46

The research reported herein was partially supported by a grant from the National Science Foundation (GP-7069). Use of the University of Wisconsin Computing Center was made possible through support, in part, from the National Science Foundation, other United States Government agencies and the Wisconsin Alumni Research Foundation (WARF) through the University of Wisconsin Research Committee.

A MEMORY NET STRUCTURE: PRESENT IMPLEMENTATION AND A PROPOSED LANGUAGE

by

Stuart C. Shapiro

Technical Report #53

December 1968

TABLE OF CONTENTS

A MEMORY NET STRUCTURE: PRESENT IMPLEMENTATION AND A PROPOSED LANGUAGE

Abstract

A memory net structure has been designed which is particularly useful for storing, processing, and retrieving information of the type used in semantic analysis of natural language, question answering and theorem proving. The net consists of blocks, or items, each one of which may be an unstructured unit or may represent a structure consisting of an ordered pair or triple of items. Any given structure (or unstructured unit) in the memory is represented by exactly one item no matter how many other structures make use of it as a substructure. Any item may be associated with an external print name. The structures are built using symmetric links so that all structures which use a given item are reachable from that item.

Procedures to manipulate this net and some to use it in a question answering system have been programmed in Burroughs B5500 Extended ALGOL, also making use of ASLIP, a SLIP like package of list processing procedures.

A net manipulating programming language is proposed whose basic data units are the items and structures of this net, just as the basic data units of a list processing language are elements and lists.

INTRODUCTION

The memory net structure herein described (MENS) developed as a generalization of the SAMENLAQ structure [1]. After almost all of the work discussed here had been done, it was discovered that these structures are very similar to the structures used in and proposed for the DEACON project. The SEMENLAQ structure is similar to the ring structure described in DEACON [2], and the MENS structure is similar to the triangle structure described by Longyear [3]. In fact, the generalization process in both cases was very similar. The major differences between the two structures is that the DEACON connective rings are ordered but without a starting point, causing Longyear to add extra triangle structures to identify the subject, attribute and value, while the SAMENLAQ and MENS structures inherently distinguish among left name, relation and right name; the MENS structure was approached from a slightly different viewpoint than the triangle structure, resulting, I believe, in the former's being more easily comprehensible as dealing with building blocks of a complex net structure.

The MENS structure is precisely the proper one for the storage and use of Simmons and Burger's "events" [4], which they describe as follows [pp. 4f]:

> The primitive elements of our model are <u>objects</u>, <u>events</u> and <u>relations</u>. An event is defined either as an object or as an event-relation-event (E-R-E) triple. An object is the ultimate primitive represented by a labeled point or node (in a graph representing the structure). A relation can be an

object or an event... Any perception, fact or happening, no matter how complex can be expanded into a nested structure of E-R-E triples.

Use of the MENS structure for this sort of model would be much more natural than the use of lists and list processing languages. Also, since an item of the MENS structure is a relational statement which is directly accessible via the external print names of each of its elements, as well as its own print name, the structure is better than most list structures for question-answering and similar information retrieval systems [5-11]. In fact, in order to directly access relational statements from as many different elements, Levien and Maron had to store each relational statement in four different files [13]. Since the same physical item is used to represent a given structure or print name throughout the memory, substitution of a certain value of all instances of a given variable in a statement is accomplished in one step.

THE "MENS" STRUCTURE

One of the major concepts underlying the SAMELAQ structure is that in the representation of a name-relation-name statement, the relation should be represented in the same way as the names, and be distinguished as a relation only by the way it is linked into the name-relation-name structure. This provides two main benefits: the memory may be entered through a relation as easily as through a name, in fact using the same procedures; a term used as a relation in some statements may be used as a name in others, e.g., to record its properties or its relation to other relational terms. In the graphic representation of this structure (see fig. 1), a pointer goes from the left name through the relation to a slash name which has pointers to all nodes which serve as right names in relational statements with the given left name and relation.

It eventually became apparent that the SAMENLAQ structure was insufficient to handle certain desirable structures, e.g., n-ary relations where $n > 2$, such as "J. Krebb is Vice President in charge of production of Acme Corp."; structures modifying other structures, such as "S. Jones was hired as Production Manager of Acme Corp. in 1965"; storage of complex statements which give information about relations and which may be interpreted as rules of inference for querying the memory, such as "If R is symmetric and xRy then yRx". The solution of these problems seemed to be to represent an xRy statement in the same way that names and relations are represented. Thus relational statements become names or relations in other relational statements. That is what the MENS structure does.

The unit of the MENS structure is an "item", which may represent an unstructured unit or an xRy triple or an Rx double (for unary relations). To achieve this, every item has three pointers - a left name pointer (LNP), a relation pointer (RLP) and a right name pointer (RNP). If an item represents an unstructured unit, it has none of the pointers. If it represents an Rx statement, it has an RLP and an RNP pointing to the items representing R and x respectively. If it represents an xRy statement, it has an LNP,

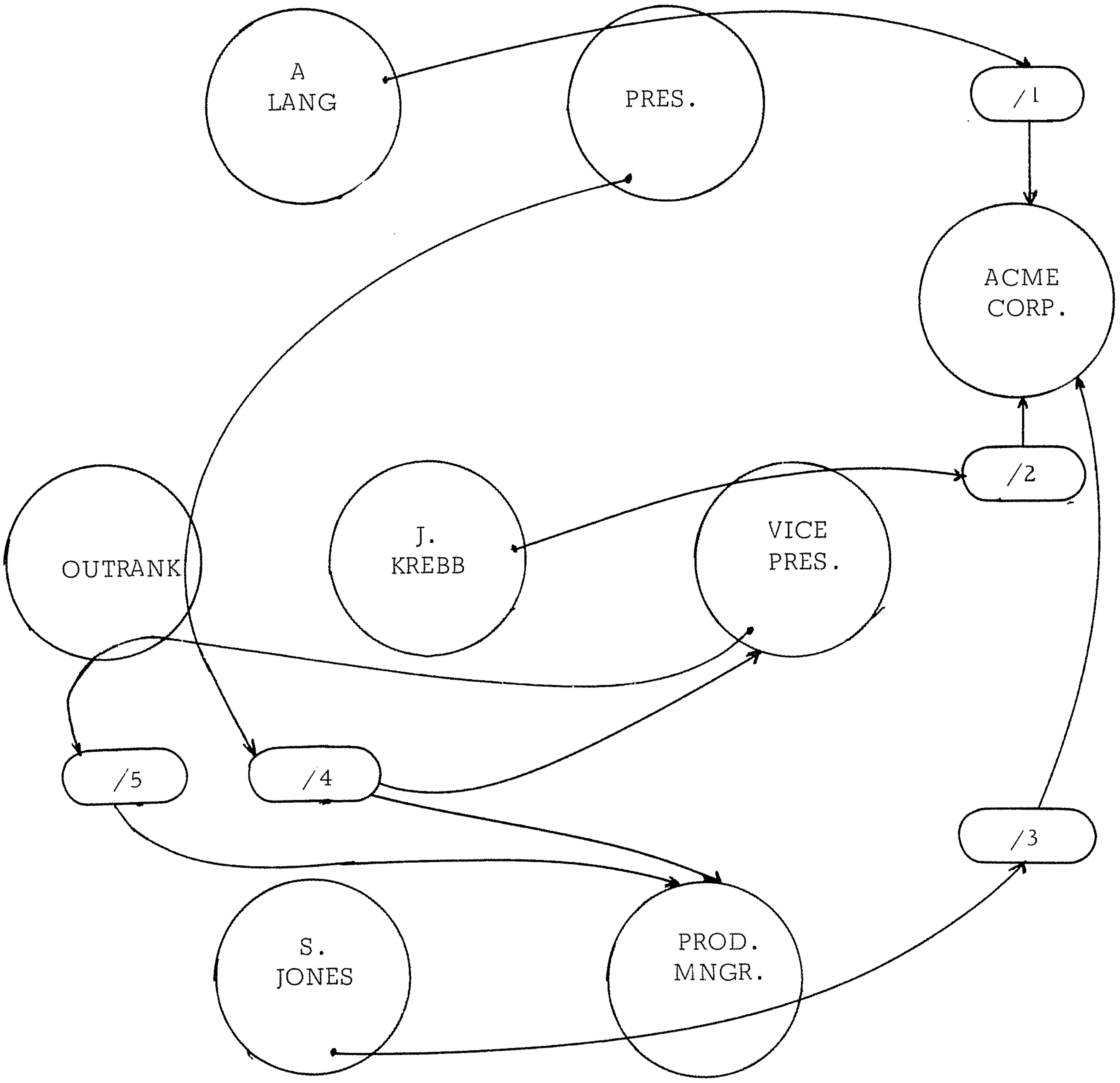

FIG. 1 - An Example of SAMENLAQ Structure

an RLP and an RNP pointing to the items representing x, R, and y respectively. In addition to these pointers there are reverse pointers so that from an item, one can find all structures using that item. These are the left name reverse pointer (LNRP), relation reverse pointer (RLRP) and right name reverse pointer (RNRP). Whenever there is an LNP going from item i to item j , there is an LNRP going from item j to item i . The RLNP's and RNRP's are similarly related to the RLP's and RNP's respectively. Thus, the MENS structure is a directed (generally non-planar) graph, whose vertices are the items, and which has six different sets of arcs, whose existence in the graph is subject to the constraints mentioned above.

We shall call LNP's RLP's and RNP's <u>down pointers,</u> and if either of them go from item i to item j , we shall say that item i <u>points down</u> to item j . Further, if there is a path of down pointers of any length from item i to item j , we shall say that item j is <u>in the structure represented</u> by item i or is <u>used by</u> that structure. Specifically, the <u>structure represented</u> by item i consists of item i , all paths of down pointers starting at item i , all items on those paths, and the reverse pointers for the down pointers on the paths.

Since an item may have at most one of each of the down pointers, but any number of the reverse points, the latter are stored on reverse pointer lists. Each item has (of course) three reverse pointer lists, called LNRPL, RLRPL and RNRPL.

Also needed is some way of distinguishing a complete relational statement from a relational phrase. For example, if the statement ((LANG PRESIDENT ACME) RESIGNED 1967) were stored in the net, we would not want the statement (LANG PRESIDENT ACME) retrieved in answer to some inquiry as to Mr. Lang's occupation. This is done with a FACT flag on each item. If the FACT flag of item i has the value 1, it represents a structure which is an independent statement of fact (as far as the memory net is concerned)*.

In addition, each item has a MARK flag which may be used for various purposes.

Now that the items have been completely described, we may picture them and the MENS structure. Figure 2 shows an item and where the information associated with it will appear. Figure 3 shows how the three troublesome examples mentioned above would appear in the MENS structure.

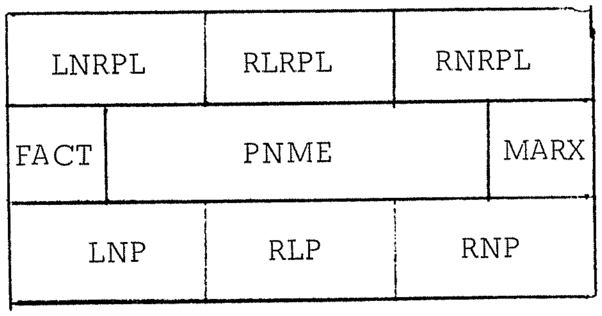

Figure 2 - An Item

*By "independent statement of fact" I mean a statement that may be typed out with no qualifiers. It was either entered in the form in which it appears or was derived from other such statements. Alternatively, we may say that an item block whose FACT flag has the value 1 is the head of a complete structure even though it may also serve as a substructure in another structure.

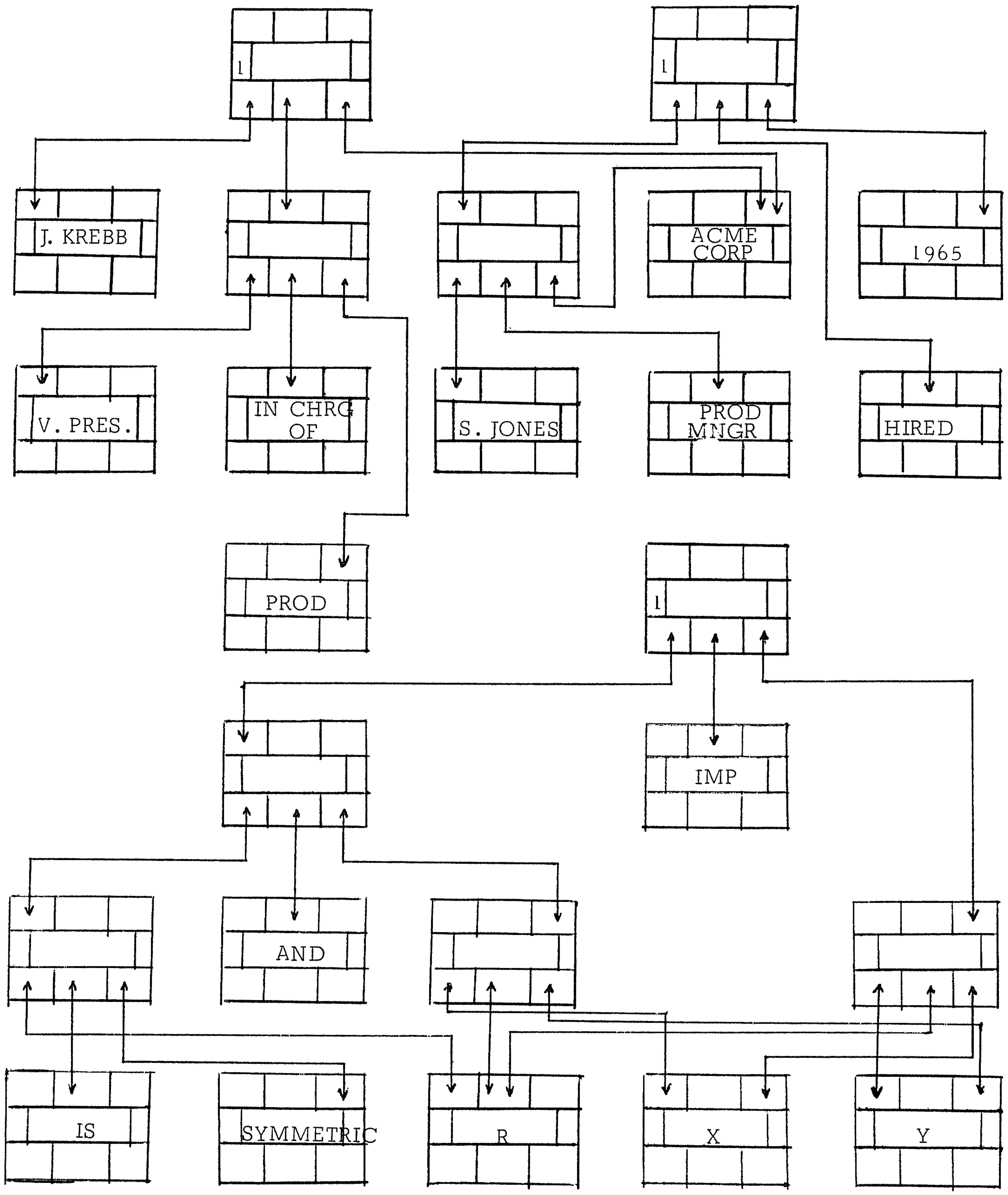

FIG. 3 - An Example of MENS Structure

CURRENT IMPLEMENTATION

As presently implemented, using Burroughs B5500 Extended ALGOL and ASLIP, the items reside in the alpha array ISPCE [1:127, 1:255]. Each item consists of five consecutive words in a row of the array, so that 51 items fit in each of the 127 rows. The internal name of an item, used as a pointer and for certain parameters, is formed in the following way: if the first word of the item is in ISPCE[I, J] then the internal name of the item is the 15 bit number formed by concatinating I & J. The nine fields of each item are located in the five words as follows:

Word	Word Part	Field
1.	[1:1]	MARK
	[2:1]	FACT
	[3:15]	LNP
	[18:15]	RLP
	[33:15]	RNP
2.		PNME
3.		LNRPL
4.		RLRPL
5.		RNRPL

Words 2-5 contain the names of ASLIP lists which contain the appropriate information. The print name is held in a list to allow it to be any number of words of any length.

An available item list is kept, with the variable AVITLST containing pointers to the first and last items on the list. Each item on the available item list contains a pointer to the next one. At first, only the first row of ISPCE is initialized and placed on the available item list. Succeeding rows are added only as they are needed. The number of rows being used is stored in ITROWSUSED. Garbage collection is automatic in that whenecer a pointer is removed from an item, the item is checked to determine if it is empty (has neither down pointers, nor reverse pointers nor a print name). If so, it is returned to the available item list.

To find an item, given its print name, a symbol table is used. The table is in the alpha array SYMTBL [0:1, 0:511]. It is so large in order to minimize the number of names mapped into any entry. Normally a symbol table entry will contain either 0 or a pointer to an item. If, however, two or more names are mapped into one entry, an ASLIP list is created, its name is placed in the symbol table entry and pointers to all the items whose print names were mapped into the entry are placed on the list. The proper item for any print name is found by reading down this list and testing the PNME of each item on it to find one that contains the same name as the one being looked for.

The method of finding a symbol table entry from a print name depends on the fact that the name is stored in the right hand 30 bits of successive ASLIP list elements. The method is:

1) add the list elements forming the name (mod 2^{30})

2) add the 3 successive 10 bit numbers in the 30 bit result from step 1 (mod 2^{10})

3) the left hand bit is the first subscript of SYMTBL, the remaining 9 bits, the second.

A relational statement, as it is structured in the net may be represented in a list as follows: A statement is a list of two or three elements. Each element is either a list containing a string of characters forming a print name, or a statement. Thus the lists representing some of the examples used above are:

```
( (A. LANG)(PRES.)(ACME CORP.))
((PRES.)(OUTRANKS)(VICE PRES.))
((J. KREBB)((V. PRES.)(IN CHRG. OF)(PROD.))(ACME CORP.))
(((S. JONES)(PROD. MNGR.)(ACME CORP.))(HIRED)(1965))
((((R)(IS)(SYMMETRIC))(AND)((X)(R)(Y)))(IMP)((Y)(R)(X)))
```

For communicating with a human at a teletype, the following abbreviations are used when outputting lists and may be used when inputting lists:

[b	for	((
]b	for	))
*b	for	) (

(where "b" stands for at least one blank)

So the last list above may appear:

```
[ [R* IS* SYMMETRIC] (AND) [X* R* Y]  *  IMP*  (Y* R* X)].
```

The MENS structure representing a relational statement is constructed from the list representing the statement. An item is never built that already exists, instead the existing item is used in the structure representing the (possibly) new sentence. The existence of needed items is determined in two ways: (1) At the lowest level of a statement, we want an item with a given print name. We find it, if it already exists, by looking the print name up in the symbol table; (2) At higher levels, we want an item whose down pointers point to a given double or triple of items. We simply take the intersection of the appropriate reverse pointer lists of those items, and if the intersection is nonempty we have the item we seek.

A problem that occurs in the MENS structure is to determine if a given item is in the structure represented by a second given item. That is, if there is a down path from the second item to the first. The straight - forward way to do this would be to search the down paths from the second item and check if the first item is ever reached. Since there are a finite number of items in the structure, the search will end and the question will be decided. However, we may be able to save search time by starting from both items, searching up from the first and down from the second. If they are, in fact, connected, the two searches will meet in the middle. If they are not, we will do extra work to find out. A further consideration, though, is that an item has either zero, two or three down pointers, but may have any number of reverse pointers. Making use of this, the search procedure used decides at each step whether to extend the search down from the top or up from the bottom depending on which has produced less items left to look at; i.e., which search has looked at items with a smaller average branching factor.

PROCEDURES THAT HAVE BEEN WRITTEN

1. Procedures to Manipulate ASLIP Lists*

PROCEDURE REMELT(ELT, LST)

Searches the list LST for an element with the same contents as ELT and deletes it from the list.

INTEGER PROCEDURE LENGTH(L)

Returns the number of elements on the top level of the list L .

BOOLEAN PROCEDURE MEMBER(ELT, LST)

Returns TRUE if there is an element of the list LST whose contents are the same as the contents of ELT, otherwise FALSE.

ALPHA PROCEDURE INTERSECT(LST1, LST2)

Returns the name of a newly created list which contains exactly those elements of both lists LST1 and LST2.

PROCEDURE LUNION(LSTL, LSTR)

Appends on the bottom of the list LSTL all elements of the list LSTR that are not already on LSTL.

ALPHA PROCEDURE SECOND(LST)

Returns a copy of the second element of the list LST (the [3:15] field zeroed out).

*These are procedures that, at the time written, were not part of the ASLIP package.

2. Basic System Procedures for the Memory Net

PROCEDURE INITIT

Initializes the next row of ISPCE by making it a chained available item list with AVITLST pointing to the top and bottom of the chain. ITROWSUSED is increased by one. If no more rows are available, an appropriate message is typed out.

ALPHA PROCEDURE NUITEM

Returns a pointer to a new item; calls INITIT if necessary; initializes the item by setting the first word to zero and putting empty lists in the other four words.

PROCEDURE ERAITEM(ITEM)

Returns the item ITEM to the available item list, removing any reference to it from the symbol table and erasing its PNME, LNRPL, RLRPL and RNRPL lists.

BOOLEAN PROCEDURE ITEMMT(ITEM)

Returns TRUE if LNP = RLP = RNP = 0 and PNME, LNRPL, RLRPL, and RNRPL are all empty, FALSE otherwise.

ALPHA PROCEDURE VSYMGEN

Returns a pointer to a new item and gives the item a unique print name. The print name is a "V" followed by the three characters whose internal machine code is the pointer to the item.

PROCEDURE INITCONS

Initializes those items that other procedures need to refer to (e.g., the items representing the quantifiers of predicate calculus).

PROCEDURE TYPSYMTBL

Types out on the teletype all entries in the symbol table, including the address of the symbol table, the internal name of the item and the print name.

PROCEDURE TYPAVITLST

Types out on the teletype the contents of AVITLST in the format: <last item on the available item list> , <first item on the available item list>.

3. Procedures that Operate on Fields of Items

PROCEDURE SETMARK(ITEM)

Sets the MARK field of the item ITEM to 1.

PROCEDURE SETFACT(ITEM)

Similar to SETMARK.

PROCEDURE REMMARK(ITEM)

Sets the MARK field of the item ITEM to 0.

PROCEDURE REMFACT(ITEM)

Similar to REMMARK.

PROCEDURE REMLNRP(ITEM, PNTR)

Removes the pointer PNTR from the LNRPL list of the item ITEM. If the item is then empty, it is erased.

PROCEDURE REMRLRP(ITEM, PNTR)

Similar to REMLNRP.

PROCEDURE REMRNRP(ITEM, PNTR)

Similar to REMLNRP.

PROCEDURE ADDLNRP(ITEM, PNTR)

Adds the pointer PNTR to the bottom of the LNRPL list of the item ITEM.

PROCEDURE ADDRLRP(ITEM, PNTR)

Similar to ADDLNRP.

PROCEDURE ADDRNRP(ITEM, PNTR)

Similar to ADDLNRP.

PROCEDURE SETLNP(ITEM, PNTR)

If the LNP of the item ITEM is different from the pointer PNTR and not 0, the item it points to has the LNRP pointing to ITEM removed. The LNP of ITEM is set to PNTR and if PNTR is not 0 , a pointer to ITEM is added to the LNRPL of the item PNTR.

PROCEDURE SETRLP(ITEM, PNTR)

Similar to SETLNP.

PROCEDURE SETRNRP(ITEM, PNTR)

Similar to SETLNP.

ALPHA PROCEDURE SETPNME(ITEM, NAME)

This sets the PNME of the item ITEM to be the contents of the list NAME, and makes the appropriate entry in the symbol table. If NAME already has an entry in the symbol table, then if ITEM and the item listed in the symbol table have different down pointers, a message is typed out that an attempt has been made to doubly define NAME, and nothing else is done. If the down pointers are the same or one of the items has no down pointers, the reverse pointers of ITEM are merged with those of the item in the symbol table and one composite item is formed. The procedure returns the internal name of the item whose print name is NAME.

4. Procedures that Return Values of Fields

ALPHA PROCEDURE LNP(ITEM)

Returns the LNP of the item ITEM.

ALPHA PROCEDURE RLP(ITEM)

Returns the RLP of the item ITEM.

ALPHA PROCEDURE RNP(ITEM)

Returns the RNP of the item ITEM.

ALPHA PROCEDURE PNME(ITEM)

Returns the PNME of the item ITEM.

ALPHA PROCEDURE LNRPL(ITEM)

Returns the LNRPL of the item ITEM.

ALPHA PROCEDURE RLRPL(ITEM)

Returns the RLRPL of the item ITEM.

ALPHA PROCEDURE RNRPL(ITEM)

Returns the RNRPL of the item ITEM.

BOOLEAN PROCEDURE MARK(ITEM)

Returns TRUE if the MARK field of the item ITEM is 1, FALSE otherwise.

BOOLEAN PROCEDURE FACT(ITEM)

Similar to the procedure MARK.

5. Procedures that Operate on the Net Structure

ALPHA PROCEDURE LERN(LN, RL, RN)

Returns a pointer to an item whose LNP points to the item LN, RLP points to the item RL and RNP points to the item RN. LN may be zero.

PROCEDURE MARKSTRUCT(ITEM, ONOROFF)

Sets the MARK fields of all items in the structure represented by the item ITEM to 1 if the Boolean parameter ONOROFF is TRUE, to 0 if ONOROFF is FALSE.

ALPHA PROCEDURE DPRLST(ITEM)

Constructs an ASLIP list containing the relational statement expressing the structure represented by the item ITEM, and returns the list's internal name.

PROCEDURE DNPRSTRDC(ITEM)

Causes the list DPRLST(ITEM) to be typed out on the teletype.

ALPHA PROCEDURE STRUCTURE (SENT)

Sets up the MENS structure representing the statement in the list SENT and returns a pointer to the top item.

BOOLEAN PROCEDURE STRUCMEM(ITT, ITS)

Returns TRUE if the item ITT is in the structure represented by the item ITS, FALSE otherwise.

PROCEDURE ERASE(ITEM)

Removes LNP, RLP and RNP of the item ITEM and the appropriate reverse pointers. If any items are thereby made empty, they are returned to the available item list by ERAITEM.

ALPHA PROCEDURE COPYNSUB(STRUC, ITN, ITO)

Returns a pointer to an item which represents the same structure as does the item STRUC, except the item ITO is replaced by the item ITN.

ALPHA PROCEDURE REWRITE(WFF)

Changes the structure represented by the item WFF so that, interpreted as a formula of the predicate calculus, all instances of each bound variable are a new symbol created by VSYMGEN. The quantifiers are expected to be represented by "A" and "E".

6. Main Programs

PROCEDURE MENTAL

This is the beginning of a basic question-answering program using the MENS structure. The presently available inputs are:

(FACT <name>) <statement> .

Creates the MENS structure representing the statement contained as the list <statement>, sets the FACT field of the top item to 1 and sets the PNME of the top item to (<name>).

(REQUEST STOP) ()

Exit is made from MENTAL.

(REQUEST PRINT) (<name>)

The list expressing the structure represented by the item whose print name is <name> is typed out.

(REQUEST REWRITE) (<name>)

The procedure REWRITE is called with the argument being the item whose print name is <name>.

The main program is a test program allowing all the procedures listed above to be called from a remote teletype. Initially the user must type any input. He may then call any of the above procedures by typing the name of the procedure followed by a comma if the name is six or more letters long, or enough blanks to make a total of six characters. This is followed by the desired parameters typed as the eight BCL characters which form the appropriate computer word. The parameters are separated by commas. (This form of input may not be the best, but it was never intended to be used by anyone but myself.) When ALPHA or BOOLEAN procedures are called, their values are typed out. Other procedures which do not themselves type something have "DONE" typed to signify their completion. In addition to the above procedures, the ASLIP procedures RDLSTDC(REFCNT) and PRLSTDC(LST) may be called in this manner.

A PROPOSED NET MANIPULATING LANGUAGE

Although there already exists a set of procedures to manipulate the MENS structure as described above, they were written to experiment with the feasibility of the structure, rather than for general use in using it. There reamin a number of questions to be resolved before building a programming language to manipulate a MENS net.

The first question is: who will be the users of the language? They will be the programmers who would now use lists and other data structures for theorem proving, question-answering and semantic analysis of natural languages. Thus we may expect programming sophistication and knowledge of other programming languages of our users. A different class of users is the social scientists who construct, manipulate and query directed graphs in their work. The MENS structure is also suitable for them, but we cannot assume they have much programming sophistication.

The best thing for the latter users would be to use the net manipulating language to build an interactive information storage and retrieval system that they can interact with using their own jargon. I intend to build such a basic system as a demonstration of the use of the net manipulating language.

The language will be a package of procedures imbedded in some other high level language (e.g., ALGOL), thus giving the user full flexibility in constructing his system. The package that already exists will be a start, but must be revised so that each procedure does a single, well defined job as efficiently as possible.

To get this maximum efficiency a new look will be taken at the basic representation of the MENS structure. Certainly, list processing capabilities will be needed. These are presently supplied by ASLIP, which was designed so that it could be implemented quickly on the B5500. It is very possible that a different type of list processor will be more efficient in the MENS system.

In fact, a very frequest process in the current method of working on MENS is the intersection of reverse pointer lists when looking for items representing certain structures. As presently done, intersecting lists is a rather slow process. One possible improvement would be to have all reverse pointer lists ordered. This would make searching a list faster, and the extra time needed for storing a list element would be more than made up by the saving achieved when doing the multiple searches necessary for intersecting. Perhaps, though, a more basic change in the representation of an item would eliminate the need to intersect lists entirely.

Other possibilities to be investigated when designing the language are:

Keeping track of how many pointers are associated with an item would eliminate checking for empty lists before replacing an item onto the available item list.

Having special structures to represent numerals might be necessary. We would want to be able to store and operate on numerical information without having to represent all numbers.

Special arrangements may have to be made to represent and utilize variables. We would want to be able to build a structure with one or more variable items to represent different structures with the variables replaced with different items. We would want the system to have some natural way of doing this, and of identifying from such a structure all items which could replace the variable items in the structure.

With these things worked out, a net manipulating language could be constructed which would be a highly useful aid for complex information processing.

REFERENCES

1. Shapiro, Stuart C., Woodmansee, G. H., Krueger, Myron W., "A Semantic Associational Memory Net that Learns and Answers Questions (SAMENLAQ)", Computer Sciences Department Technical Report #8, The University of Wisconsin, Madison, Wisconsin, January, 1968.

2. Craig, James A., Berezner, Susan C., Carney, Homer C., Longyear, Christopher R., "DEACON: Direct English Access and CONtrol", AFIPS, Vol. 29, Proceedings of the Fall Joint Computer Conference, 1966, pp. 365-380.

3. Longyear, Christopher R., "Memory Structure in DEACON Natural Language Question-Answering Systems", P-129, General Electric Company, TEMPO, Santa Barbara, Calif., 1966.

4. Simmons, Robert F., Burger, John F., "A Semantic Analyzer for English Sentences", SP-2987, System Development Corporation, Santa Monica, California, 1967.

5. Elliott, Roger W. "A Model for a Fact Retrieval System", unpublished Ph.D. Dissertation, University of Texas, 1965.

6. Green, B. F., Wolf, A. K., Chomsky, C., Laughery, K., "Baseball: An Automatic Question Answerer", in (12), pp. 207-216.

7. Levien, R., and Maron, M. E., "Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval", #RM-4793-PR, The RAND Corporation, Santa Monica, California. 1965.

8. Quillian, M. R., "Semantic Memory", unpublished Ph.D. dissertation, Carnegie Institute of Technology, 1966. Also #AFCRL-66-189, Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, 1966.

9. Quillian, M. R., "The Teachable Language Comprehender: A Program to Understand English", unpublished paper, Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, March, 1968.

10. Raphael, R., "Sir: A Computer Program for Semantic Information Retrieval", unpublished Ph.D. dissertation, Massachusetts Institute of Technology. Also #TR-2, Project MAC, M.I.T., Cambridge, Mass., 1964.

11. Feigenbaum, E. A. and Feldman, J., (eds.) Computers and Thought, McGraw-Hill, New York, 1963.

12. Levien, R. E. and Maron, M. E., "A Computer System for Inference Execution and Data Retrieval", Communications of the A.C.M., Vol. 10, #11, November, 1967, pp. 715-21.