[*Proceedings of the Ninth International Congress of Linguists*, Cambridge, Mass., August 27-31, 1962 (The Hague: Mouton, 1964)]

THE IMPACT OF LANGUAGE DATA PROCESSING ON LINGUISTIC ANALYSIS

PAUL L. GARVIN

A number of linguists have in recent years become increasingly involved in language data processing activities. The purpose of this paper is to assess the effect of this new area of interest on the field of descriptive linguistics.[1]

From a linguistic standpoint, language data processing can reasonably be defined as the application of data processing equipment to natural language text. Of greatest interest is of course the application of computing machinery.

Language data processing can either serve linguistic ends, as in automatic linguistic analysis, or more practical ends, as in the fields of machine translation, information retrieval, automatic abstracting, and related activities. The latter can be summarized under the heading of linguistic information processing.

All areas of linguistic information processing are concerned with the treatment of the content, rather than merely the form, of documents composed in a natural language. This emphasis on content constitutes one of the major differences between this aspect of language data processing and the field of descriptive linguistics as presently constituted, where the main emphasis is on linguistic form (although more recently, descriptive linguists have become increasingly interested in problems of meaning.).

Within the field of linguistic information processing, a major division can, from a linguist's standpoint, be made between on the one hand machine translation and on the other hand information retrieval, automatic abstracting, and related activities. This division is based primarily on the manner in which the particular activity is concerned with the treatment of the content of the document. In machine translation, the major objective is one of recognizing the content of a document in order to render it in a different language. In the other activities, for which I have proposed the cover term content processing, recognition of the content is only the first step. More than simple rendition is required; the content of the document has to be processed further for some such purpose as the inclusion of its entirety or portions of it under index terms, or the retention of certain portions and rejection of others in order to create an extract or abstract. Content processing thus involves not only the recognition but

[1] Work on this paper was done under the sponsorship of the AF Office of Scientific Research of the Office of Aerospace Research, under Contract No. AF 49(638)-1128.

also the evaluation of content, since for both indexing and abstracting, pertinent relevance judgments have to be applied.

There are two areas of language data processing in which linguistic work has found serious application in fact as well as in theory. These are automatic linguistic analysis and machine translation. While automatic linguistic analysis is as yet still in the planning stage, some of the proposed approaches are set forth in sufficient detail to deserve discussion. Machine translation has progressed beyond the planning stage some time ago, though it is still far from being fully operational. In other areas of language data processing, linguistic contributions are as yet so ill-defined that their discussion from the viewpoint of this paper is premature.

In assessing the impact of language data processing on linguistics, the empirical principle as stated by Hjelmslev[2] provides three generally acceptable evaluation criteria by its stipulation of the requirements of consistency, exhaustiveness, and simplicity. It is the contention of this paper that language data processing has served to pinpoint the difficulties that are encountered in carrying out what everybody agrees on as a desirable goal: a maximally consistent, exhaustive, and simple linguistic description.

This is most evident in the case of consistency. In data processing, inconsistency is not merely undesirable, it carries a severe penalty: it is almost trivial to make the point that a computer program just will not run unless the set of instructions is consistent within itself. The linguistic information underlying the program must obviously be equally consistent. The avoidance of inconsistency therefore becomes an overriding operational objective. The means of meeting this objective is explicitness, since this is the mechanism by which inconsistencies are uncovered for correction.

Language data processing applications require the formulation of linguistic information with a degree of explicitness that is often not met in ordinary linguistic discourse. I should like to exemplify this from the area of automatic linguistic analysis.

One of the basic assumptions of my conception of automatic linguistic analysis is that linguistic techniques can be made computable. Let me here discuss the difference in explicitness between the verbal statement of a technique and its formulation for purposes of automation.

A technique which I have found extremely useful in the syntactic analysis of an exotic language is the dropping test, serving as an operational means of ascertaining the presence of a relation of occurrence dependence (one in which a unit A presupposes a unit B for its occurrence). At the Congress in Oslo, I defined this test as follows:

> The procedure is what I have called dropping: that is, in an utterance containing both A and B (or both A, B, and C), omit one of the units and inspect the resultant truncated utterance.
>
> For occurrence dependence, the dropping test will work as follows: A is dependent on

[2] See Louis Hjelmslev, *Prolegomena to a Theory of Language*, Francis Whitfield transl. (Baltimore, 1953), p. 6.

> B in an utterance containing both, if an otherwise identical utterance, but from which A is dropped, is also occurrent in the text, or is accepted by the informant as viable. B is dependent on A if the utterance from which A is dropped is non-occurrent in the text, or is not accepted by the informant as viable.[3]

For the purposes of the linguistic analyst, "utterance" can be accepted as a common-sense behavioral unit, it can be assumed that units A and B will have been previously specified by some linguistic method, and the statements "occurrent in the text" and "accepted by the informant as viable" seem to be adequate enough descriptions of the conditions for the positive or negative result of the dropping test.

For purposes of automatic linguistic analysis (even though at present only the processing of text but not the computer simulation of informant work can be envisioned) all of the above factors have to be specified in considerably more detail in order to formulate a computer subroutine simulating the dropping test:

> The dropping test can be simulated by an essentially cumbersome series of comparisons (which ... can be simplified, once certain conditions are met). For each comparison, the [computer] routines will have to identify a pair of unequally long strings of elements, such that the longer of the two strings contains all the elements of the shorter one, plus one additional element. The one element present in the longer string and absent in the shorter one can be said to be 'droppable' from the longer one, if both strings have been found to recur in the text sufficiently frequently to allow the assumption that the difference in their length is not due to chance. For every identified longer string, the droppability of each element will have to be tested by finding an appropriate shorter string. Those elements for which shorter strings are not found in the input text can then be assumed not to be droppable, provided enough recurrences have been found so the that absence of a particular shorter string is not attributable to chance. In order to allow for the necessary recurrence, [an extremely] large input text would ... be required. ...
>
> For a dropping routine to operate within the logically prescribed restrictions - that is, without an initial dictionary or grammar code - each trial would have to compare every string of n elements present in the input text to all appropriate strings of $n - 1$ elements. A series of passes could be envisioned, with the value of n increasing for each pass from a minimum of 2 for the first pass to the maximum found in the text, for the last pass. Assuming a punctuated text, this maximum value of n could be made very much smaller than the total number of elements in the entire text by requiring that no string be allowed to contain a period or other final punctuation mark - this would restrict the permissible length of a string to the span between two such punctuation marks, or between one such mark on one side, and the beginning or end of the entire text on the other. That is, the maximum permissible length of a string would be that of the longest sentence in the text.
>
> The reverse procedure, in which the program first ascertains the maximum value for n and then decreases it with each pass, is equally thinkable.
>
> Either procedure for applying the dropping test, to be carried to its logical conclusion, seems to require a program of quite unmanageable proportions.[4]

[3] Paul L. Garvin, "Syntactic Units and Operations", *Proc. VIII Internat. Congress of Linguists* (Oslo, 1957), p. 629.

[4] Paul L. Garvin, "Automatic Linguistic Analysis - A Heuristic Problem", *1961 Internat. Conf. on Machine Translation of Languages and Applied Language Analysis, National Physical Laboratory, Symposium No. 13* (London, Her Majesty's Stationery Office, 1962), p. 663.

Let me now compare the above description of the dropping subroutine for automatic linguistic analysis to the dropping test as practiced in linguistic field work.

The first observation that can be made is that the computer simulation of the test reduces itself to a series of comparisons of strings of unequal length. Next, it may be observed that the units to be tested for droppability are in both instances either observationally given or defined by prior procedure. In the case of automatic linguistic analysis, the units dealt with are simply printed English words which by virtue of being "observable" to the input mechanism become the proper units for processing by the program. The most significant difference between the test as used in field work and its automation lies, however, in the practice of the linguistic investigator of selecting particular utterances and particular units within these utterances as the objects upon which to perform the behavioral test. In the computer subroutine as described above, this is clearly not possible, and hence its execution would result in a computer run of quite unmanageable proportions. It is, in other words, necessary to explicitate not only the conditions of the test itself, but also to explicitate the set of conditions under which the test becomes capable of execution. In my plan for automatic linguistic analysis, I have attempted to do this by deferring the use of dropping routines in the analysis program to a stage in the process at which the conditions for its application have been created by the use of subroutines based on other linguistic techniques. In particular, I am proposing not to use dropping routines until after the program has led to the specification of certain linguistic classes and to use the dropping routines then to test for occurrence dependences of classes of words, which can be expected to be quite finite in number, rather than for occurrence dependences of individual words, the number of which can be expected to be unmanageably large.

The essential features of the dropping test, by virtue of which it is diagnostic of a dependence relation, thus remain unaltered when the test is automated. Some important attendant conditions, on the other hand, have to be adjusted to the constrains imposed by automation: firstly, form classes which in linguistic field work are implicit in the systematic similarities of questions to informents and informant responses, must be made explicit; secondly, field work allows the random access to the readymade store of the informant's memory, whereas in automatic linguistic analysis the necessary store of accessible forms has to be prepared by previous procedures.

Let me now turn to the questions of exhaustiveness and simplicity. I should like to discuss these on the basis of some illustrations taken from machine translation.

First, the matter of exhaustiveness. This is one of the most difficult requirements to define in linguistic analysis. I have commented on it in a previous context, and at that time I stated that a significant consideration is whether or not the analyst is dealing with classes of unrestricted or restricted membership.[5] In the first case, exhaustiveness can only be achieved by a listing of classes, since obviously a complete

[5] Paul L. Garvin, "On the Relative Tractability of Morphological Data", *Word*, 13 (1957), 22-3.

listing of an unlimited membership is self-contradictory. In the second case, a listing of not only all classes but also of all members is possible. This is, however, a purely theoretical definition of exhaustiveness. It does not touch upon the heart of the matter, which is the extent to which exhaustiveness can be achieved, or must be achieved, in a particular task of linguistic analysis and how the requirement can be met in practice. In machine translation, the program as was stated further above is to a significant extent based upon a linguistic description of the source language. Since the aim of machine translation research is to produce ultimately a program that will be capable of dealing with randomly selected text, the question of exhaustiveness is of extreme practical importance and has to be faced from the beginning. There are two areas in which the problem of exhaustiveness arises: first, the lexicon, where in machine translation the problem concerns the machine dictionary; second, the grammatical description of the language, where in machine translation the grammar code and the syntax routines of the translation algorithm are involved. The field has not yet progressed sufficiently to allow the inclusion of problems of semantic equivalence and multiple-meaning resolution in the present discussion.[6]

Operationally, the problem is that the research has to be conducted, and the system developed, in stages. This remains equally true if the point of view is adopted that a complete linguistic description must precede the development of a machine translation system; such a prior complete description must still be prepared gradually. In either case, research in stages means dealing with one linguistic problem area at a time, without violating the requirement of exhaustiveness. The planning of the research stages thus becomes the primary question.

The details of planning for exhaustiveness involve the following:

In machine translation research, it is not possible to use the convenient scholarly device of the "et cetera", or of suggesting, "If this technique is carried further, it will then allow the treatment of the rest of the data."[7] The computer will not accept this kind of instruction. It is therefore necessary to make other provisions for exhaustiveness in spite of the limited scope of the dictionary and syntax program that the realities permit at any one given stage before the final aim has been achieved.

In regard to the machine dictionary, the question of exhaustiveness as to the number of dictionary entries is not of great theoretical interest, since it differs very little from the problem of exhaustiveness faced by lexicography in general. The only problem faced by machine translation researchers is that of having efficient procedures for

[6] Don R. Swanson, "The Nature of Multiple Meaning", *Proceedings of the National Symposium on Machine Translation.* H. P. Edmundson, ed. (Englewood Cliffs, N. J., 1961), p. 386: "Now that the stage has been set in previous discussion by the picturing of polysemia as a 'monster' or a 'blank wall', let me say that there isn't a great deal more to be said about multiple meaning that isn't either obvious or else wrong. ..."

[7] Cf. George L. Trager and Henry Lee Smith, Jr., *Outline of English Structure* (= *Studies in Linguistics, Occasional Papers* No. 3) (Norman, Okla., 1951), p. 55: "A full presentation would [include a complete description of phonemics, morphophonemics, morphology, and syntax]... No such full grammar is attempted here. The purpose is to present enough material for discussion to illustrate the procedures and techniques involved."

dictionary updating, and this problem has been solved satisfactorily by most groups.

The question of making provisions for exhaustiveness in the actual translation algorithm is much more interesting. It must first be noted that there are essentially two technical aspects to every computer program: first, a table-lookup procedure, in which the program looks up information in a table for use in later processing; second, a logical-tree-type algorithm, in which the program goes through a series of yes-no decisions (often based on information looked up in a table), in order to arrive at an appropriate end result. Provisions for exhaustiveness here consist essentially in writing a program that allows room for later additions as more information becomes known and can be included. The "et cetera" is replaced by a more explicit device. In the tables of a program, provisions for the addition of further information are made by leaving sufficient blank fields, that is, spaces to be taken up by later instructions and information. An important consideration in the design of the program then becomes the size of these fields that are to be left blank for later use. In a logical-tree-type algorithm, provisions for later additions can be made by building into the algorithm end points to which further branches can be added by which to treat information that may later turn out to be of importance. Thus, the linguist's statement that "additional data can be handled by the same technique" is replaced by an open-ended exit in the program.

In the machine translation program with which our department is operating, this can be exemplified by the blank fields that are contained in our grammar code for the addition of grammatical information for which we have planned, but which we have not yet been able to pin down sufficiently to include in the in the program. The open-ended exits are exemplified by the provisions of our syntax program to print out under certain conditions "notices of syntactic difficulty" whenever the algorithm for coping with such a difficulty has not yet been written.

Now to consider the matter of simplicity. I have on a previous occasion pointed out the difficulty inherent in a criterion of simplicity.[8] At that time I raised the question of defining simplicity more clearly by stipulating whether or not it is to be gauged in terms of minimizing the inventory of units, or by minimizing the number of rules. It seems to me now that the question cannot be answered in the abstract. It is impossible to specify the simplicity of a sequence of procedural steps or of the logical structure of a description in an objective way. It is too much a matter of esthetics.

It is not unreasonable, on the other hand, to attempt to specify simplicity in terms of the attainment of a particular aim, such as efficiency in the use of equipment. I should like to exemplify this by a brief discussion of the grammar code used in our machine translation program.

The purpose of our grammar code is to provide all the grammatical information that is required for the efficient operation of the syntax routines. One important task

[8] Paul L. Garvin, review of *Prolegomena to a Theory of Language* by Louis Hjelmslev, in *Language*, 30(1954), 70.

of these routines is to carry out an agreement check, that is, to ascertain on the basis of the grammar codes that have been furnished to the program whether or not certain adjacent words are in grammatical agreement with each other, such as for instance a noun and preceding adjectives. For purposes of maximum efficiency of operation, we have devised a grammar code which takes up more space in the computer memory but allows the rapid completion of agreement checks by a computer operation similar to ordinary subtraction of one digit at a time, an operation called "masking". The drawback of this type of grammar code is that it takes up a considerable amount of memory space. From the standpoint of storage, a grammar code compressed into the minimum amount of space is obviously vastly preferable,

It turns out, therefore, that in this particular case the requirement of simplicity has to be formulated quite differently in terms of the different purposes to which a particular set of elements is put. For purposes of the operation of agreement checks, a grammar code spread out over more space but allowing rapid completion of the check, is the most efficient and consequently the simplest. For purposes of saving memory space, the maximally condensed code is most efficient. It is thus possible to formulate a requirement of simplicity - if one equates efficiency with simplicity - quite clearly in terms of a particular purpose.[9]

The above discussion may explain why I have come to regard language data processing a very important application of linguistics. It is a challenge to linguistics as a science. The challenge is not theoretical, but operational - it is directed at both the methods and the results of linguistics. The strong requirement of exhaustiveness forces the treatment of minor subpatterns of a language, and not merely of its major patterns.

By enforcing its requirements, the computer has become an analytical instrument for linguistics, where previously only recording instruments were available. This may have important theoretical implications.

Thompson Ramo Wooldridge Inc.
Los Angeles

[9] We use both solutions in our machine translation program. We store the grammar code in its condensed form, and we have a simple subroutine which transforms the condensed grammar code into the spread-out grammar code for use in agreement checking.