DOCUMENT RESUME

ED 111 336 IR 002 334

AUTHOR Porch, Ann
TITLE Preliminary Design for a Language Analysis Package (L. A. P.).
INSTITUTION Southwest Regional Laboratory for Educational Research and Development, Los Alamitos, Calif.
REPORT NO SWRL-TN-5-71-74
PUB DATE Aug 71
NOTE 16p.

EDRS PRICE MF-$0.76 HC-$1.58 Plus Postage
DESCRIPTORS *Computer Programs; Content Analysis; *Data Processing; *Design; Electronic Data Processing; Indexes (Locaters); *Information Retrieval; Item Analysis; Language Research; Permuted Indexes; *Specifications; Word Frequency
IDENTIFIERS Computer Software Specifications; *Language Analysis Package

ABSTRACT

A series of computer programs to handle natural language retrieval and analysis in a manner analogous to that of a statistical package is discussed. The document presents an overview of several language analysis projects currently underway, and of several research approaches to problems in language analysis. The manner in which the Language Analysis Package (LAP) could be used with each approach is considered, and design specifications for a state-of-the-art system are presented. A preliminary implementation of such a system using programs currently available at the Southwest Regional Laboratory is suggested. An overview of the design considerations and algorithms for LAP is also included. (Author/DGC)

A-1

IR

SOUTHWEST REGIONAL LABORATORY TECHNICAL NOTE

DATE: August 18, 1971

NO: TN 5-71-74

TITLE: PRELIMINARY DESIGN FOR A LANGUAGE ANALYSIS PACKAGE (L.A.P.)

AUTHOR: Ann Porch

ABSTRACT

A package of computer programs to handle natural language retrieval and analysis in a manner analogous to that of a statistical package is discussed. The document presents an overview of several language analysis projects currently underway, and of several research approaches to resolving problems in language analysis. Emphasis is presented on the way the L.A.P. could be used with each approach. Design specifications for a package sensitive to the state-of-the-art are documented. A preliminary implementation, using programs in SWRL's possession, is suggested as a first step in the development process. An overview of the design considerations and algorithms for the preliminary package is presented.

PRELIMINARY DESIGN FOR A LANGUAGE ANALYSIS PACKAGE (L.A.P.)

Introduction

For years there nave been "packaged programs" in statistical areas. These programs offer generalized computational capabilities in a form and format especially suited to easy use by researchers whose basic orientation is not that of Computer Science.[1] Researchers in the social sciences, for instance, are able to perform complex multivariate regression analysis by computer without undergoing any special training in programming or computer operations.

A language analysis package (L.A.P.) with a power comparable to that of statistical packages would have considerable general utility.

As machines become more powerful and their use expands to include more disciplines, it is increasingly significant for researchers in many fields to be able to use the speed and versatility of the computer for processing natural language text as well as numerical data. For example, researchers doing studies of textbooks now suffer from the fact that their work usually must be done by hand. In those instances where computer technology is employed, a special purpose program is generally written by the resident programmer, who may or may not have specialized training in techniques of natural language processing. The results are costly, both in time and money spent on processing with inefficient or one-shot programs. Because such programs are limited in scope and written for a special purpose, the researcher has little flexibility available to him, and finds that a relatively minor change in his research perspective may make the computer program unusable.

---

[1] See Biomedical Computer Programs, W.J. Dixon (ed.), Univ. of Calif. Press, (1970) 600 pp. and Statistical Package for the Social Sciences N. Nie et. al. (ed). McGraw Hill.

In the past ten years, a great deal of work has been done throughout the country and the world in natural language processing in fields such as artifical intelligence, information retrieval, machine translation, computational linguistics, and computer stylistics. Hundreds of computer programs have been written, debugged, run, and then shelved when the researcher went on to another project. A number of these programs are the product of months of careful work by experts. Some researchers, such as Borden and Watts at Pennsylvania State University, are trying to develop generalized systems which handle many of the basic tasks associated with natural language processing. The design proposed here suggests the use of the best of those existing programs whose authors are willing to release them.

Such an approach has several advantages: the package will reflect the power of the finest specialized programming skill presently available. The development costs will be minimized since the major programming task will consist of interfacing the existing programs or subsections in a modular fashion under the direction of one control routine, rather than developing each of the specialized routines from scratch. By carefully constructing the package in a highly independent, modular manner, individual routines may be easily "un-plugged" and replaced should a more efficient or powerful routine be developed. Thus, the system will be dynamic and open-ended, constantly updating itself to keep pace with the state-of-the-art.

One obvious disadvantage to such an approach is the degree to which it creates machine dependency. Since programs are written in

different computer languages at different computer installations using different machines, some account must be taken of the problem which will arise in "minor" modifications necessary to make them run at the particular installation chosen for the L.A.P. development. Some care must be taken to try to keep the entire package as machine independent as possible. Certainly it can only be implemented on a computer which had the ability to compile and run most of the major computer languages currently in use for language processing. One such installation exists at UCLA, where the IBM 360 mod 91 has compilers for the following languages: PL/1, FORTRAN IV (G & H), COBOL, SNOBOL, LISP, APL, 360 ASSEMBLER, ALGOL.

Certain hardware and system software requirements will also be essential to the efficient development of the L.A.P. Among these are an efficient system sort-merge routine, multiple tape drives, relatively large amounts of direct-access storage and considerable available core. Again, the UCLA installation is one example of a computer center which is amply equipped in all areas.

Processing Features

A survey of the work currently being done in the field of language analysis[2] reveals seven major areas of present interest and usage which can logically be included in the L.A.P. Of the seventy-five projects listed in the November, 1970 issue of Computers and the Humanities, nine dealt with frequency counts, seventeen with KWIC production, fourteen with semantic or content analysis, eight with statistics, thirteen with index production, six with retrieval systems, six with sentence

[2] See Porch, Ann, "People and Projects in Natural Language Processing: A Preliminary Bibliographic Directory;" TM 5-71-10, August 5, 1971, 107 pp.

parsing and six with miscellaneous items such as machine translation. Several projects must be considered to fall into more than one category. Ratios in Linguistics in Documentation (Current Abstracts), Language and Automation, and Computer Studies in the Humanities and Verbal Behavior are much the same, although with a slightly heavier emphasis on retrieval and parsing.

An ideal language analysis package, then, should have the ability to perform efficiently in each of these seven basic areas, and the flexibility to operate in any manner desired, either sequentially or concurrently. In addition, the L.A.P. should be modifiable at any time, either for more effective general use or for a particular research application.

Flexibility is perhaps the most important single consideration in the development of such a package. The L.A.P. will be useful only to the degree that it is adaptable to a variety of projects and approaches. The fewer constraints on the user, the more likely it is to be used. To save researcher time and money, the L.A.P. should provide the user with as many options as possible, allowing him to select precisely and easily only the functions he requires. No section of the package should be "called in" unless the researcher specifically requests it. He should not be limited to simply an exclusive "OR" type selection, where he can only chose to do either a KWIC or an Index, but should be able to combine any or all routines and subroutines in any fashion he desires. He may, for example, want to produce KWIC's on words occurring within his inclusion list while simultaneously producing an index of all words except those in his exclusion list and a frequency count of every word

in the text. He should be able to use only the retrieval aspect of the L.A.P. or only the statistical portion without being penalized by the fact he is using a package rather than a single program designed specifically for his purpose.

Data Base Manipulation

Since a number of researchers may be making use of the L.A.P., the system should have the ability to differentiate among data bases, selecting the base or sub-base from a large library of resident data bases stored on magnetic tape or disc and making it available to the researcher for his processing. For example, the entire library might include the complete works of Shakespeare, the ERIC files, and all California State approved first grade text books. Researchers using the L.A.P. should be able to easily obtain an input file containing only first grade reading books or only ERIC documents dealing with reading.

Often, data bases which a particular researcher may wish to use have been prepared elsewhere with each having different input conventions and formats. For example, one data base might be prepared with a logical record of 100 and in EBCDIC code, utilizing both upper and lower case characters, while another might have a logical record length of 72, in ASCII code, and be in upper case only with capitals indicated by a "/" preceding the capitalized letter. The L.A.P. should be able to handle virtually any input format and set of conventions that the researcher can specify.

A researcher may want to use output from one step in the L.A.P. procedure as input to another. He may want to select subsets of a given data base, process each separately, then cross reference the results or subsets of the results. He may want to do transformations on the data as it is being processed, and use the transformed data as input. L.A.P. should be able to save output for further processing and should be able to save subsets of data once they are selected, in order to save the expense of repeated retrieval processing. The researcher should be able to present the system with a new file or retrieve and use a file either he or another researcher has previously used or created.

Modes of Operation

In addition to the flexibility of modularity, input formats and file handling, the L.A.P. should take advantage of the best features of two basic kinds of operation. An interactive, conversational system can provide the user high flexibility with little training. He interacts with the computer by answering questions, provides the program information about options he intends to implement for a particular run. On the other hand, a non-interactive, "batch" processing environment is significantly less expensive. For example, one Los Angeles service bureau[3] prices interactive time at $360 per CPU hour, while batch processing is only $150 per CPU hour. Language analysis processing requires considerable CPU time, since most computers are not designed for text scanning and string manipulation, but rather for numeric processing. If the L.A.P. could provide interaction to collect the parametric

[3] C & C Computing, 8939 S. Sepulveda, Los Angeles, California

information required for the run from the user, and batch processing for the remainder of the run on the data base, it would be optimal in both areas. It can do so by having an interactive module which sets up the parameters for the batch run, which can then be scheduled to run at a time optimal for cost considerations.

Certainly, the need for such a package exists. The design proposed here is not in any way meant to be a complete solution to that need, but rather a tool with enough inherent flexibility to survive after scratching the surface.

L.A.P. Design

An overview of the L.A.P. design is shown in the macro-flowchart (See Figure 1). The design consists of function modules for each of seven basic types of language analysis: frequencies, KWIC's, retrieval, indexing, parsing, semantic analysis, and statistical analysis such as type-token percentages, cross tabulations, etc. Once a pilot model is implemented, work can begin on refinement and optimization, and further programs of interest may be added. One such additional program is FAMULUS, a bibliography generating program, in FORTRAN IV.

Sub-modules such as a routine scanning for words, a dictionary look-up, or a routine for sorting will be included within the larger modules. Each of these function modules will consist of a full-blown program which will have subroutines or internal modules within it, any one of which, like the main modules, may be "unplugged" and replaced by a more efficient routine. There will be an additional processing module to translate internal format conventions into those of the input data.

The control of the program flow will reside in the control module, the parameters of which will be set during the interaction with the user provided by the interaction module. Any or all of the nine main processing modules can be called into action by the control routine, and the order and structure of the calling procedure need bear no resemblance to the linear thinking of the user who interacted with the interaction module, but can be structured in terms of machine efficiency for the particular combination of calls required by the particular run.

For the purposes of a pilot study of the feasibility of the L.A.P., programs in SWRL's possession can be used. At the present time, the program library includes programs for the following modules: frequencies, index, KWIC, parse, semantic analysis, retrieval, dictionary look-up. Several of the programs were collected from natural language specialists throughout the country, several were written by the author. Coincidentally, all are in PL/1. In order to implement the L.A.P. design discussed here, four routines need to be written, and sufficient interfacing prepared to allow the existing programs to run in the control environment.

Following is a detailed description of the system design for the modules which need to be written, together with comments on techniques of interfacing them with several of the existing programs.

Interaction Module

The function of the interaction module is to act as an interface between the user and the package. It may run as a front-end portion of the total program, if the computer system utilized has interactive

capability, or it may run at a separate facility (such as on a SWRL minicomputer) and prepare user control parameters to be appended to the input data which will then be run in batch mode at the main facility.

It will ask the user questions about the parameters of the run he is initiating and will utilize his answers to prepare computer compatible control parameters to be read by the control module.

Since it is a separately functioning entity, it will serve as a training program for researchers using the package for the first time. As each control parameter is compiled through the question and answer process, a statement will be printed out for the user's information. If the interaction module is running as a front-end portion of the total program, the control statements will be passed directly to the control module. If it is running separately, the control statements will be output in a form appropriate for use as input to the main package program, such as punched cards or magnetic tape.

After the researcher has used the interaction module for a while he may find that he is sufficiently familiar with the requirements of the user control language so that he feels competent to prepare his own control statements without computer assistance. Certainly, such user expertise is one of the goals of the interaction module. As a teaching, as well as a functional program, it will keep a running count of user success and failure in the question answering process, and output such information to a system programmer whenever it is polled.

The system programmer can use such information to modify and upgrade the package for maximum success within the environment of the actual researchers making use of the system.

Control Module

The function of the control module is to read a set of control statements containing run parameters preceding the data for a particular computer run, and to perform a decision making function for that run.

In any particular case, the user will specify which package functions he wishes to use, as well as the particular output specifications he has. For example, he may wish to produce a KWIC, a rank-ordered frequency count and a parsing of his text. He will indicate his needs by means of a user control language which will be punched and appended to the beginning of his input data. The control module will read the user control statements and compile a decision table which can be used by the program during execution to determine which subroutines will be called for that run. If the user has made syntax errors in his preparation of the user control statements, the control module will return error messages to him which will enable him to correct his errors before re-submitting the job. The user control language will be designed in such a way that typical errors, such as the omission of a comma, will be automatically corrected, allowing execution to proceed. In such cases of automatic error correction, a message will be printed on the output indicating the assumptions made by the control module, enabling the user to check for possible misunderstanding of his intent. If the program is run in interactive mode, these assumption statements will be printed out before the program continues to further execution, and a verification of correctness will be required from the user.

For each run, the user will specify the form of his input data, such as record length, columns punched, location of variables (for statistics module) and size of data base being used. The control module will scan this information and set parameters for the run to optimize usage of computer equipment and peripherals. Such parameters will control selection of I-O subroutines, storage and access subroutines, etc. Messages will accompany the output, indicating the options used in a particular run. Provision will be made for user override of the defaults.

The user will also indicate special conventions used in his data, such as a slash preceding a letter to indicate upper case, or an "@PP" preceding a character stream to indicate the beginning of a new paragraph. The control module will evaluate the form of the input data, decide if sufficient information is present within the data to allow the requested function modules to perform, and determine whether the translate module needs to be called to provide the interface of data and programs. Messages will be output to the user indicating missing information which prevents execution. As in other cases, translation parameters for the particular run will accompany output. If the translation module is needed, the control module will pass the information contained in the user control statements concerning input conventions to the translation module. During execution, the control module will perform subroutine calls according to the decision table established from the user control statements.

Translation Module

The function of the translation module is to provide the interface between the user's data and the data conventions required by the particular

subprograms he wishes to use. It may be viewed as a subfunction of the control module.

Since it is extremely costly to convert a large data base to another format, the translation module will work in the opposite direction, converting the relatively few program conventions into the format in which the data exists. Such a conversion will be accomplished in the following manner.

Each of the function modules will have an array associated with it which is accessible to the translation module. The arrays will each have a dimension of 256, corresponding to the 256 possible 8-bit codes. Each position in the array will hold an octal number equivalent to the new value (the data dependent value) which should be utilized by the particular function program for the current run. The translation module will set up the arrays for those function modules being called by the current run, using the old value (the function module dependent value) as a subscript to locate the appropriate position within the array into which to store the data dependent value. For example, if the KWIC program is written to expect a slash ("/") preceding each character, which is to be taken as upper case, and the user's data has been prepared with a dollar sign serving the same function, an octal 133 (equivalent to an EBCIDIC "$") would be placed in position 97 of the array associated with the KWIC program, since 97 is the EBCIDIC decimal equivalent of a slash ("/").[4]

---

[4] EBCIDIC codes have been used, since the UCLA computer facility is a likely one on which to set up the package. The same algorithm could be used with a computer which uses ASCII, with an octal 040 being placed in position 47 of the array.

Each of the function modules will have a specially prepared initialization subroutine which initializes each of the program-dependent variables (such as a variable "capital") to the value contained in the appropriate position in the array associated with that module.

The utilization of the general purpose array will allow great flexibility in those cases where a new module is to be added or substituted to the system, since no modification will be necessary to the total system in order to handle a completely different set of input requirements.

Output Module

The function of the output module is to direct various portions of the output to appropriate devices at the user's discretion.

In general, the default situation will be that all output goes to the printer; however, user's may have particular needs or preferences, and may elect to use another output device. For example, the user may wish to save his output in some computer compatible form for later input to some other program. In such a case, he may specify output to magnetic tape, punched cards, or punched paper tape.

Another, although complex use of the output module, would be an instance where the user wants his output to serve as input for another module within the package itself. In such a case, the output module would not only put the output onto a selected device, but also would put the data into an appropriate storage location within the system.

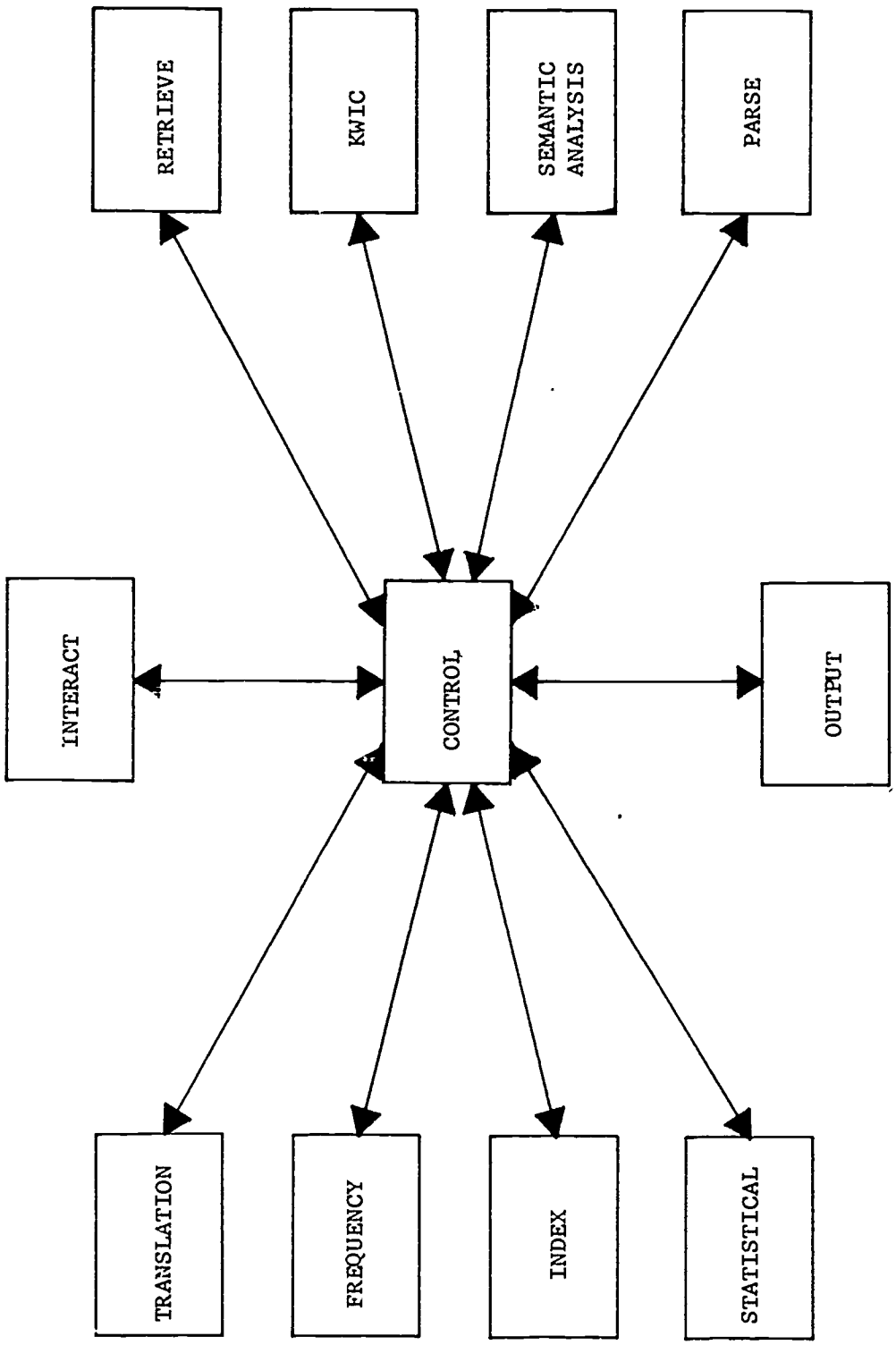
RETRIEVE
KWIC
SEMANTIC ANALYSIS
PARSE
INTERACT
CONTROL
OUTPUT
TRANSLATION
FREQUENCY
INDEX
STATISTICAL

Figure 1