American Journal of Computational Linguistics *Microfiche 32 : 58*

SEMANTIC-BASED PARSING AND A NATURAL-LANGUAGE INTERFACE FOR INTERACTIVE DATA MANAGEMENT

JOHN F. BURGER, ANTONIO LEAL, AND ARIE SHOSHANI

System Development Corporation
Santa Monica, California 90406

ABSTRACT

We describe a natural-language recognition system having both applied and theoretical relevance. At the applications level, the program will give a natural communications interface facility to users of existing interactive data management systems. At the theoretical level, our work shows that the useful information in a natural-language expression (its "meaning") can be obtained by an algorithm that uses no formal description of syntax. The construction of the parsing tree is controlled primarily by semantics in the form of an abstraction of the "micro-world" of the DMS's functional capabilities and the organization and semantic relations of the data base content material. A prototype is currently implemented in LISP 1.5 on the IBM 370/145 computer at System Development Corporation.

INTRODUCTION

In a recent article in Scientific American, Dr. Alphonse Chapanis says, "If truly interactive computer systems are ever to be created, they will somehow have to cope with the...errors and violations of format that are the rule rather than the exception in normal human communication"[1]. An example

dialogue produced by two persons interacting with each other by teletype-writer to solve a problem assigned to them by experimenters showed that "not one grammatically correct sentence appears in the entire protocol."

Many existing language processors (Woods, Kellogg, Thompson, etc.) [2,3,4] are limited to what Chapanis calls "immaculate prose," that is, "the sentences that are fed into the computer are parsed in one way or another so that the meaning of the ensemble can be inferred from conventional rules of syntax," which are a formal description of the language. In effect, users are required to interact with these systems in some formal language, or at least in a language that has a formal representation in the computer system that a user's expression must conform to (we are thinking, in the latter instance, of Thompson's REL, which has an extensible formal representation facility). In addition, most natural-language question-answering systems, including all referenced above, require that a user's data be restructured and reorganized according to the particular data base requirements of the natural-language system to be used.

At the level of artificial intelligence research [5,6,7], there is some interest in systems that recognize meaning in natural-language expressions by methods that do not require compiler-like syntactic analysis of an expression prior to semantic interpretation. We believe it is possible, practical, and feasible, using new linguistic processing strategies, to design a natural-language <u>interface</u> system that will permit flexible, intuitive communication with information management systems and other computer programs already in existence. This interface is open-ended in that it has

no prejudice about the user's system functions and can be joined to almost any such system with relatively little effort. It is, in addition, able to infer the meaning of free-form English expressions, as they pertain to the host system, without requiring any formal description or representation of English.

THE SEMANTIC INTERFACE ALTERNATIVE

The syntactic inflexibility of existing natural-language processors limits their usefulness in interactive man-machine tasks. Our approach does not use a collection of syntax rules or equations as they are normally defined. Instead, we construct a dictionary in which we define words in terms of their possible meanings with respect to the particular data base and data management system (DMS) we want to use and according to the possible relations that can exist between data-base and DMS elements (e.g., an averaging function on a group of numbers) in the limited "micro-world" of this precisely organized data collection. Words appearing in a user's expression that are not explicitly defined are ignored by the system in processing the expression; an example would be the word "the," which is usually not meaningful in a data management environment. We thus avoid the expressive rigidity that formal syntactic methods impose on the user and the excessive time and resource consumption that results from the combinatorial explosions usually produced by such methods.

We distinguish in their definitions between two types of words: content words and function words (or "operators"). Content words are words whose

"meanings" are the objects, events, and concepts that make up the subjects being referred to by users. More precisely, for data management systems, these meanings (or "concepts") are the field names and entry identifiers for the data base and the names for available DMS operations such as averaging, summing, sorting, comparing, etc. Function words serve as connectors of content words. Their use in natural language is to indicate the manner in which neighboring content words are intended to relate to one another. In the example "the salary of the secretary," used below, "salary" and "secretary," are content words, and "of" is a function word used to connect them.

Many content words are context sensitive. In a particular data base, for instance, the word "salary" may refer to the data-base field name SECSAL if the salary is "of a secretary," but may also indicate the field name CLKSAL if it is a "salary of a clerk." In recognition of this we therefore define each content word by a set of one or more pairs of the form

((X1 Y1) (X2 Y2) ... (Xn Yn))

where the Xi and Yi are "concepts" (that is, field names, etc.) as described above. This expression may be interpreted as, "if the word so defined is contextually related in a sentence to X1, its particular meaning in this context is Y1, if it is so related to X2, it means Y2, and so forth." This particular contextual meaning of the word is called its <u>sense</u>. Two content words are considered to be semantically related if the intersection of the Xi's from the definition of one word with the Yi's from the definition of the other is not empty.

To get a more intuitive understanding of this process, suppose, again, that a data base contains entries for both secretaries and clerks with salaries for each. Suppose "Suzie" is an instance of a secretary and "Tom" is an instance of a clerk. We then have three words defined as follows:

Suzie	((SUZIE SECY))
Tom	((TOM CLK))
Salary	((SECY SECSAL) (CLK CLKSAL))

Processing the phrase "Suzie's salary" would intersect the Yi ("(SECY)") from the definition of "Suzie" with the Xi's ("SECY" and "CLK") from the definition of "salary." The intersection is non-empty ("(SECY)"), and, in discovering the semantic relationship, the sense "SECSAL" is assigned to the word "salary." Similarly, "Tom's salary" assigns the sense "CLKSAL" to "salary."

A particular implementation of the natural-language interface processor operates for a particular DMS/data-base target system. It contains a particular dictionary created for that target system. For a particular dictionary, the set of all lists of pairs as described above, therefore, constitutes the equivalent of a concept graph or network for the particular data base analogous to those used by many of the more conventional parsers for semantic analysis following (or during) the syntactic phase of parsing.

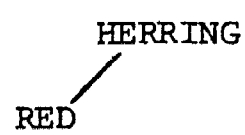
In the analysis of a particular input by our system, two words in context are tested using the "intersection" method described above and, if they are found to be semantically related, they are considered candidates for "connection" as described below. Two words so connected form a phrase.

Function words are defined as operators or processors that perform this semantic test. The definition of one function word differs from that of another according to its slope (see below) and also in that the operational definition of a function word can reject a connection even though the two words may be semantically related. In the operational definition of the function word may be a list of acceptable concepts or a rejection list of unacceptable concepts. In most conceivable data bases, the phrase "salary in the secretary" would be thus rejected by the function word "in."

As the analysis of an input expression proceeds, a "clumping" of word and phrase meanings is more and more explicitly connected until, normally, the processing of the entire sentence results in a tree structure made up of the connected senses of all the content words from the sentence. This result we term the sentence graph even though the input expression may not be a grammatically complete sentence. This sentence graph will be translated into a formal DMS statement.

We recognize that the linear ordering of the words in an input expression is not entirely random and that certain aspects of the function of syntax must be taken into account. This is done by means of a new and powerful algorithm based on what we call the syntactic-semantic slope. Linguists generally recognize that whenever two units of meaning are combined, one is semantically dominant and the other subordinate, as a modifier is subordinate to the modified word. After combination, the dominant word may be used in most cases to refer to the conjoined pair. Thus, a "red herring" is a "herring" (not a "red"), and the "salary of the secretary" is a

"salary." If this relationship of dominance is represented vertically on a rectangular graph (i.e., dominance on the Y-axis), and if the linear ordering of the words in the expression is represented on the X-axis in normal left-to-right order, then the connection of an adjacent pair of content words or phrases will describe a linear slope on the graph. The slope is positive or negative as the dominating sub-unit is, respectively, to the right or to the left of the subordinate sub-unit. For example, the phrase "red herring" makes a positive slope, thus:



and "the salary of the secretary" makes a negative slope:

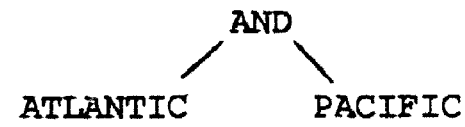
Thus, the operational meanings of function words operate on the meanings of nearby content words. Dominance is assigned, semantic relationships are verified, and the relationships so discovered are accepted or rejected. If accepted, the two word-meanings are connected, and the acceptable sense is assigned to the dominant word.

Function words may connect content words in "positive," "negative," or "peak" connections. The following are examples of each manner of connection:

1. "Of" is a negative operator, as in "the salary of the secretary":

2. "'s" is a positive operator, as in "the secretary's salary":

3. "And" is a peak operator, as in "Atlantic and Pacific." In contrast with positive and negative operators, peak operators add a representation of their own semantics into the structures they build:



4. Between any two adjacent content words there is an implicit "empty" operator that is a positive operator, as in "red herring":

In general, all prepositions are defined as negative operators. This is equivalent to the rule

NP → NP PREP NP

used by syntactic processors. The positive empty operator is equivalent to the rule

NP → ADJ NP

and others, while verbs and conjunctions are defined as peak operators, giving our statement of rules such as

S → NP VP NP

and

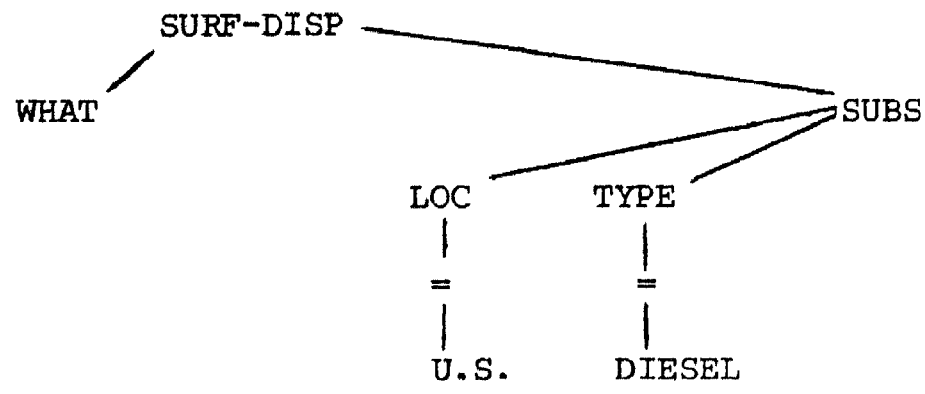
NP → NP CONJ NP.

Each operator has the facility to accept or reject any semantic relation according to the precise definition of the function word for the host data management system.

Progressive connection of word meanings and previously connected groups or "phrase meanings" results in a tree graph that we call the sentence graph. For example, the question "What is the surface displacement of U.S. diesel submarines?" could, for a particular data base, produce from the dictionary a string of content-word and function-word definitions that might be represented typographically like this:

((SUB SURF-DISC)) <OF> ((U.S. LOC)) ((DIESEL TYPE)) ((LOC SUBS) (TYPE SUBS))

As a result of processing, these will assemble into a tree structured (using the senses of the words) like this:



Even though this tree, or sentence graph, is created as a result of semantic relationships instead of formal rules of grammar, it still closely resembles the "parse tree" produced by most conventional syntactic language processors. With respect to the user's target data management system, the sentence graph is precise and unambiguous and contains enough information for a

straightforward translation into the formal query language of the DMS. In SDC's DS/3 language, for example, the above question would be expressed as

PRINT SURF-DISP WHERE TYPE EQ DIESEL AND LOC EQ U.S.

The response to the user's question will thus be the response from his DMS to the formal query statement.

The user's input in this hypothetical example is proper in form and grammar. However, it need not have been. The request

OBTAIN SURFACE DISP FOR US SUBS SUCH AS HAS TYPE EQ DIESEL.

would produce exactly the same sentence graph and therefore, exactly the same formal query statement with the same response from the DMS.

It is not likely that a syntax-based parser would have anticipated the odd language-use and grammar of this last request. Without a syntax rule that would allow for the phrase "such as has" such a parser would not look at the semantics involved and would be unable to interpret the request. Our syntax algorithm gets the same results that would be expected from the application of syntax rules without the need to anticipate each grammatical construct expected from the user.

In overview, the parsing algorithm makes a series of positive, negative, and peak connections based on the operational meanings of the function words (including the "empty" operator) and on the relations between meanings of the content words. The algorithm adheres to the following rules:

Rule 1: Connections between content words are possible only if the result of the intersection test described above is non-empty

and if this result is not rejected by the operation of the function word performing the test. The function word definition also determines which word supplies its X's and which its Y's for the test. It thus controls which word has its sense determined if the test is successful. Most often (though there are exceptions), positive operators use the X's from the word to the right and the Y's from the word to the left of the operator. Positive operators, therefore, determine the sense of the word to the right. This is illustrated using, again, the secretary and her salary. Consider the definition of "Suzie" and "salary" as shown on page 5. The phrase "Suzie's salary" has two content words, "Suzie" and "salary," separated by the function word, "'s." This function word is a positive operator and, hence, applies the intersection test to the Xi from the definition of "salary" with the Yi from the definition of "Suzie." These values are, respectively, "(SECY CLK)" and "(SECY)." The intersection yields "(SECY)," which is acceptable to the "'s" operator, and the connection is made with "salary" as the dominant word. The sense of "salary" is the Yi associated with "SECY" in the definition of "salary," hence, "SECSAL." This selection process is reversed for negative operators, while peak operators employ both kinds of tests, one on each side of the peak.

Rule 2: No node in a sentence graph may have more than one dominating node. That is to say, all connections must result in trees. This is a common assumption consistent with conventional syntax-driven parsers.

Rule 3: Given a subtree, a constituent on its left has the possibility of connection only to nodes of the subtree's positive adjacent slope, and a constituent on the right can connect only to the nodes in the adjacent negative slope. Intuitively, this means that if the nodes of a subtree are connected by "lines" that are "opaque barriers," then a constituent on either side of the subtree may connect to it only on those nodes that it can "see." It may not connect to nodes on the "inside" or the "far side" of the subtree. This is a powerful heuristic rule that eliminates the need to try connections to many syntactically impossible portions of the subtree. In effect this one rule, together with the definitions of the function words, replaces all the syntax rules used by most conventional parsers.

Rule 4: In order to minimize disconnection of existing subtree structures (backup) and still consider all possible connections, the system should, whenever possible, construct subtrees starting from the top and make new connections from below. This rule leads to the following algorithm: Scan the constituents from left to right making negative connections, then scan from right to left making positive connections. Scan thus back and forth until no more connections can be made. Then make any possible peak connections and repeat the algorithm. Continue this process until all constituents have been connected into a single tree.

We have observed that if ambiguities exist under these conditions, they will be semantic and, in all probability, not resolvable by any further processing

or analysis of the expression. Therefore, there is no need to carry along temporary multiple construction possibilities. The algorithm may either query the user at this point for disambiguation or abort the processing and inform the user of the reason.

REFERENCES

1. Chapanis, Alphonse. Interactive human communication. Scientific American, May, 1975.

2. Woods, W. A. Transition network grammars for natural language analysis. Communications of the ACM, October 13, 1970.

3. Kellogg, C. H., et al. The CONVERSE natural language data management system: current status and plans. ACM Symposium on Information Storage and Retrieval, University of Maryland, 1971.

4. Thompson, F. B.; Lockman, P. C.; Dostert, B.; Deverill, R. S. REL: a rapidly extensible language. Proceedings of 24th National Conference, ACM, New York, 1969, 399-417.

5. Riesbeck, C. K. Computational understanding. Theoretical Issues in Natural Language Processing: Proceedings of an Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics and Artificial Intelligence. Cambridge, Massachusetts, June 10-13, 1975.

6. Waltz, D. L. On understanding poetry. Theoretical Issues in Natural Language Processing, Proceedings of an Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics and Artificial Intelligence. Cambridge, Massachusetts, June 10-13, 1975.

7. Schank, Roger, and Tesler, L. G. A Conceptual Parser for Natural Language. Stanford Artificial Intelligence Project. Memo No. AI-76, January, 1969.