Advantages of a Transformational Grammar for Question Answering

Fred J. Damerau
IBM Corporation
Thomas J. Watson Research Center
Yorktown Heights, New York

A number of researchers in artificial intelligence, for example, Woods(1975, p.88 ff.), have asserted that transformational grammars are not a satisfactory basis on which to construct natural language understanding systems, primarily because of efficiency considerations. The evidence for such a claim is by no means strong, Petrick(1976), and it can be argued that transfer of new theoretical insights into a language understanding system based on transformational grammar is facilitated, Plath(1973). This note shows that a transformational parser can also simplify problems of relating canonical representations of queries to data base representations.

Consider a data base consisting of a set of company names each with an associated list of employees. A natural question for such a data base is M How many people does company Y employ?" Our grammar produces an underlying tree structure whose bracketted terminal string is something like (I), from which a Knuth-style semantic interpreter produces a LISP form like (2).

(1) (EMPLOY (company Y) ((how many) person XI)).
(2) (SIZEOF(SETX 'XI '(TESTFCT XI (EMPLOY Y 1977))))

TESTFCT would trigger extraction of names from the data base, SETX would create a set of these names, and SIZEOF would determine the cardinality of that set. So far, this is simple enough and no difficulty arises. The first query system we constructed had a small data base of business statistics of large corporations, Plath(1973), Petrick(1973). Consider in this context a question like "What were GE's 1970 earnings?". The underlying structure was something like (3), where the semantic interpreter produced a LISP form of roughly (4).

(3) (EQUAL (the X5 (GROSS GE X5 1970)) (some amount XI)).
(4) (SETX 'XI '(FORATLEAST 1 'X7 (SETX X5 (TESTFCT X5 (GROSS GE 1970)) (EQUAL X7 XI))))

FORATLEAST implements the default quantifier, and TESTFCT finds GE's gross income. This data base also contained the total, number of employees for each company. If we were to ask

(5) How many employees does GE have?

the system would produce an underlying structure related to (1), leading to a retrieval program like (2). Unfortunately, we need a retrieval program like (4), with "EMPLOYEE" substituted for "GROSS". We could of course modify the SIZEOF function to be sensitive to the data field it dominates and return the set rather than the cardinality of the set in appropriate cases, but this is aesthetically unattractive (although this is in fact what we did in our very first system). We could also modify our translation equations and semantic interpreter so as to be sensitive to this situation. While this might be satisfactory in one or two cases, the number of special cases can become very large.

In our present application, which is an English query system for the planning files of a small city near our laboratory, there are many more situations of this kind. One can ask

(6) In what zone/planning area/ census tract/ etc., is parcel 5 located?

For each of these questions, the underlying structure has a top level verb of "LOCATED", where the translator would prefer "ZONE" or "PLANNING AREA" etc. Again, one could make the LOCATED function sensitive to its arguments, or insert the appropriate equations into the translator, but the complexity of either solution is much greater than before.

Transformational grammars customarily have two sets of rules, *cyclic* rules, which apply successively to each level of embedding, and *postcyclic* rules, which apply globally to the entire sentence. Our grammar has an additional set of rules, called *string transformations,* Plath (1974), which apply to strings of lexical trees. The transformational parsing program calls each of these sets of rules separately. Since the parser is basically a tree processor, it can be applied, via an additional set of rules, to underlying structures like those for (5) and (6), and modify the structures in such a way that the semantic interpreter can produce correct code without data base specific modifications. In the case of (5), the output of the new processing phase, called the *precycle,* is a structure like (3) instead of a structure like (1), with a data identification of "EMPLOYEE" rather than "GROSS". At the cost of an additional call to the transformational parser, we have insulated both the semantic interpreter and the data base functions from the organization of the data base, confining the necessary modifications to a single table of rules. We have not yet found a class of structural changes we wished to make because of the data base which required more than one rule. Therefore, the cost of writing new rules has been much less than the cost of generating new programs for these special cases would have been.

While I am sure other system developers are able to solve this general problem, as they must in order to proceed in their work, we have nonetheless been pleased to note that our decision to use a transformational approach on linguistic grounds has had additional benefits on practical grounds.

References:

Petrick, Stanley R. 1973. Semantic Interpretation in the REQUEST System. IBM Research Report RC 4457, IBM Corp., Yorktown His., NY.

Petrick, Stanley R. 1976. On Natural Language Based Computer Systems. IBM Journal of Research and Development, vol. 20, No. 4, pp. 314-325. July, 1976.

Plath, Warren J. 1973. Transformational Grammar and Transformational Parsing in the REQUEST System. IBM Research Report RC 4396, IBM Corp., Yorktown Hts., NY.

Plath, Warren J. 1974. String Transformations in the REQUEST System. American Journal of Computational Linguistics, Microfiche 8, 1974.

Woods, William A. 1975. Comment on a paper by Petrick, in Directions in Artificial Intelligence, R. Grishman, ed., New York University, New York, 1975.