the syntactic parse of the input. The third stage is the creation of what we call a 'semantic construction' for the input. It has the form of a LISP S-expression which is ready for evaluation. Each rule of the grammar has an associated semantic function whose explicit arguments are the meanings of the elements on the right-hand-side of the rule. The evaluation of this function yields the meaning of the element on the left-hand-side of the rule. As the syntactic parse proceeds the "semantic parse" builds up the final semantic construction for the input from the semantic functions associated with each rule used in the syntactic parse.

Thus there is no interaction between "syntactic" and "semantic" routines at runtime. Many systems use an interactive process in which parsing begins but semantic routines can be called in to disambiguate certain phrases and the semantic routines in turn can call for further parsing of a piece of the input, etc. Instead of an interactive process of this type, our grammar is designed to obtain the semantic information from the syntactic parse. This is based on the assumption that a sentence contains two types of information. The first type of information is carried by the substantive or content words in the sentence. These words indicate which items in the data base or world model used by the evaluation program are to be dealt with in evaluating this particular sentence. The second type of information which can be obtained from the sentence is carried by both the functional words used in the sentence and the organization or grammatical structure of the sentence. This information tells us which of the evaluation procedures available are to be applied to the relevant items in the data base and also gives us the control structure to use in applying these functions or procedures.

Our grammar has been written to maximize the amount of this type of semantic information that can be obtained from the syntactic parse. The grammar is quite different from the usual grammars used for natural language processing which traditionally start with the rule

S → NP VP .

It would be virtually impossible to construct a semantic function to attach to this rule. Not only is this rule too general but it has already begun an incorrect breakdown of the sentence for semantic purposes. From this point on in the parse, the main noun phrase is separated from both the verb which will indicate what action to take with the noun phrase and from any clause in the VP following the verb. Another example of a series of traditional rules which lead to a parse which would have to be judged a poor parse on semantic grounds is:

```
NP → DETERMINER N
NP → NP PREPHRASE
NP → N
PREPHRASE → PREP NP
DETERMINER → ARTICLE
```

Consider the phrase "the factors of 4", which would be parsed by these rules as shown in Figure 1.

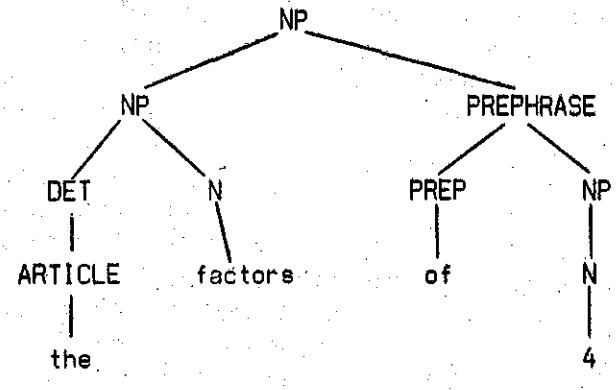


Figure 1. Tree for *the factors of 4*.

The information as to which type of determiner is used is not discovered until the lowest level in the parse and even then both the definite and indefinite article are lumped into one lexical category. This has no correspondence to the actual scope of the determiner and allows no point in the parse for assigning the particular function associated with each quantifier and article. In our system, each semantic function (except for the explicitly transformational ones, see Section 4 below) is assigned at a level where all the arguments to the function are available.

These rules create a similar problem with the preposition. Traditionally all prepositions have been grouped into one lexical category and parsed by the same grammatical rules. The semantic insight about "of" which is used by our system is that this preposition is used to designate a function, e.g., the FACTOR function and its argument(s), e.g., 4. Other examples are: the denominator of 1/2, the sum of 2 and 4, and the intersection of {a,b} and {a,c}. We use the semantic function APP for applying functions in the data base to their specified arguments. The traditional rules split the function from its argument in the parse tree and provide no rule with

which the APP function can be associated. We have assigned "of" its own lexical category and also created a lexical category FCN for nouns which are names of functions. Thus the rule is

```
NP → FCN \OF\ NP                    (APP ;1; ;3;)
```

[Note: The form *;n;* in the specification of the arguments to a semantic function refers to the *n*th element on the right-hand-side of the rule. These slots will be filled in by the semantic functions for those elements as the parse proceeds.]

As can be seen in this example, our grammar differs from traditional grammars in two basic ways -- first by assigning words "lexical" categories on the basis of their semantic role and second by making sure that each rule in the grammar has semantic relevance. This has the affect of considerably flattening the parse trees so that more of the context is considered at each level. However it should be made clear that we are not in the position of needing virtually a separate rule for each sentence. We do not create categories, either lexical or grammatical, on the basis of *a priori* syntactic considerations but we have found it possible to create many broad categories on semantic grounds. Consider the sentence types:

1) How many x are y?
2) Which x are y?
3) Is x y?
4) Give the x that is y!

We have created the grammatical category SUBST to cover all the grammatical constructions which can fill the y slot. The SUBST-rules are:

Rule	**Example**
SUBST → NP	Is 2 a factor of 4?
SUBST → ADJ	How many factors of 12 are even?
SUBST → ARITHRELS	Which factor of 12 is greater than 6?
SUBST → PREPP	Give the factors of 12 that are between 1 and 6!

This group contains a surprisingly diverse range of traditional syntactic categories. However viewed on semantic grounds their function in the example sentences is identical. Each of the types of SUBST's is semantically represented by a constructive set. Thus we have the set of factors of 4, the set of even numbers, the set of numbers greater than 6, and the set of numbers between 1 and 6. Some of these sets such as the set of even numbers are infinite sets. In constructive set theory

infinite sets are represented by a characteristic function rather than by a list of members, as we shall describe in Section 3 below.

Considering that the x in the sentence types will also be a set, it is easy to see what the rules and their associated semantic functions should be for the four sentence types given. C stands for the cardinality of a set, I for the intersection of two sets, and S for the subset relation on two sets.

```
1)  Q → \HOWMANY\ NP LINK SUBST          (C (I ;2; ;4;))
2)  Q → INTERROGATIVE NP LINK SUBST      (I ;2; ;4;)
3)  Q → LINK NP SUBST                    (S ;2; ;3;)
4)  C → \GIVE\ NP                        (;2;)
    NP → NP RELPRONS                     (I ;1; ;2;)
    RELPRONS → RELPRON LINK SUBST        (;3;)
```

The use of a greater variety of lexical categories does have the affect of lengthening the grammar but it also has advantages. Like the DEACON system [3], we use a separate lexical category for each function word. The reasons for this are obvious in a system which aims at extracting all the semantic information possible during the parse. However it is not only the functional words which have been carefully categorized but also the substantive words. For example, nouns are currently divided into the two categories N and FCN. The FCN's are nouns which name functions, e.g., factor, sum, and intersection. The N's are nouns which name sets (constructive sets) e.g., fraction, number, and prime.

Our handling of the classification of substantive words, which is very similar to that of Sager [16], is based on the premise that categories can be formed which contain words that are both naturally related to each other with regard to the subject matter and naturally related with regard to their grammatical role. In the case of our nouns, this is obvious. The division into FCN and N clearly has semantic relevance. It also clearly has grammatical relevance because FCN's as function names will always be accompanied by their argument and this argument slot must be taken account of in the grammar. Thus the division into categories adds syntactic precision to the grammar as well as increasing its power as a tool for extracting semantic information.

When the parse is complete, the semantic construction which represents the semantic surface structure is passed to the evaluation program. Before discussing the evaluator, a comment should be made on the ambiguity problem. One motivation for performing syntactic and semantic analysis interactively is so that the semantic routines can be used to resolve syntactic ambiguities as they arise. Our system however also provides a natural means of disambiguation. Most of the lexical ambiguity which arises from words having multiple lexical categories is resolved by the

grammar since generally only one of the forms will parse. Grammatical ambiguity which may arise from unresolved lexical ambiguity or from ambiguous parsings of a single lexical form is often resolved by default in those cases where all the grammatical parses result in the same semantic construction. Semantic ambiguity which arises from ambiguous grammatical parses with different semantic constructions is resolved by the evaluation phase if only one of the semantic constructions can be successfully evaluated. Thus the natural progression of the system eliminates spurious ambiguities. The remaining ambiguities are those which result when more than one alternative semantic construction can be successfully evaluated or when the evaluation of a single semantic construction reveals an ambiguity because of multiple senses of an item in the data base. These ambiguities will have to be resolved by the evaluator. The evaluator is the natural place for this disambiguation to be performed since it has access to the data base or world model including information about the context of the conversation and it can be provided with a wide range of heuristic disambiguation procedures.

2.3 Evaluation

The evaluator is a LISP program which evaluates the semantic constructions using a recursive inside-out technique. The evaluation is essentially independent of the natural language processing components. This has several advantages. First it can be programmed in a different language. Second the fact that the natural language processing routines are not dependent on the form of the data base means that a) the entire system does not need to be reprogrammed in order to take advantage of new ways of representing knowledge in the data base (which is a rapidly advancing field), and b) different types of knowledge can be stored in different types of data structures. Also the various semantic functions can be programmed using different data base manipulation techniques and the code for these functions can easily be changed or extended.

3 Constructive Semantics

By saying that CONSTRUCT embodies a *theory of meaning*, we mean that there is a set-theoretical characterization of the "world" that the CONSTRUCT fragment is referring to, and that there is a map between the sentences of the fragment and objects in this set-theoretical world. For example, consider the sentence:

How many factors of 4 are there that are also multiples of 2?

The sentence is a question, and the correct answer is the cardinality of a set, i.e.,

$$| \{x | x \text{ is a factor of } 4\} \cap \{x | x \text{ is a multiple of } 2\} |$$

The set-theoretical structure that would be required to deal with this sentence would have to have a FACTOR and MULTIPLE function over the domain of natural numbers, as well as the logical and set manipulation functions.

This kind of semantics was developed to deal with the formal languages of mathematical logic. Tarski in [19] showed how to give composition rules for the syntax of first-order logic and how to characterize the meaning for the formulae built up from those composition rules.

The power and importance of the method soon became obvious to logicians. In examining the models that a particular formal language had, it became clear what the expressive power of that language was. The metaphors of intended interpretation were stripped away, leaving for each theory an exact appraisal of what the theory really was. Thus, conflicting accounts of such historically difficult problems as entailment, tense, and modality became analyzable in a systematic way.

3.1 Objections to Model-theoretical Semantics

There have been serious doubts raised about the relevance of logic in general to computational linguistics. Most of the substantive objections are only about the technique of representing information using predicate calculus and then using resolution theorem proving to provide the inference machinery. Minsky [8] raises a somewhat more general objection that stresses the holistic character of "common sense" reasoning; he argues that one cannot "confront . . . [a logical] system with a realistically large set of propositions." (4) Nevertheless, Minsky still considers only the proof-theoretic aspects of mathematical logic, and does not really deal with the semantic aspects that we are suggesting are relevant to computational linguistics.

Objections specifically about model-theoretic semantics include the following:

> 1) "While formally adequate, set-theoretical semantics is not the way to look at natural language for computers." We have no ready answer to this objection except to note that it applies equally well to all other theories.
>
> 2) "Set-theoretical semantics commits one to a referential view of language -- that words refer to things." But the "objects" in the set structure could be any theoretical or abstract construction, just as the "atoms" in a LISP program need

(4) P. 73 of [8].

not be material objects. Set-theoretical semantics is independent of metaphysical and epistemological commitments.

3) "The notion of 'defining the meaning' in set-theoretical semantics presupposes our ability to perform impossible computations." This belief is based on a confusion about what it means to "define" or "evaluate" the meaning of a sentence. The sense of "evaluation" used by logicians means to give a characterization of the truth conditions of a sentence. The "evaluation" notion current in computer science is that of carrying out a computation that leads to an answer. Knuth [6] has called these two notions *existence* and *construction*. It is important to note that we can know the truth conditions (i.e., the "meaning") without being able to carry out the computations those conditions imply. Patrick Suppes' example is the question:

Are there natural numbers x,y,z bigger than 0, n bigger than 2, such that $x^n + y^n = z^n$?

It is perfectly reasonable to say that we know what this question "means" -- it is a statement of Fermat's last theorem -- but that we cannot answer it yet.

3.2 Computer Implementation

This last objection is quite serious in that any computer implementation of a set-theoretical semantics must contend with infinite set structures in a computationally graceful way. For example the question

How many even numbers are prime?

involves three infinite set in its meaning conditions:

$$| \{x | x \text{ is even}\} \cap \{x | x \text{ is a number}\} \cap \{x | x \text{ is prime}\} |$$

The theoretical solution is to base the semantics on *constructive set theory*, in which potentially infinite (or very large) sets are represented by characteristic functions. These functions are then available to be called directly, combined with other functions, or examined for information extraction. The computational problems are then no different than in any similar system, except that one has an exact characterization of the conditions of success. (5)

---

(5) See Rawson [15] for a description of the feature extraction and heuristic techniques that were used in the evaluator to compute the answers to questions like the above.

12 A Semantic Theory for Computational Linguistics

We propose *constructive semantics* as the basis for computational linguistics. Of course this semantics is particularly suited to the fragment that we chose (elementary mathematics). But the importance of constructive semantics extends to other systems with a less exact subject matter, and there are benefits to be obtained from the application. For example, we are rather impressed with the notions of *inference* of Roger Schank and his associates [17]. We think that their ideas would improve by having their conceptual dependency system formalized with a modal logic semantics.

4 Semantic Transformations

A common theme in computational linguistics is to show how a given system obtains "transformational" powers. Woods in [23] showed how his augmented state transition networks extend beyond the power of context-free grammars to cover syntactic transformations. Winograd [22] argued that his procedures obtain similar capabilities by simply being programmed to do so. Petrick [13] used an explicitly transformational system.

Beginning with Chomsky [1] generative grammarians postulated a syntactic deep structure with transformations mapping to the surface structure. One motivation for the appeal to the transformation was that the unaided context-free grammar was clearly insufficient to account for all the syntactic features observed in natural language.

This point was made quite forcefully by Postal in [14]. Using the embedding properties of context-free languages, Postal demonstrated that certain constructions in Mohawk could not be recognized using any context-free grammar. The importance of his argument rests on the unquestioned assumption that the grammar should account for all "syntactic" features (such as inflection) and provide a "structural description" that accounts for "all the grammatical information about a sentence which is in principle available to the native speaker". (6)

There is no reason why the role given to a phrase-structure grammar should be the role that Postal assumes. Moreover, the notion of "syntax" that Postal is looking for may be incoherent. One of the major theoretical insights of Winograd [22] is that the division into "syntax", "semantics", and "pragmatics" may not represent the way that our knowledge is used in

---

(6) [14], p. 137.

understanding natural language. (7) One can give plausible interpretations to such Chomskian [1] non-sense sentences and non-sentences as "Colorless green ideas sleep furiously." and "Furiously sleep ideas green colorless." (8)

4.1 Control Structure and Semantic Transformations

The CONSTRUCT system uses a context-free grammar to lay out the surface control structure of the semantic functions. Some of these functions (such as PLUS) perform a simple computation. Other functions, however, cause an interrupt that changes the control structure. These semantic functions are *transformational* in that their combined effect is to set up an alternate control structure that calls only functions that compute values in the straightforward way.

Consider the question:

(1) *Is 2 or 3 odd?*

The CONSTRUCT grammar parses (1) with a tree structurally similar to the one in Figure 2:

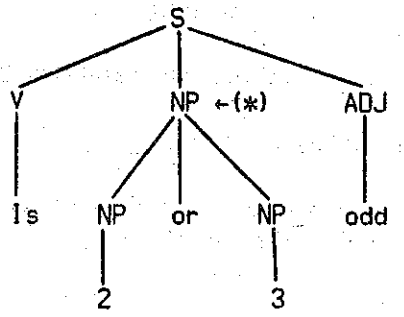


Figure 2. Surface Syntax Structure of *Is 2 or 3 odd?*

---

(7) An example that will illustrate this point is "The city councilmen denied the women the permit because they were revolutionaries", where determination of the antecedent of *they* depends on knowledge about the political character of city councils.

(8) See for example Minsky's frames paper [8], pp. 25-26.

Note that the noun phrase "2 or 3" is parsed at a low level in the grammar, while the semantic effect of the word "or" has its impact at a higher level where the phrase "odd" is available. Thus, the semantic function associated with the node (*) causes the alternative control structure shown in Figure 3 to be set up.

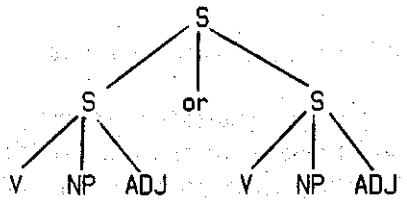


Figure 3. Semantic Deep Structure of *Is 2 or 3 odd?*

The structure in Figure 3 is the *semantic deep structure* for (1). The function associated with (*) causes a *semantic transformation.*

Winograd's solution is to interrupt the parsing process itself with "demons" to handle special words such as "or". The major disadvantage of Winograd's method is that the description of the flow of control becomes very difficult, and the tendency develops to rely on the code itself as the best (if not the only) description of what is happening. (9)

Another example of a semantic transformation is sentence (2).

(2) Does 12 have 12 as a factor and as a multiple?

Figure 4 gives the surface structure, and Figure 5 gives the semantic deep structure obtained after a transformational semantic function alters the control structure.

---

(9) This reliance on the code can of course be carried to the extreme of suggesting that the program itself is a theory of meaning.

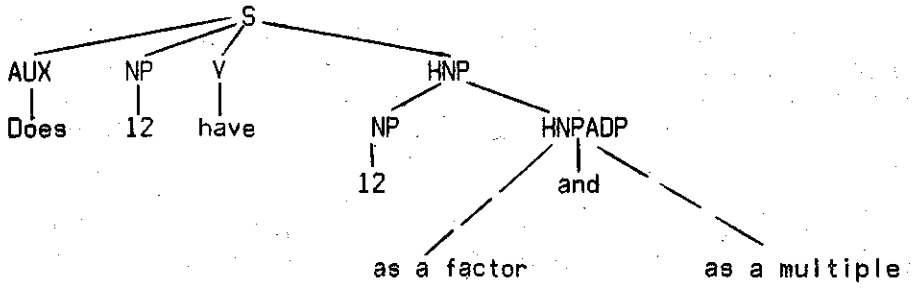


Figure 4. Surface Syntax Structure of *Does 12 have 12 as a factor and as a multiple?*

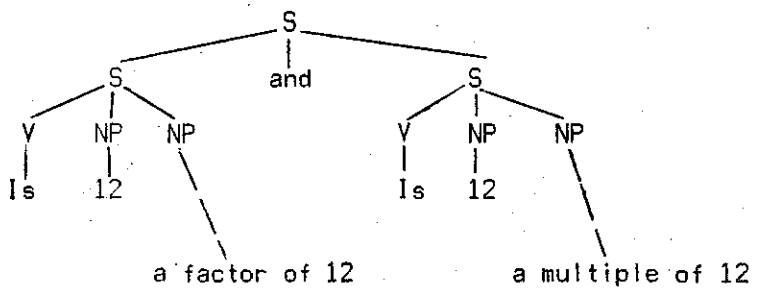


Figure 5. Semantic Deep Structure of *Does 12 have 12 as a factor and as a multiple?*

Semantic transformations are similarly useful in handling quantifiers. In a sentence such as

(4) Does 12 have any factors that are even?

there is a reasonable case to be made that the "syntactic scope" of the quantifier *any* extends only to the noun phrase in which it occurs. The natural "logical scope" of *any* would however contain more of the sentence, extending over the predicate phrase. In the CONSTRUCT system, the parse provided by the context-free grammar associates the quantifier with the noun phrase; then, the evaluation of the semantic functions associated with these rules sets up the control-structure where the scope of the quantifier corresponds to its logical scope.

4.2 Context-free Grammars Revisited

As we suggested previously, a good deal of effort has gone into exposing the limitations of context-free grammars. It is our view that syntax analysis should only be expected to lay out the control-structure of the semantic functions, where some of the functions may be transformational (when they correspond to "natural" transformations). This necessitates a rethinking of the limitations of any syntax-analysis method.

For example, one limitation of a context-free grammar is that there cannot be arbitrarily many constituents at a given level. This typically happens for adjective phrases, as shown in Figure 7. The best that a context-free grammar can do is to use some recursion generating a tree as in Figure 8, where the constituents become embedded.

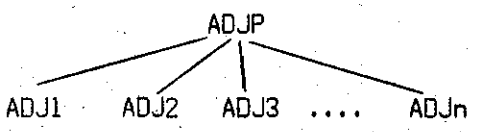


Figure 6. Immediate Constituents.

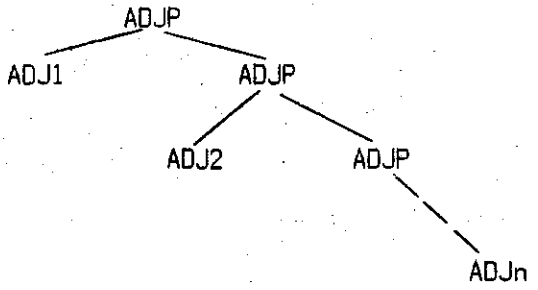


Figure 7. Context-free "approximation" of Immediate Constituents.

The real difficulty with Figure 7 is that it makes the semantic analysis unwieldy. In the examples in the CONSTRUCT fragment, the typical situation is that each ADJi is a set structure, and it is necessary to have all the sets available at the same level *of the semantic analysis* in order

to evaluate them correctly. In CONSTRUCT, a tranformational semantic function provides Figure 6 as the semantic deep structure.

By similar means it is possible to show that none of the limitations ascribed to context-free grammars apply when one has semantic functions associated with the rules.

5 Alternate Approaches to Semantic Transformations

Several approaches that are quite close in spirit to the semantic transformations of the CONSTRUCT system deserve to be mentioned. Two such approachs are the interrupts in the parsing process of Winograd's blocks world [22], and the augmented transition networks of Woods' system [23]. Two other approaches that are closer to the semantic transformation notion of CONSTRUCT are those of Montague and Knuth.

5.1 Montague and English As A Formal Language

Richard Montague [9],[10], (and philosophers such as Moravcsik and Gabbay [11] who have proceeded from where Montague left off) are motivated primarily by the analogy that English is a formal language, i.e., similar to the languages of first-order logic. Montague and his co-workers also noted the above problem of the scope of quantifiers. Their solution has been to postulate large set structures to serve as denotations for quantified phrases such as *every man* and *some woman*. Then, the meaning of a sentence such as "every man loves some woman" can be defined in a way that appears to avoid any transformational mechanism. The Montague approach is quite elegant logically. It fails to satisfy linguists in that their need is for something like the insight into psychological reality and structure of the brain that they believe tranformations provide. (10) For computational linguistics, the Montague method is computationally awkward since it escalates the complexity of the type structure beyond what can be reasonably represented in a machine. One way to characterize this escalation of type structure is to say that information is being encoded at a low level (in the denotation of the noun phrase), and then being decoded at a higher level (in the whole sentence). We feel that the theoretical approach of the CONSTRUCT system is more computationally feasible. In addition it fits into the linguistic tradition of transformational grammar, and points the way to eliminating the notion of syntactic deep structure. Fillmore [4] has presaged the elimination of syntactic deep structure, calling it "a level the properties of which have

(10) See Partee [12], p. 243 for a discussion of the relation between Montague grammar and traditional transformational grammar.

more to do with the methodological commitments of grammarians than with the nature of human languages".

5.2 Knuth and the Semantics of Context-free Languages

Donald Knuth [6] suggested a system of attributes for the nodes of a context-free parse tree. The calculation of the values for these attributes proceeds in such a way that information is passed both up and down the parse tree. The effects that one obtains are formally equivalent to our semantic transformations and to the Montague set structure encodings of information. The approach is particularly applicable to the implementation-independent definition of programming languages, called "declarative" by Knuth and his students.

Wilner [21] uses a multi-processing environment to calculate the values for the Knuth attributes of a parse tree for a compiler called SIMULA 67. He shows that one can use the order in which these values are calculated to determine an *augmented derivation tree.* (11) The augmented derivation tree is quite similar to the semantic deep structures that the CONSTRUCT system obtains. One difference is that the augmented derivation tree structure arises implicitly from the order in which the Knuth attributes are assigned values; thus the augmented derivation structures are somewhat arbitrarily determined by the attributes selected. In the CONSTRUCT system, the explicit notion is that we are computing an alternative control structure that arises from the initial syntax analysis.

6 Schematology and Transformational Semantics

Although we have described the theory of language that is embodied in CONSTRUCT in terms of model theory and formal semantics, the program itself is procedural in nature. For example, the evaluation component of the system performs dynamic manipulation of functions at runtime by creating and destroying specialized functions as it evaluates the semantic parses passed to it. Moreover, the semantic transformations are functions mapping semantic parses to semantic parses, and as such, can be described only in procedural terms. At present, most representations of procedural knowledge are in terms of programs, and therefore are not very accessible to the kind of model theoretic analysis presented in the preceding sections. A method of analysis of procedural knowledge called *schematology* has been developed that has promise of giving a mathematical

(11) Pp. 172-184 of [21].

account of the notion of semantic transformation. (12)

6.1 The Motivation for and the Development of Schematology

Schematology is designed to extract from a program the control structure of that program. It isolates those parts of the program such as conditionals, branches, returns, and subroutine calls that affect the sequence of machine instructions executed. This is done by replacing the actual logical conditions and arithmetic and logical functions of a program, such as an ALGOL program, by predicate and function symbols. This is strictly analogous to the replacement of the actual words of a sentence when representing the logical structure of the sentence using first-order logic. By studying such abstractions one is able to analyze many programs at once in much the same way that first order logic analyzes many sentences at once.

Much of the work on schematology has a motivation which is surprisingly similar to a problem that has plagued linguists ever since Chomsky introduced the notion of a transformation in the 1950's--that of finding "natural" constraints on transformations. Although it is intuitively obvious that programming languages vary greatly in their expressive power, it is possible in most languages to write for any given recursive function a program that will compute that function, and according to Church's thesis it is impossible to do any better than that. Hence, most programming languages are universal in the sense that one can write a program for an arbitrary recursive function in them. For example, both ALGOL and machine language are universal in this sense despite the fact that ALGOL is clearly a more powerful and more expressive language than machine code.

Given this universality, it is impossible to compare programming languages on the basis of which functions they can be used to compute. However, Hewitt [5] has been able to compare classes of schemata, and to show, for example, that in some well-defined senses, schemata that permit recursive function calls are more powerful than ones which forbid them. It is this type of comparative study that should be useful for computational linguistics.

The problem of finding natural constraints for transformations faces the same difficulties as attempts to compare directly the power of programming languages. Instead of a direct analysis, we considered classes of schemata for transformations, and tried to characterize what classes are required to give the computational power needed to analyze natural language by computer. It is important to have a clear idea of the computational power needed to analyze natural language.

---

(12) See [7] and [5].

6.2 The Analysis of Semantic Functions in Terms of Schemata

As an example of how semantic transformations can be analyzed in terms of program schemata, consider again the semantic parse of the question

Is 2 or 3 odd?

Figure 2 (in Section 4 above) presents a version of the surface syntax structure for this sentence, and Figure 3 gives the corresponding semantic deep structure.

The schematological representation for the computation needed to evaluate this is as follows. Let x1 be the list of choices, let x2 be the set of odd numbers, and the interpretations of the functions be as follows:

```
S1  checks for an empty list (NULL)
S2  appends two lists together (APPEND)
S3  is the subset function which accepts two arguments A and B
and returns A if A is a subset of B and returns NIL otherwise
S4  selects the first element of a list (CAR)
S5  selects the rest of a list (CDR).
```

The constant NIL is used to represent the empty list. The schema that represents the desired computation is:

```
(V x1 x2) = (repeat (z <- NIL)
 (if (S1 x1) then (return z))
 (z <- (S2 z (S3 (S4 x1) x2)))
      (x1 <- (S5 x1))).
```

This means that z is to be initialized to NIL, and the remaining statements are to be executed until the condition in the first is true, that is, the list x1 has become empty, and the (return z) statement is executed. The left arrow represents assignment. The second statement builds the answer list (z) while the third reduces the argument list by removing from it the choice just considered in the second statement.

6.3 Classes of Schemata

Following Hewitt [5], it is possible to define a number of different classes of schemata. At

present we have investigated only two such classes -- one that represents the non-transformational semantic functions of the CONSTRUCT system, and one that represents all of the semantic functions that are used in the system. Although we shall call the latter class the transformational schemata, it includes as a subclass the non-transformational schemata. Intuitively, a non-transformational schema is merely a sequence of assignment statements while a transformational schema is a non-recursive program schema in the sense of Hewitt [5]. As should be obvious, there is a transformational schema that is not equivalent under Hewitt's notion of equivalence to any non-transformational schema. The details of this are given in Rawson [15].

Linguistically, a non-transformational schema represents a program for computing the value of a semantic parse by simply doing an end-order traversal of the semantic parse tree. That is, one merely evaluates the nodes in order starting with the innermost expressions and working outward. This is essentially the algorithm used by the top-level of the LISP interpreter to evaluate S-expressions. On the other hand, a transformational schema, which permits looping, allows one to change the order in which the sub-expressions of a semantic parse are evaluated, and permits the repeated evaluation of certain portions of the parse. This seems to be the key computational concept in our notion of semantic transformations.

6.4 The Problem with Schematology

The major open problem with our analysis of the computational structures needed to do semantic analysis is that the class of transformational schemata is too powerful: there are too many control structures which fall into the class. We must find some class intermediate between the non-transformational and the transformational schemata that represents more accurately the control structures really needed for natural language understanding. All that we can really claim to have done is to have reformulated the classical problem of finding natural constraints on transformations in terms which make it more susceptible to solution.

7 Conclusion

The 700-rule grammar of the CONSTRUCT system recognizes a reasonable fragment of the sentences one finds in elementary mathematics texts. Our choice of fragment emphasized complexity of structure over a large vocabulary, and hence we obtained a good variety of ways of asking the same questions. (13)

(13) A modified version of CONSTRUCT is being used in a system for teaching set theory at Stanford beginning this fall.

The original contribution of CONSTRUCT is not the grammar itself or the use of set-theoretical structures as the semantic world for the meanings of utterances. The importance of the system is the notion of *semantic transformation.* The important aspects of this notion are:

1) The analysis of the surface syntax of an utterance is the control structure of a program that computes the meaning of the utterance.

2) The execution of this program to compute the meaning of the utterance may involve functions that change the flow of control, setting up an alternative control structure, which we call the *semantic deep structure.*

3) Techniques from the mathematical theory of computation, in particular schematology, may be useful in giving a formal characterization of these semantic transformations in a way that will both provide useful tools to the computational linguist and aid the general linguist as well.

References

1. Chomsky, Noam, *Syntactic Structures*, Mouton, The Hague, (1957).

2. Colby, Kenneth Mark, and Enea, Horace, Idiolectic language analysis for understanding doctor-patient dialogues, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, Calif., (1973), pp. 278-284.

3. Craig, J.A., Berezner, S.C., Carney, H.C., and Longyear, C.R., DEACON: Direct English Access and Control, *AFIPS Conference Proceedings*, 29, (1966), pp. 365-380.

4. Fillmore, Charles J., The case for case, *Universals in Linguistic Theory*, E. Bach and A. Harms (Eds.), Holt, Rinehart, and Winston, New York, (1968), pp. 1-88.

5. Hewitt, C., PLANNER, doctoral thesis, MIT AI, (1971).

6. Knuth, D. E., Semantics of context-free languages, *Mathematical Systems Theory* 2 (1967), pp. 127-145.

7. Luckham, D. C., Park, D. M. R., and Paterson, M. S., On formalized computer programs, *Journal of System and Computer Sciences*, (1970).

8. Minsky, Marvin, Frame systems, unpublished paper.

9. Montague, Richard, English as a formal language, *Linguaggl nella Societa e nella Tecnica* (Language in Society and the Technical World), Milan, (1970).

10. Montague, Richard, The proper treatment of quantification in ordinary English, *Approaches to Natural Language*, K. J. Hintikka, J. M. E. Moravcsik, and P. Suppes (Eds.), D. Reidel, Dordrecht, (1973), pp. 221-242.

11. Gabbay, Dov M., and Moravcsik, J. M. E., Branching quantifiers, English, and Montague-grammar, forthcoming paper.

12. Partee, Barbara, Comments on Montague's paper, *Approaches to Natural Language*, K. J. Hintikka, J. M. E. Moravcsik, and P. Suppes (Eds.), D. Reidel, Dordrecht, (1973), pp. 221-242.

13. Petrick, S., A recognition procedure for transformational grammars, Doctoral dissertation, MIT, (1965).

14. Postal, Paul M., Limitations of phrase structure grammars, *The Structure of Language*, J. A. Fodor and J. J. Katz (Eds.), Prentice-Hall, Englewood Cliffs, New Jersey, (1964), pp. 137-151.

15. Rawson, F. L., Set-theoretical Semantics for Elementary Mathematical Language, doctoral dissertation, Stanford University, (1973).

16. Sager, Naomi, Syntactic formatting of science information, *AFIPS Conference Proceedings*, 41, (1972), pp. 791-800.

17. Schank, R., Goldman, N., Rieger, C. J., and Riesback, C., MARGIE: memory, analysis, response generation, and inference on English, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, Calif., (1973), pp. 255-261.

18. Smith, N. W. A question-answering system for elementary mathematics, Technical Report No. 227, Institute for Mathematical Studies in the Social Sciences, Stanford, Ca., (1974).

19. Tarski, Alfred, "The Concept of Truth in Formalized Languages", in *Logic, Semantics, and Metamathematics*, Oxford, (1930).

20. Wilks, Y., *Grammar, Meaning and the Machine Analysis of Language*, London, 1972.

21. Wilner, W. T., Declarative semantic definition, STAN-CS-233-71, Computer Science Department, Stanford University.

22. Winograd, Terry, Procedures as a representation for data in a computer program for understanding natural language, Doctoral dissertation, Massachusetts Institute of Technology, 1971.

23. Woods, W.A., Transition network grammars for natural language analysis, *Communications of the Association for Computing Machinery*, Vol. 13 (1970), No. 10, pp. 591-606.