Some Lucubrations and Specifications for a Natural Language Analyzer

JOHN B. SMITH
Department of English and the Computation Center
Pennsylvania State University

The article attempts to classify the kinds of problems to which the computer has been applied in natural language analysis. The difficulties encountered with existing programming languages and systems are inferred and some suggestions are made for a system that would make analyses of this sort more practical. This system is then related to the concept of the associative processing computer with the contention that the proposed system would be quite close to what might be expected as a high order language for this kind of machine. The overall purpose of the article is to stimulate discussion of the computing needs of natural language analysts by making specific suggestions for future systems.

Within the past several years, there has been a marked increase in the use of the high-speed computing machine for the analysis of language.[1] Many graduate departments thoroughly entrenched within humanities divisions now accept a computer language as the second research skill formerly satisfied by a demonstrated reading knowledge of German or French. There are now at least three journals (*Computers and the Humanities*, *Computer Studies*, and *Hephaistos*) dedicated to publishing computer-assisted research in the humanities and related areas. But perhaps most encouraging of all is the developing tendency of large funding agencies to sponsor the training of humanists and their research activities as evidenced by the institute financed by the N.S.F. and the A.C.L.S. held this past summer at Kansas University.

Thus, while the growth in the use of computers for language and humanistic research has been large, it has not been so large as it might have been. All too apparent is the fact that computing machines were not designed for doing language analysis or many of the other tasks that this group might wish for them to perform, and there seems no clear intent on the part of the computer industry to alleviate these difficulties. For example, if one attempts to do language analysis at more than fifty percent of the universities in this country, he must choose between the lesser of two evils in selecting a programming language. If he uses FORTRAN, then he most likely will have to write himself or have written for him a number of assembler language string handling routines. At every turn, he must be conscious of how the machine is 'viewing' his data and how to 'trick' it into doing what he might want it to do. Such games, while satisfying when you 'win', soon grow tiresome to the point of sapping much of the intellectual effort supplied by the researcher. If he chooses a string-oriented language such as SNOBOL, programming is easier, but costs for executing programs skyrocket some three of four hundred percent over what a physicist might expect to pay for 'comparable' computing. If the user happens to be in a university environment which has IBM equipment, he might use PL/1 for his programming language. This choice will solve many of the problems associated with other languages, but not all. For example, he must still use structures and arrays for much of his work; he must index these lists with numbers, not with letters of the alphabet; he must devote considerable energies to managing the input and output of his data; and while he has greater flexibility in managing his data files, he is faced with the most complicated and meticulous file specification language in the industry. In brief, he is confronted at every turn with what is for a considerable time an overwhelming mass of detail.

What can be done about these inhospitable conditions that face the scholar interested in language analysis? We could complain to the representatives of the computer industry that they are not meeting our needs. To this we are likely to get one of two answers. They might say that the proportion of use by language analysts does not warrant hardware and software development. But this is a chicken and egg argument — if machines were more amenable for language analysis, more language analysts

[1] I shall use the term language to refer to so called "natural" languages — those that are or have been used by human beings for general communication — as opposed to formal or artificial languages such as those used to program computers. Where the latter is intended, the term formal language will be used.

would use them. The second reply the computer people often give and one that they love to make with an annoying note of piousness is that language analysts don't know what they want in the way of computers and computer languages. The unfortunate truth is that we have not been able to specify very precisely what our needs are and how they could be satisfied. It is this point that I wish to consider here. I shall look first at some of the tasks in language analysis that have been done on computers and from these try to infer some of the inherent problems. Secondly, I shall consider some of the characteristics that a system should have to meet these requirements; and, finally, I shall examine the implications that such a system has for hardware realization. I don't realistically expect everyone involved in language analysis to agree with these suggestions. Rather, I hope to evoke a lively yet serious consideration of our computer needs. If this can be done, then in the not too distant future we, as a group, should be able to tell the computer manufacturers what we want and how to give it to us.

Among the tasks that have been done or could be done using a computer are the following: concordance and word index construction, collation of texts, author identification, stylistic analysis, content analysis, machine translation, and a catch-all that might be termed linguistic analysis. To compile a word index implies first of all the encoding into machine-readable form the text — often quite massive — the breaking down of that text into individual words with their contexts, an alphabetical sort of these records, and finally a printed copy of these sorted records. The task is straightforward and seemingly simple, but it is evident from the work done in the early sixties that general concordance utilities almost invariably have to be modified for individual texts.

To collate several texts involves passing several files in sequential order and checking for a match of individual elements. The tricky part is the 'restart' process following a deviation of several words. This is quite difficult if done in a batch mode environment and often necessitates several passes of the data sets. Ideally, the task should be done in an inter-active environment with manual restart capabilities.

Author identification studies have most often been based upon the distributions of several word types — usually function words — over a text. But like the preceding tasks, such analysis implies the encoding of the text, the manipulation of it in terms of individual words or items, followed by exhaustive and sophisticated analysis of quantitative measures derived from the text.

The area of language analysis that has perhaps received the widest attention is that of stylistic analysis. Definitions of "style" have ranged from broadly defined semantic patterns that permeate a text down to the distribution of a single function word. But inherent in most definitions and analytic approaches is the notion of a distribution of certain elements or categories over a work often accompanied by some attempt to establish interrelations and dependencies among them. To perform any sort of sophisticated analysis of this kind involves the same problems noted in previous examples: encoding and storage of text followed by some process to extract measures that can be "fitted" to a stylistic model. While results in this field have been significant, too often the mechanical problems of text manipulation have over-shadowed the more important tasks of defining comprehensive models of style and evaluating them with regard to actual pieces of writing.

Another area that is only slightly removed from stylistic analysis in terms of techniques is that general area termed content analysis. The implications of research in this area are most immediate and most important in the fields of information retrieval, automatic indexing and classification of documents, as well as automatic abstracting. Because of the voluminous files that develop, most efforts in this area have approached the problem in terms of word associations defined over the entire text. To introduce even the slight refinement of searching for associations of synonomously equivalent terms demands complex, expensive thesaural programs; but work in this area is still hampered by the problems of file manipulation and voluminous output. Attempts to define the content of a document more closely by reducing the defined range of associativity are promising; however, it is evident, from the criticism of the work of Iker and Harway, that the problem cannot simply be "plugged" into a statistical utility. Before such debate can be resolved, we must know more about the normative patterns that exist within natural language texts.

Although interest diminished during the late 1960s, there is some indication that renewed consideration may be directed toward machine translation projects. Drawing heavily from work in stylistic and content analysis, such efforts may attempt to reduce full, idiomatic texts to a restricted subset of words and syntactic patterns but with the important addition of stylistic parameters. The text in restricted form could be translated into a similar version of the second language and then expanded into the full idiomatic expression implied by the stylistic parameters. However, before such a translator can be expected most of the technical problems of text processing present in the other examples must be solved as well as the development of a precise, parametric stylistic model similar to that mentioned above.

The last category that will be listed is that of general linguistic analysis. This group could include almost any type of natural language analysis ranging from a pattern analysis of graphemes to a study of the semantic structure of an author's canon. If he begins with the former, the researcher must be able to break a word into individual letters or groups of letters, but scanning of this sort represents a high relative cost for computation. If he is doing semantic analysis, he is faced with the problem of defining associative contingencies, categorizing, establishment of distributions, statistical analysis and inference. Too often the mechanical problems in the first two or three steps preclude the researcher's ever getting to the task of inferring comprehensive structures for the work under investigation.

If we look back at this list of some of the more general types of language applications for which the computer has been used, what can we infer as to general characteristics? First, the text must be encoded, usually through a key punch, into a machine readable form. If it is of any length, this can be an extensive, laborious, and expensive undertaking. Secondly, it must be read into the machine and broken down into some sort of storable form. On many machines, the input expense is inordinately high in comparison with other tasks. If the data set consists of a number of sub-parts, the management of them can become extremely messy. Also, the function of scanning a string of input text is not one that the machine was designed to do particularly efficiently and costs for this service can also become exorbitant. Perhaps the most serious problem involves the relocation of an item in a text. Since the language analyst's data sets are often voluminous, he must either pass them sequentially a number of times — and hence accumulate a large bill — or use large random access files. Both alternatives imply laborious file management problems. Fourthly, internal manipulation involves primarily sorting, table look-up tasks, character-by-character comparisons of strings, and the accumulation of various counts. Once these counts are derived, however, the analyst may call in various statistical procedures that involve a degree of 'number crunching'. Therefore, both efficient string scanning facilities as well as the complete algebraic and logical manipulative capabilities of the large scientific machines are needed. If the analyst's purpose is to produce output for photo-reproduction, such as a concordance, his machine must be able to produce high quality print-out. On the other hand, if his purpose is analytic, he may produce many pages of data — if he is running in a batch mode — so that he can later leaf through it to find the particular information he desires.

If existing systems make such tasks difficult and/or expensive, what sort of system would be better? First, to absorb the high relative costs of input, the system should be designed to carry both batch and inter-active services simultaneously. By doing this the user could have his text prepared in the batch mode and then perform much of his actual analysis interactively. If he knows that he can easily and quickly obtain a desired piece of information, then he may no longer require the voluminous printed output now produced in much language analysis work. Thus, the application and system languages should be designed to work efficiently in both modes. Secondly, the researcher, especially the humanist, should be freed from as many of the details of input and output and file manipulation as possible — these tasks should be handled by the system after initial file creation. Thirdly, the language should be in free format form with as much of the jargon and gobbledy-gook removed as possible. Fourthly, since utilities almost invariably have to be rewritten for specific applications, the language should embody a number of functions that perform the fundamental tasks now done by utilities, but in a germinal form so that the user may tailor his program to do his specific job. Obviously these functions and the hardware should be designed in conjunction. Finally, a number of statistical procedures, graphic and other display facilities, as well as an extensive library of computer accessible dictionaries, thesauri, grammars, etc., should be immediately available and easily employed through the application language.

This list of desirable attributes for a language analysis system could obviously be longer, but from it we can see some of the things that such a system should do that are not done easily or not done at all by existing languages and systems. Below, I shall give a sketch of some of the characteristics of such a language and consider some of its implications both for the hardware and the system as a whole. The description is obviously not complete, but emphasizes those aspects that are most immediately relevant for the language analyst. The functions described might be thought of as being embedded in a general purpose language such as PL/1 with its algebraic, logical, macro, and subroutine capabilities.

The language should contain the usual facilities for variables, matrices, and multi-dimensional arrays. In addition, there should be facilities for defining structures. The user should be responsible for specifying the logical relations among the elements of the structure, but he should not be responsible for specifying the size of the structure. That is, he should not have to specify that there will be 500 repetitions of the particular logical arrangement of variables. At most he should be asked to supply a rough estimate of the number of elements for each variable or structure. All allocation of physical resources should be maintained by the system; more on this point will be said later. If the data in the structure is stored on some external device after its creation, the system should also store in that data set a description of the structure and all specifications necessary to access that data set in its logical arrangement. Thus, the user could call for a structure by merely giving its name: the system would, in turn, take care of all internal allocations and input specifications.

Often in language analysis it is useful to maintain a text in the form of several lists or structures. These might include a dictionary of word types that is associated with another list of index values or indicators of text locations of individual tokens of this particular word type. A second function, GROUP, could be used to indicate that several such structures are to be associated with one another. Subsequent operations, such as a search, might then be applied to this collection of structures to locate data that can be accessed easily only from the combined use of these individual components. Thus, a number of different structures might be associated with a single text. The system would maintain a catalogue of these group designators and would write with them on the data sets where they are stored descriptions of any logical relations that might exist among them.

Since so many tasks associated with language processing involve table look-up and pattern searches, extremely powerful and efficient functions to perform these operations must be built into the system at the most fundamental level. These tasks could be accomplished through combinations of several functions, but the most basic should be a SEARCH function. It will take as its arguments a text, a set of texts, or perhaps a group of lists or structures and, secondarily, a set of values to be sought in the designated texts or lists. As a default condition, the system should search whatever major text or structure is currently being processed. The value parameter, on the other hand, may be literals, variables, or logical configurations of both.

Working closely with the SEARCH function should be another function called RETURN. It will be used to indicate what values from the structures or texts are to be returned once the SEARCH function finds the appropriate locations. Stated another way, the SEARCH function can be performed on a data configuration used as a key, and the RETURN function will return in list form data contained in the structure configuration for those entires having the specified key.

Another function that could be used in conjunction with the SEARCH function is a CHANGE function. It takes as its arguments a list of locations, a structure designation, and a value. The function will change the structural elements at the specified locations according to the value specified. Again the value could be a literal or a masked literal resulting in direct modifications of those characters of the variables indicated; or it could be a logical or algebraic function that modifies the variables accordingly.

Another important aspect of language analysis is that of category designation. Since a number of non-disjoint categories may be imposed over a text or list, it is desirable for the designation of categories to be flexible and not result in any literal or actual rearrangement of the textual materials. For example, category designation should not be done by flagging, sorting, and rearrangement of the text or list. Instead, categorization should be viewed as a set of logically designated independent partitionings of the set of elements of a text or list. This designation may take the form of an algebraic or logical function, or it may be lists of elements or a logical vector denoting membership or exclusion. To realize this, a CATEGORY function could be used with a parameter indicating the text over which the category exists and a set of parameters indicating categorical groupings of items in that text. As indicated above, the item specification could be a logical or algebraic expression or even a call to a subroutine that performs a number of such operations. Thus category specification can be by literal inclusion, variable inclusion, and procedural or algorithmic inclusion.

Interest in the statistical analysis of natural language texts has increased dramatically in the past few years. Often the most difficult aspect of these investigations is not the statistical procedure itself — there are catalogued procedures for most models that are likely to be used — but is the preparation of the data for input into one of these procedures. To facilitate this step of the process there should be a COUNT function and a DISTRIBUTE function. Given a set of texts and a value, COUNT will return the number of times that value occurs. Of course, value could be a literal, a variable, a category, or an algebraic or logical expression. DIST, given a set of texts, a value similar to that provided for COUNT, and a unit interval, would return the distribution of that value over the text using the appropriate unit for a grid. The output of these functions could be stored, used as the set of arguments for a statistical function, or fed to a more complex statistical utility. To facilitate this last process, there should be an extensive library of statistical procedures that can be called directly from the user-language program.

The final phase of software design that I wish to discuss is that concerning input and output. It is hoped that after initial text processing, most users will approach the system through remote, visual display terminals of some sort on an inter-active basis. One immediate benefit would be a reduction in the voluminous printed output often produced in language analysis as it is done with existing systems. For example, a user with access to a system of the sort that I have been describing might never wish to produce a printed concordance. Instead, he would store a program that would supply him with the context information he needs, when he needs it, and for only those words he is interested in. Of course, he should have the facility to get a hard or printed copy of any display information that interests him, but the system I have described would make a vast amount of this output unnecessary. Obviously, then, the system should have simple yet flexible and sophisticated display and graphic capabilities. However, through a verb such as DISPLAY, the user should be able to get a line graph of a distribution or a listing of a concordance output. Hopefully, the more detailed display procedures could be accessed through an analog device such as a light pen, so that the user could modify the display directly and in a roughly continuous fashion.

Many of the other input and output functions associated with data sets should be transparent to the user. For example, once the logical structure of a text is specified, the user should be able to output the file with only the word STORE and possibly a designation of some physical device such as his own disk pack or tape. The system should then output the data in the format specified by the logical declaration of the structure along with the description of that structure and all information necessary to read the data set. The user, in turn, would not be concerned with input at all. He would access the file through the function he wishes to perform. For example, given a SEARCH request, the system would locate the appropriate structure and read it in if it is not in the accessible memory of the system at the time of the request.

The input of text for the first time, however, is a different matter entirely. Although natural language analysis has increased dramatically and will undoubtedly continue to do so, there must be some alternative to keypunching large volumes of text. The obvious solution is the optical character reader. Since the system I am describing is at least five years away from realization, there may be a realistic hope that developments in that field will make input via this device practical. If so, then I believe we can expect a second Gutenburg revolution in the availability of materials for scholarly research. Another hopeful sign is the increasing tendency of publishers to distribute the computer tapes used in the automated process of type-setting. We may soon see the day when new publications can bypass the transliteration problem altogether. Thus, while we can expect continued advances in language analysis on computers, the automation of the laborious and costly step of initial text preparation may well revolutionize the field.

However, the heart of the system I have been discussing lies not in the input phase but in the automated file handling capabilities and the SEARCH function. The first aspect, file handling, is well within the capabilities of present systems. All sophisticated monitors maintain extensive catalogues. The only problems inherent in realizing the file system described are in the input/output specifications and the bulk of material that would have to be added to the system catalogue. I/0 specifications could easily be generated from details of the logical structure specification and, hence, present no fundamental difficulty. The second problem, of the added bulk of material, essentially reduces to the SEARCH problem; for we could merely shift the file maintenance portion of the catalogue to a lower cost device and then search it on request. This solution becomes unfeasible only in proportion to search time required to locate a particular data set. Thus, if we can solve the search problem, we can solve the file problem.

Conventional search techniques have become quite sophisticated. Many are very efficient for data sets with certain, rigidly specified organizational schemes. For example, a binary scan of an ordered data set of N items can locate any item in the list in at most log2(N) +1 tries: thus, only twenty seeks are necessary to locate any item in a list of one million items. However, frequent modification of the file is costly and impractical. Hashing techniques are promising but are efficient only when considerable space in the file is left unused and only with a fast, uniformly distributed hashing function. These techniques are largely untested on large natural language data sets. Random access files are unquestionably the widest used system for quick location of a data item, but the user pays heavily in system overhead costs during processing. Thus, the kind of system that I have described could be realized with conventional hardware and conventional techniques, but the cost is likely to be enormous; and with multiple users, the system could grind to a halt because of overhead alone. In short, this kind of system seems highly impractical for existing computers.

There is a kind of computer, however, that can handle the search problem and, indeed, most of the other problems we have seen associated with natural language analysis; but this design has been implemented only in limited or pilot applications. It is also the subject of considerable controversy at the present time. The kind of machine that I am referring to is one with what is called an associative memory and one that is capable of associative processing. I shan't attempt a complete description of this concept of computer architecture — an excellent introduction to the topic and a considered summary of the factors pertaining to its implementation is M. H. Cannell, *et al.*, "Concept and Application of Computerized Associative Processing", distributed by the MITRE Corporation — but I will at least define the terms and discuss briefly the relevance of this approach to language analysis. An associative memory differs from the conventional, sequentially-accessed memory in that its entire contents can be searched simultaneously for a given item of data. This mode of operation is exactly opposite that used in the serially addressed machines of today. Existing machines search for a data item by taking an address specification and returning whatever is stored at that address. The associative memory machine takes the value or data item to be located, simultaneously scans the associative memory for that item, and returns the locations at which that item occurs. This is done by placing the desired content in a register and then "driving" the memory for the particular logical configuration of bits contained in the content register. Where that content resides in the memory will be indicated by a pulse on the sense wires for that location. With a bit more hardware, the concept can be extended to allow a simultaneous search for a logical configuration of several data items; and a third level of the associative concept includes the simultaneous processing or manipulation of data items in the associative memory itself.

The tremendous importance of such a computer for language analysis should be obvious. The SEARCH function seen to be the heart of the system described above is exactly realized by the search capability of the associative memory. If we deal with data structures, the extended capability of the associative store would allow search for logical configurations of data items which in turn would allow effective implementation of a function that groups or categorizes the data set by partitioning The returned locations of such a category search could be translated quite easily into a linear distribution of that category. In short, the system that was inferred from the problems that appear inherent in natural language analysis would seem to be quite similar to what a high order language for an associative memory machine might be. Such close and obvious congruence of problem language and hardware design is quite encouraging for fast, efficient, and flexible language analysis applications.

The major questions that remain concern the feasibility and likelihood of developing such machines. Cannell, in the paper cited above, argues that the feasibility of developing such a system has been demonstrated: the operating systems in several current machines use small associative memories and the Air Force has an experimental machine at its Rome base with a 2,000 bite associative memory. Also, the most probable mode of implementation of associative capabilities involves large scale integrated circuits. From one standpoint, the use of chips with each memory component consisting of several flip-flops, and gates, etc., is a logical extension of the IBM 370 series and the CDC 7600. Both of these achieve their high performance characteristics through a high speed monolithic buffer. To change that buffer to an associative memory would seem not only feasible but the logical direction for hardware design to go. This development would truly mark a fourth generation of computer architecture.

In closing, let me remark that I don't feel that the language and system I have discussed is the complete answer. My purpose has been to stimulate interest and a consideration of what our computer needs are. Strong refutation as well as approval of my arguments would indicate that I have succeeded. The important point, however, is that the time has come for those of us involved in language analysis to specify what is wrong with existing systems and what future systems should look like. Let us at least meet once and for all the challenge of the computer industry that we are unable to state our needs.