A NET STRUCTURE FOR SEMANTIC INFORMATION STORAGE, DEDUCATION AND RETRIEVAL*

by

Stuart C. Shapiro
Computer Sciences Department
University of Wisconsin
Madison, Wisconsin, U.S.A.

Abstract

This paper describes a data structure, MENS (MEmory Net Structure), that is useful for storing semantic information stemming from a natural language, and a system, MENTAL (MEmory Net That Answers and Learns) that interacts with a user (human or program), stores information into and retrieves information from MENS and interprets some information in MENS as rules telling it how to deduce new information from what is already stored. MENTAL can be used *as* a guestion-answering system with formatted input /output, as a vehicle for experimenting with various theories of semantic structures or as the memory management portion of a natural language question-answering system.

1. Introduction

In order to develop machines capable of "understanding" natural language, it is extremely valuable, if not necessary, to design a method of organizing a corpus of data to facilitate the storage and retrieval of information on many subjects, some in depth, some in breadth; to facilitate the storage, retrieval and use of the many complex relationships among real-world concepts; to facilitate the storage, retrieval and use of information which tells how other information in the corpus may be used to further explicate implied relationships among concepts; and to facilitate the identification from the vast corpus of data of those pieces of information most directly relevant to any given topic.

This paper describes a data structure (MENS) and procedures for manipulating it (MENTAL) that have been designed to meet the requirements outlines above. This system is intended to be used as the memory of a natural language question answering machine (see [7;8;9;10]) and could also be used as the memory of a general theorem prover or problem solver. Since the system allows its user (either a human or an outside program) to specify the relations that will be used for the basic structuring of information, the system can be used for experimenting with data structures suitable for various contents and purposes. The major features of the data structure are;

It is a net whose nodes represent conceptual entities and whose edges represent relations that hold between the entities.

A distinction is made between n-ary relations about which information and deduction rules are to be stored and strictly binary relations that are used only to structure information about other entities. The former are represented by nodes in the net, just like any conceptual entity. The latter relations are the ones used as th edges of the net.

Some nodes of the net are variables, and are used in constructing general statement, and deducation rules.

Each conceptual entity is represented by exactly one node in the net from which all information concerning that entity is retrievable.

Nodes can be identified and retrieved eithe by name or by a sufficient (though not necessarily complete) description of their * connections with other nodes , likewise identified.

The system and data structure described here follow along the general lines laid out by such systems as Semantic Memory [11], TLC [12], Protosynthex II and III [14,19,20], GRAIS [3] and SAMENLAQ [17,18], but differ mainly in the clear separation of the two levels of relation, and in the ability to store and use general deduction rules.

All the procedures for storing information into the data structure, as well as all those for explicit retrieval and some of those for implicit retrieval have been programmed in PL/1 and are running interactively on an IBM System/360. Al the research reported herein has proceeded both theoretically and by writing, checking out, revising and improving programs in PL/I, SNOBOL3 and Burroughs Extended ALGOL.

The research reported herein was partially supported by a grant from the National Science Foundation (GJ-583) and partially by USAF Proj. RAND (project #1116). Use of the University of Wisconsin Computing Center was made possible through support, in part, from the National Science Foundation and the Wisconsin Alumni Research Foundation (WARF) through the University of Wisconsin Research Committee.

A more detailed discussion of MENS and MENTAL which also shows their applicability as an experimental vehicle is given in [16].

2. Basic Concepts of the Structure

To explain the MENS structure, it is first necessary to explain what is meant by a conceptual entity. A conceptual entity is anything about which information can be given. That is, anything about which one can know, think, or believe something; anything we can describe or discuss or experience. Individual objects, both concrete and abstract, are conceptual entities, as are classes of objects, actions and classes of actions. Conceptual entities are really relative to individual intelligent beings. My concept of a given person, named John 0. Smith, may be different from yours. Therefore, "John 0. Smith" names a different conceptual entity for me than it does for you. Naturally, a word is not a conceptual entity, but may designate or refer to zero or more conceptual entities for each intelligent hearer. MENS has been designed to store information about conceptual entities.

The information that is stored about a conceptual entity is semantic information, in that that meaning of a given conceptual entity for a given being is whatever that being can say about the conceptual entity. This includes everything that being can infer from any further piece of information about the conceptual entity.

The basic goal underlying the design of MENS has been generality. It was desired to be able to store any conceptual entity, regardless of its field of knowledge, and any information about the conceptual entity. Of course, stored informayon is useless unless it can be retrieved, so ease and efficiency of retrieval was a major goal along with adequacy of representation.

Assuming the above discussion, the basic motivating factors for the design of MENS were:

1. Unified representation: All conceptual entities about which information might be given and questions might be asked should be stored and manipulated in the same way.

2. Single file: All the information about a given conceptual entity should be reachable from a common place.

3. Multientried, converging search: A search of the file should start from as many places as possible and proceed in parallel, converging on the desired information.

4. Storage of deduction rules: Rules determining how deductions may be made validly, even when specific to certain areas or relations, should be stored in the memory file just like other information, and the system should be able to use them in directing its deductive searches.

5. Direct representation of n-ary relations: N-ary relations, for any n, should be as natural for the system as binary relations.

6. Experimental vehicle: The file should be designed without any commitment to a particular semantic theory, i.e. the memory system should be a research vehicle for experimentation on various ways of structuring the information in it.

In the rest of this section, we will describe how these motivating factors led to the particular structure decided upon.

Unified representation requires that every conceptual entity have a memory structure representation which can be put into relationships with representations of other conceptual entities. It further requires that all conceptual entities be represented in the same way regardless of their exact relationships to other conceptual entities. We will refer to a conceptual entity or to the logical representation of a conceptual entity as an Item. When referring to the computer structure used to implement the representation of a conceptual entity, we will use the term item block. In illustrations, we will picture an item block as a rectangle within which we will place an English word to indicate what concept the item block represents. If no such word exists some other symbol may appear so that the item block may be referred to. The full implication of unified representation is that every word sense, every fact and event, every relationship that is to be a topic of discussion between the system and its human discussant will be represented by an item. Therefore, the items must be tied together by relationships that are not conceptual entities. The reasoning for this is as follows. Statements (e.g., "Brutus killed Caesar.", "The sky is green.") are conceptual entities since we may say things about them such as someone believes them or they are false. Therefore, they must be represented by items, and such an item must bear some relation to the items (Brutus, kill, green) that make up the statement. If this latter relation is a conceptual relation, the fact of this relationship's holding between two items may be discussed and thus must be represented by an item which then must have some relationship to that relation, etc. Eventually there must be some relation which is not conceptual, but merely

structural, used by the system to tie a fact-like item to the terms partaking in it. We will refer to a conceptual relation as an item relation or simply a relation and to a nonconceptual relation as a system relation, link, or pointer. The MENS structure is, thus, a collection of items tied together *by* system relations into a directed graph with labelled edges. The nodes of the graph are the items and the edges are system relations. The edges are directed to indicate the order of the arguments of the system relation. The edges are labelled to allow for several different system relations. The distinction between item relations and system relations is very important and must be kept in mind.

Single file means that there will be exactly one item for each conceptual entity. Therefore, all the information about the conceptual entity will involve its item and be retrievable from its item block. Since the system relations are the links that tie items together and thus provide the information, this means that whenever a link goes from one item to another, there is an associated link in the reverse direction. Looking at the fact and event items as records in a record oriented file and at the links going from participating items to fact and event items, MENS is an inverted file and may be searched as one. However, it is more than an inverted file, since links go the other way also. In illustrations, a link pair is represented by a line connecting two item blocks. The name of the system relation appears in the item block where the line emanates, from it. Tor example in Figure 2 the system relation AGENT goes from item 241/00010+023 to the item representing JOHN.

Multientried, converging search implies that items equally identifiable by the human conversant should be equally identifiable by the system. By this is meant that any item named *by* an English word can be located as quickly (by the same lookup procedure) as any other item so named, rather than some being locatable by lookup while others require an extensive search. Items that do not have English names, but must be identified by description will be located via searches that are quick or involved depending on the complexity of the description. The lookup is done through a dictionary which gives the internal names for the items which represent each of the senses of each natural language word used in the conversations. The internal name of an item is its address in secondary storage, so once looked up the item block is easily found. Items are connected to facts (which do not have English names) as mentioned above and when two items are connected in the memory structure, each is reachable from the other since every link between two item blocks is stored in both directions. Another implication of multientrled, converging search is that searching the file is done by starting at an arbitrary number of item blocks (all those that can be looked up directly) and converging to the desired information structures. This involves repeated intersections of sets of items as will be explained in section 3. Special care has been taken to make this search process as efficient as possible and special constructs have been developed for this purpose.

Storage of deduction rules implies that deduction rules* should be capable of being stored in and retrieved from the memory structure in the same way that specific information is stored and retrieved. This implies that the structures used to store deduction rules must be basically the same as those used to store specific information. It further implies that the executive routines must include a very general deduction rule interpreter that is capable of initiating searches of the memory and generating appropriate consequences based on any stored deduction rule.

Direct representation of n-ary relations implies that an item representing a relational statement based on an n-ary relation should have links to each of its n arguments directly, regardless of the value of n. This means that any item must be capable of having an arbitrary number of pointers emanating from it. This number may even change throughout the life of an item as the types of system relationships it has with other items change.

There are several reasons for representing n-ary relations directly. It makes searching more efficient than if n-ary relations were stored as nested binary relation. If nested binary relations were used, item blocks would be introduced that were not true conceptual entities. Also Fillmore's case grammar theory [4,5] in which the deep structure of a sentence contains a predicate of a

By "deduction rule" is meant any statement which, properly interpreted, provides information as to what statement(s) may be concluded from what other statement(s). Deduction rules include (among others) rules of inference of symbolic logic, general statements and disjunctive statements (any clause may be the conclusion if the negation of all the others holds).

verb and n cases is a satisfying theory for a semantic deep structure and representing n-ary relations directly in MENS allows for a direct representation of Fillmore's deep structures.

Experimental vehicle implies that the user must be given the capability of declaring what and how many system relations he will use rather than having a maximum number imposed on him. He must be able lo decide what will be his conceptual entities rather than be provided with a closed set of them. He must be able to decide how items and pointers will be combined into structures to represent the information he wishes to work with. He must, finally, not be restricted as to what deduction rules the system may store and use.

3. Explicit Storage and Retrieval

Both storage into and retrieval from MENS are accomplished by describing how an item is (or is to be) connected to other items in the net. The storage instruction in effect says, "Create an item and connect it into the net in this way." The retrieval instruction in effect says, "Tell me all items that are connected in the net in this way." Both instructions are expressed in a statement, called a spec , which describes the item by describing the paths in the net that lead away from the item. These paths may be quite complicated, but the edges along the paths must be explicitly named system relations.

A subset of the input language sufficient for discussing the main points of MENS and MENTAL is defined in fig. 1 in modified BNF notation. Underlined words in lower case letters are non-terminal characters. Strings enclosed in square brackets are optional. Strings arranged vertically and surrounded by braces are alternatives—one must be chosen. Strings followed by an asterisk may appear one or more times. Strings surrounded by broken brackets are informal English descriptions of object language strings. represents a required blank; additional blanks may be inserted anywhere. The following characters are delimiters in the language: ,——1) (? . : ' = %. A "character" is any legal character except a delimiter.

A relspec is used to declare a system relation. For example the relspec

$AGENT S *AGENT M

declares a system relation whose forward pointer is named AGENT and whose reverse pointer is named *AGENT. It further declares that an item may point to at most one other via the AGENT pointer, but to an arbitrary number of other items via the *AGENT pointer. AGENT is therefore said to be a singular pointer and *AGENT, a multiple pointer.

After several such relspecs, a user might input the following buildspecs:

(.AGENT:J0HN,VERB:LøVES,0BJ:JANE)
(.AGENT:JANE,VERB:LØVES,ØBJ:JØHN)
(.AGENT:SUE,VERB:LØVES,ØBJ:J0HI\J)

This would cause the structure shown in figure 2 to be built. Symbolic names are shown on those item blocks that have them. Three item blocks were built to represent the three facts. These are shown with their internal names. The significance of the internal name 241/00010+ 023 is that the block is stored on disk track 10 at an offset of 23 within the track and that the calculated specificity measure is 241. The specific] tv measure is used in deductive processing and will be discussed later.

Item blocks are stored on disk tracks as consecutive blocks of records . Each record contains the coded name of a pointer name and, if the pointer is singular, the internal name of the item block pointed to. If the pointer is multiple, the other field of the record is a link to a multiple pointer list which contains the internal names of all the item blocks being pointed to. Each multiple pointer list is ordered on the internal names it contains (largest first) and is kept on the same disk tracks as the item block it is for.

Returning to the discussion of explicit storage and retrieval, items may be described as well as named in specs. Tor example the buildspec

(.A:HENRY,V:SAW,Ø:(.A:J0HN,V:HIT,0:JIM))

(assumming the proper relspecs had been given) would result in the building of the structure of figure 3. In this case, both the blocks shown unlabelled in the figure would have been created in response to the buildspec as well as all the pointers. If we had already input the information that John hit Jim and wanted to say that Henry saw that act, we would use the following buildspec:

(.A:HENRY,V:SAW,0:(?1,1,A:J0HN,V:HIT,0:JIM))

which uses an imbedded findspec.

The findspec causes a list of items to be found that satisfy the given description. In this case the *A multiple pointer list from the block named J0HN, the *V list from HIT and the *0 list

input → { relspec / spec }

relspec → $ linkname ⌀ { S / M } linkname ⌀ { S / M }

restrictions → (linkname [, linkname]*)

linkname → <a string of 1 to 13 characters>

spec → { buildspec / findspec }

buildspec → (. linkname : spec [, linkname:spec]*)

findspec → { name / (name[, name]*) / (findprefix linkname : spec [, linkname : spec]*) }

findprefix → ? num, num { (vname) / . }

name → { cname / 'vname / %vname }

cname → { <a string of 1 to 13 characters> / <3 digits>/<5 digits>+<3 digits> }

vname → { <a string of 1 to 13 characters, the first of which is not X, Y, or Z> / <a string of 1 to 13 characters, the first of which is X, Y, or Z> }

num → { <any integer i, 0 ≤ i ≤ 2^{31}> / # }

FIGURE 1: SYNTAX OF THE USER'S LANGUAGE

from JIM would be intersected to find the names of all item blocks fitting the description. The findprefix, "?1,1,", specifies that there should be at least 1 and at most 1 items found. That is, this findprefix declares the findspec to be a definite description. To retrieve all instances of Henry's seeing John's hitting of Jim, the findspec

(?0 ,#,A:HENRY,V:SAW,0:(?O ,#,A:JØHN, V:HIT,Ø:JIM))

could be used. The imbedded findspec could here return a list of zero or more items. The union of the *Ø lists from these items would be intersected with the *A list from HENRY and the *V list from SAW to satisfy the full request. Thus, the unioning and intersecting of ordered lists is the basic mechanism for explicit retrieval from MENS. It was thus necessary to develop efficient methods for taking unions and intersections. These methods are described in [15],

A more complicated retrieval mechanism is needed for structures in which there are two or more separate paths to one unknown item. For example the structure in figure *4* represents the information that Narcissus loves himself. To find all items such as the top one, i.e. all items representing the fact that someone loves himself, the findspec

(?0 ,#,A: %N,V:LØVES,Ø:'N)

could be used. Here N is a variable of the users language. The items on the *V list from LØVES will be retrieved and from each one the item

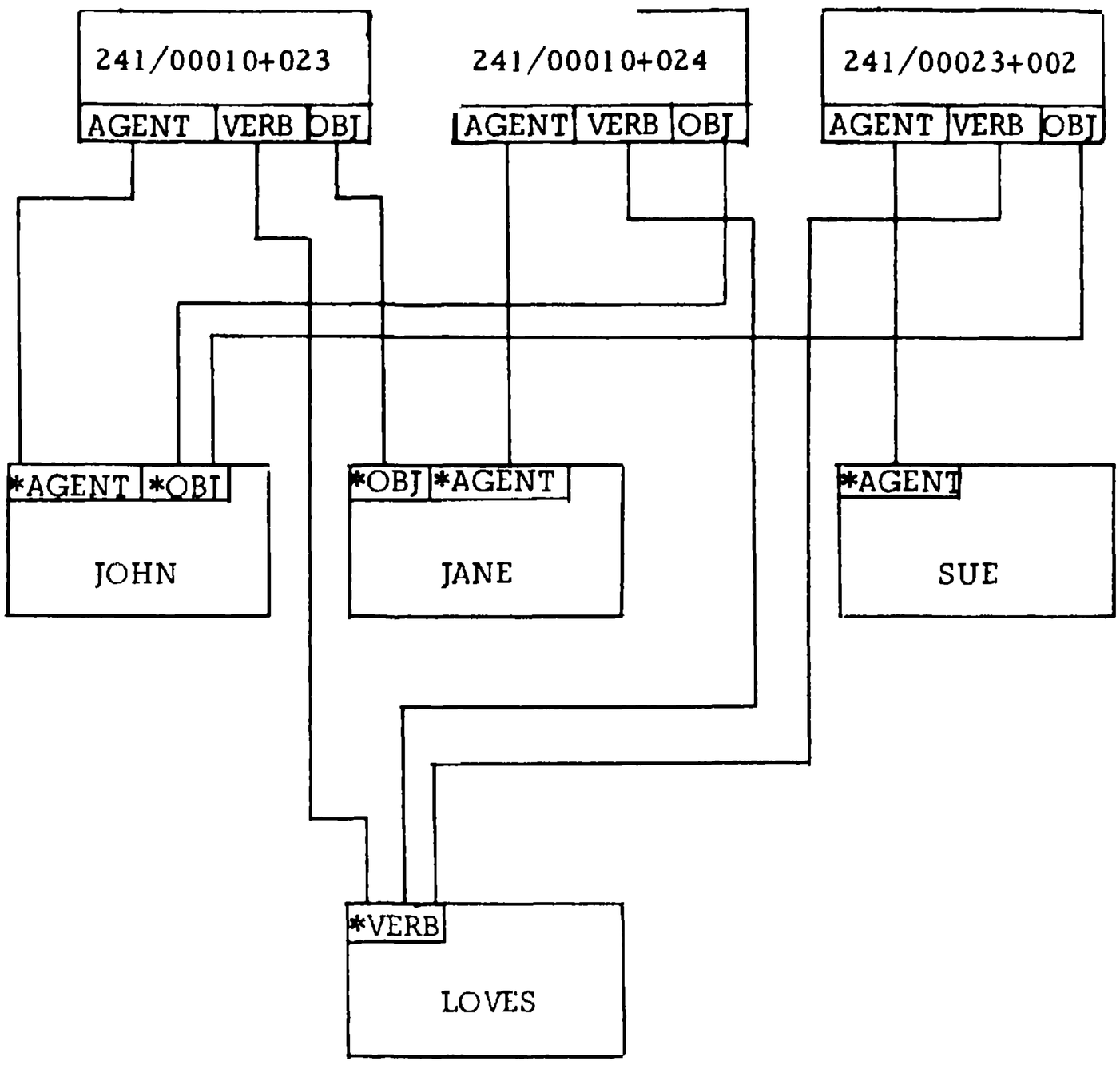

FIGURE 2: A MENS SUBSTRUCTURE, DESCRIBED IN THE TEXT.

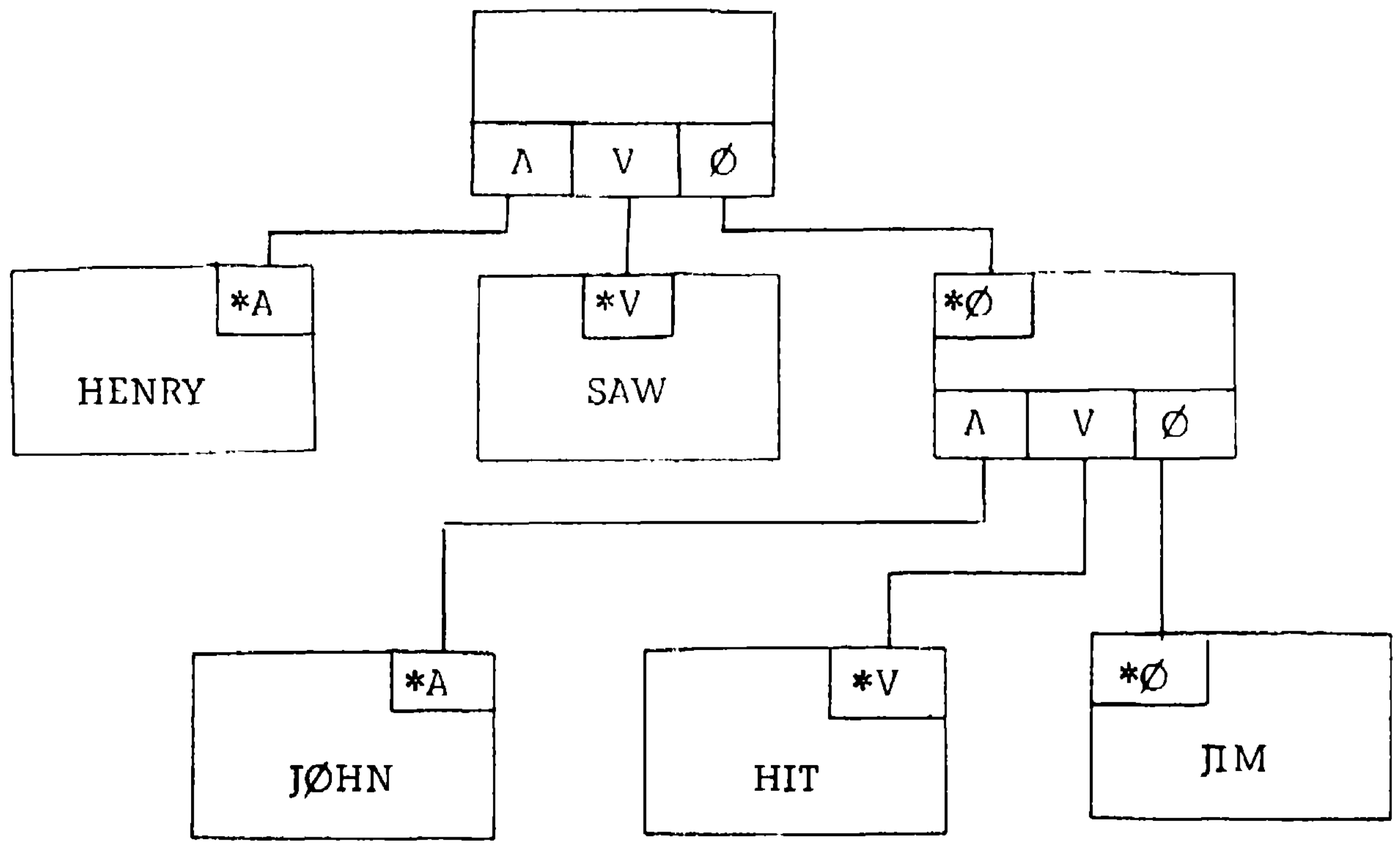

FIGURE 3: ANOTHER MENS SUBSTRUCTURE

pointed to via A will be checked for equality with the item pointed to via 0. The list formed as a result of the findspec will contain only those items for which the check succeeded and for each one, it will be recorded which item was found to substitute for N. This is done so that in case the findspec is embedded in another that also contains N as a variable, checks will continue to be made to insure that for each structure retrieved, N represents exactly one item. A complete analysis of how these checks should be made and how the union and intersection operations can be modified to handle them is given in [16].

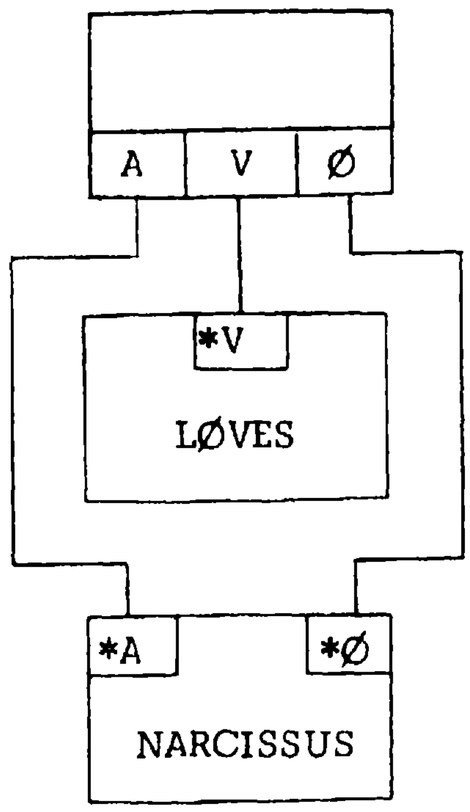

FIGURE 4: A STRUCTURE WITH TWO ITEMS CONNECTED BY TWO DISTINCT PATHS.

4. Representation of Deduction Rules

In section 3 it was shown how the MENS structure is used for explicit storage and retrieval. In this section we will explain how it can be used for deduction. Since storage of deduction rules is a motivating factor of this project, the deduction method will involve the storing of general deduction rules and the use of fairly simple theorem proving techniques. The reason for this is that we want the system to be as qeneral as possible and we want to concentrate on the data structure rather than the executive routines. It would be possible to build a complex and sophisticated theorem prover which uses MENS for its data storage, but this is not our current interest.

In order to allow for complete generality in what deduction rules could be stored, including arbitrary orderings of arbitrarily many quantifiers, it was decided to represent quantifiers and variables directly in the structure, and build executive routines to interpret the deduction rules. These routines would operate, upon being given a deduction rule, by carrying out searches required by the rule and building consequences justified by the rule. Representing quantifiers and variables directly seems to be a compromise of the motivating factor of unified representation since they will require special routines to deal with them and their status as conceptual entities is questionable. However, dealing with the order of quantification implied by some English sentences is enough of a problem that at least one linguist believes that quantifiers and variables might profitably be comprehended by the base rules of English grammar [], p. 112]. Besides, including this capability extends the use of the system as an experimental vehicle, another motivating factor.

The decision to allow direct representation of variables leads to the questions of how to represent them and what will be allowed to substitute for them. Considering the second question, the conclusion is that a variable should be able to stand for any item but not for any system relation. This is supported by the discussion in section 2 that anything about which information could be given should be represented by an item, that all items should be equally able to have information stored about them, and that system relations could not have information given about them since they are not conceptual entities . As Quinesays, "The ontology to which one's use of language commits him comprises simply the objects that he treats as falling . . . within the range of values of his variables." [13, p. 118 quoted in 2, p. 2]4]. Since the ontology of the data structure comprises the set of items (by definition of item), the values of the variables must be allowed to range over all the items, and since the system relations are to be excluded from the ontology, not allowing them to substitute for a variable reinforces their exclusion. Allowing the variables to range over all the items, however, brings up the possibility of storing the paradoxes that were eliminated from formal languages only with the introduction of types of variables or restrictions on assertions of existence (of sets). This possibility will be accepted. We make no type distinctions among the items and impose no restraints on item existence, leaving the avoidance of paradoxes the responsibility of the human informant. We will do the same with the variables. However, we do use restricted quantification. What is meant by this is that with each quantifier in a deduction rule will be included, not only the variable it binds, but also an indication of the set of items over which the variable ranges. Woods [2I] uses restricted quantification to reduce the time needed to handle a request by including in the restriction a class name and a predicate. The class name must be of a class for which there exists a generator that enumerates all the members of the class one at a time. Each member is tested with respect to the predicate. Those for which the predicate is true are acted on by the main body of the request. Our restrictions may be more general. We will allow any statement, however complex, about the variable. This statement will be used as a search specification to find all items in the structure for which the statement is true. The set of such items will comprise the range of the variable. Thus, even omega ordered type theory may be represented in the structure by entering a statement about every item giving its type and including type

We now return to the question of how variables should be represented. Each variable will be represented by its own item block. All occurrences of the same variable within a given deduction rule will be represented by the same item and no such item will be used in more than one deduction rule. The same item is used for all occurrences of a variable in a deduction rule so that a substitution made for the variable in one occurrence will at the same time be made in the others and so that all the information about what items can substitute for the variable will be reachable from one place. Different items are used in different deduction rules to eliminate the possibility of information about a variable in one deduction rule becoming associated with a variable in another. The specification measure field of the internal name of an item is used to distinguish variable items from constant items so that an item can be recognized as a variable when it is pointed to from another item.

Besides quantifiers and variables, the connectives NOT, AND, OR, IMPLIES, IFF and MUTIMP* are also represented as item relations in the structure and the executive routines that interpret the deduction rules are designed to handle them.

Deduction rules are stored using two types of items that will be recognized by the executive routines. We will call them quantifier clauses and connective clauses. A quantifier clause is the head of a quantified general statement and has four special system relations emanating from it. They are:

(0 Q points to the quantifier
(ii) VB points to the variable being bound
(iii) R points to the restriction on the variable
(iv) S points to the scope of the quantifier

MUTIMP stands for mutual implication. It is a predicate with an arbitrary number of arguments and says that its arguments mutually imply each other by pairs (are pairwise equivalent). Looked on as a binary connective, MUTIMP, like AND and OR and unlike IMPLIES and IFF is idempotent as well as associative and commutative. A possible definition of MUTIMP is:

$$\text{MUTIMP}(P_1,\ldots,P_n) \underline{\text{df}} \bigwedge_{i=1}^{n} (P_i \text{ IMPLIES } \bigwedge_{\substack{j=1 \\ j\neq i}}^{n} (P_j))$$

That is if MUTIMP $(P_1,\ldots,P_n)$ is true and P_i is true (false) for some i, $1 \leq i \leq n$, then P_i is true (false) for all i, $1 \leq i \leq n$. For two arguments MUTIMP is equivalent to IFF.

A connective clause is the head of a construction formed of several clauses joined by one of the connectives mentioned above. It has an OP system relation to the connective and one of the following sets of argument relations:

(i) ARG to the argument if the connective is unary (NOT)
(ii) ARG1 to the first argument and ARG2 to the second argument if the connective is binary (IMPLIES ,IFF)
(iii) MARG to all the arguments if the connective is associative, commutative and idempotent (AND,OR,MUTIMP)

The clauses forming the arguments of a connective clause and those forming the restriction and scope of a quantifier clause may be any net sub-structure with the requirement that a clause may contain a free variable only if a path of converse argument pointers, converse restriction pointers and converse scope pointers leads to a quantifier clause in which that variable is bound.

Examples of deduction rules are given below. Each deduction rule is given first as an English language statement and then as a buildspec. names such as 'X represent variable items .

1 . Every man is human.
(.Q:ALL,VB:'X,R:(.AGENT:'X,VERB:MEMBER, OBJ:MAN),S:(.AGENT:'X,VERB:MEMBER, OBJ:HUMAN))

2. Every car has-as-part an engine.
(.Q:ALL,VB:'X,R:(.AGENT:'X,VERB:MEMBER, OBJ:CAR),S:(. Q:EXISTS, VB:'Y,R:(. AGENT: 'Y/VERB:MEMBER/OBJ:ENGINE), S:(.AGENT: •X,VERB:HAS_AS_PART,OBJ:'Y)))

3. If a male is the child of someone, he is the son of that person.
(.Q:ALL^rXRrt.AGENTr^VERBrMEMBER, OBJ:MALE),S:(.Q:ALL,VB:'Y,R:(.ACENT:'X, VERB:CHILD_OF,OBJ:'Y)S:(.AGENT:'X,VERB: SON_pF,OBJ:'Y)))

4. John is at home, at SRI or at the airport.*
(.OP:OR,MARG:(.AGENT:JOHN,VERB:AT, OBJ:JOHNS_HOME), MARG:(.AGENT:JOHN, VERB:AT,OBJ: SRI), MARG:(.AGENT:JOHN, VERB:AT ,OBJ: AIRPORT_4))

The structure for rule 1 is shown in figure 5.

5. Use of Deduction Rules

There are six operations that can be performed with respect to a deduction rule in MENS. They are:

*This sentence taken from Green and Raphael [6]

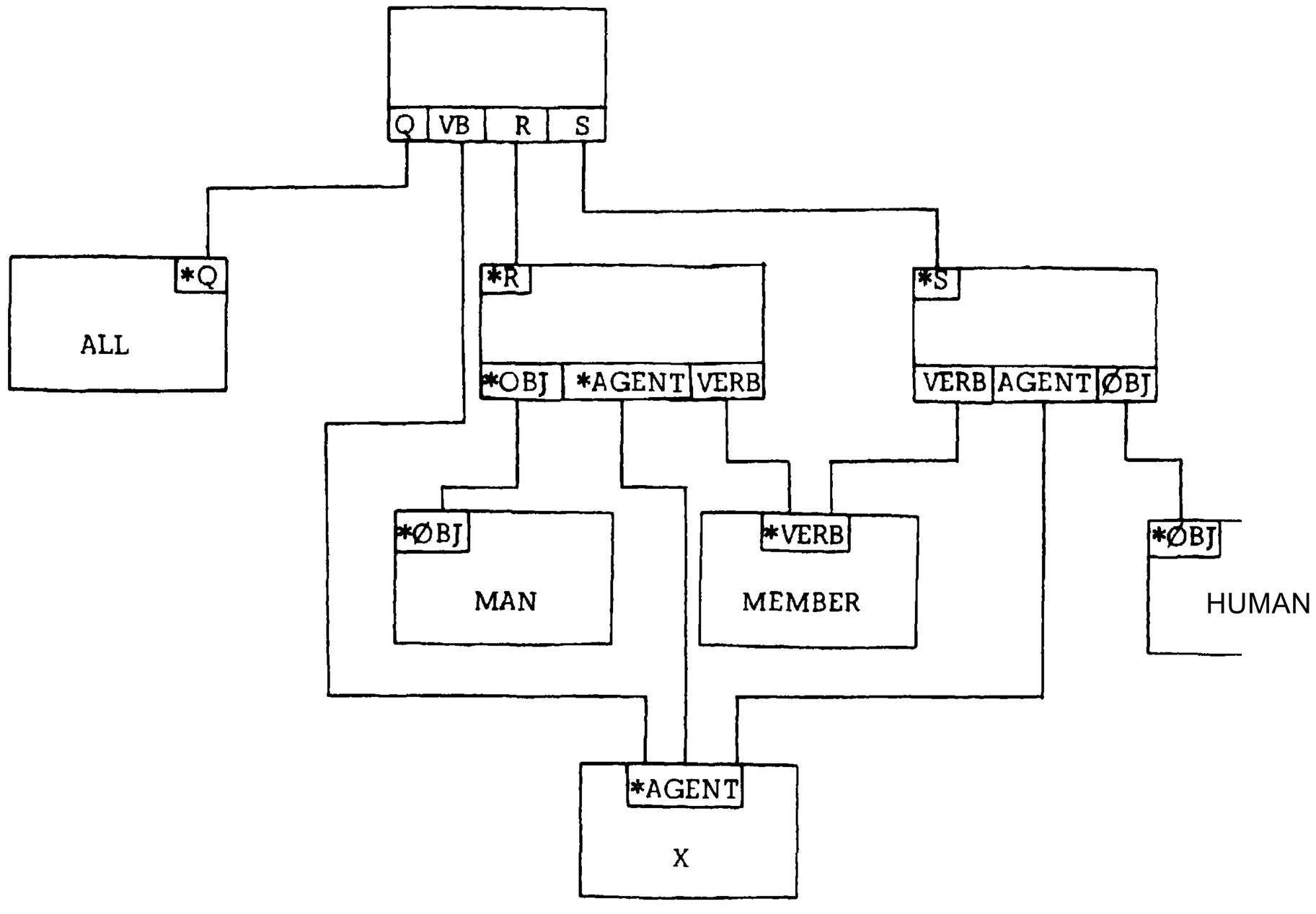

FIGURE 5: A DEDUCTION RULE IN MENS

(i) It may be used for generating consequences .
(ii) It may be confirmed by exhaustive induction, i.e. proved F-true in the universe of the data structure at any given time.
(iii) It may be deduced from other deduction rules.
(iv) It may be refuted by finding a counter-instance in the data structure.
(v) Its negation may be deduced.
(vi) It may be treated as a specific statement, which includes its use as an assumption in the deduction or negation of other rules as in (iii) or (v).

We will be mainly interested in using a deduction rule for generating consequences. In this process, there are two ways of using a restriction. They are:

(1) A possible substitution for the variable may be checked to see if it fulfills the restriction,
(ii) The data structure may be searched to find all items that fulfill the restriction.

The amount of information that may be deduced with any deduction rule depends on more than the quantifiers and the number of items found thatare able to fulfill the restrictions. Italso depends on the structure of the logica 1 connectives in the deduction rule . Tor example , a deduction rule might have a consequent that was the conjunction of several substructures. Thus, several independent substructures might be deduced from each choice of items to substitute for the variables. There are, therefore, several different ways we may use a deduction rule for generating consequences. We may instantiate over all items that satisfy the restrictions or just over those we are interested in. Similarly, we may generate all the consequences Justified by the deduction rule or just those needed to answer a particular question.

In the following sections, we will first discuss how a deduction rule useful for answering a particular question is found, and then discuss how the executive routines interpret the deduction rules and generate consequences.

6. Finding Deduction Rules

A deduction rule is needed when the number of items found to satisfy a fIndspec (see section 3) is less then the minimum number required. The problem then, is to find a deduction rule capable

of generating an item that satisfies the findspec. Say the findspec is

(i) (?0,#,L_1 :(I_{1}. . . . .I_{1m}). . . . .L_n :(I_{n1}.....I_{nm}))

where L_{1i}, ... , L_n are system relations and I_{11}. , . .. ,1_{nm} are specific items (we will assume at first that a substructure only one level deep is required and consider the case of several level structures later). In order for a deduction rule to generate the desired item, it must be headed by a quantification clause that is connected through a path of scope and argument pointers to an item which contains the labels L_1 , ... ,L_n one or more of which point to variable items and the rest of which point to some item in the appropriate list in (i). We can, therefore, locate the deduction rule by searching for any item that satisfies the findspec:

(ii) (?(),#,L_{I}.:((I_{11}.....I_{bIm}.)UV).....L_n:

((I_{n1I} ,.....I_{nmn}) U V))

where V is a list of all variable items in the data structure. For the sake of efficiency, we maintain an item which we shall call VBL. No other item in the structure contains a pointer to VBL, but whenever a variable item contains a pointer L to an item I, VBL also contains a pointer L to the item I. Thus, the findspec (ii) is equivalent to:

Oil) (?0,#,L.:(VBL,I, „. . . . .I.). . . . .L :

(VBL,I_{n1I}.....I_{nmn})).

Note that any item that satisfies (1) must be an instantiation of any item that satisfies (iii) and, further, it is possible to deduce an item that satisfies (i) only if an item satisfying (iii) exists in the data structure.

For each item, I, found satisfying (iii) we may record what substitutions we are interested in for the variables pointed to from I. If I has a pointer L_i to a variable item X_1 , we record that the only items we are interested in submitting for X. are I_{i1} . . , . I_{im} by putting them in a "possible substitution" list for X . They will later be checked against the restriction on X .

For each item I satisfying (iii), we then follow the paths of reverse scope and argument pointers until coming to an item D that is the head of a deduction rule. This will be a deduction rule capable of generating the consequence we are interested in. While following this path a trace list is created. This is a list (S_1 , . .. ,S_k) where S_1 is I, S_k is pointed to by a scope or argument pointer from D, and S_i , $2 < i < k$, is pointed to by a scope or argument pointer from S_{i-1} . The trace list will be used to limit the consequences generated to the ones desired.

In the case of failing to find items matching a findspec involving several levels, the same process is carried out, but we must be sure to allow for all possibilities of variables replacing constants. That is, each level is handled as above for progressively higher levels, and the reverse scope and argument pointers are not followed until the highest level has been done.

7. Generate

The routine to generate consequences from a deduction rule is a recursive procedure that is initially given the internal name of an item that heads a substructure with no free variables. It returns a list of items (internal names) that head substructures representing the consequences that have been generated. These substructures might then either be left in the data structure or be erased. The Generate routine is written to generate consequences according to the author's understanding of the meanings of the quantifiers and logical connectives. It is not designed to prove theorems, but to use deduction rules and other data that have been stored and are assumed to be valid by generating consequences of them.

For example, when generating a consequence headed by a universal quantifier, the restriction is used to direct a search of the data base which results in a list of items that can substitute for the bound variable and then the scope is generated When generating a consequence headed by a connective clause, the action taken depends on the connective. For example with a disjunction, if all but one of the disjuncts can be refuted, the remaining one is generated.

8. Confirm and Refute

The Confirm and Refute routines are used by the Generate routine to determine if a fact represented by a structure is true or false in the data base. It is, of course, possible for an expression to be neither confirmed nor refuted. Tor example, the statement "All men have two arms" would be neither confirmable nor refutable if we knew of

exactly 100 men, of whom 99 had two arms, but we had no information about the hundredth.

The Confirm and Refute routines also use the author's knowledge of the quantifiers and connectives. Tor example a disjunction is confirmed if and only if any argument is confirmed and is refuted if and only if all the arguments are refuted.

9. Substructure Directed Searching

Using a restriction to find all the items that satisfy it and finding an instantiation of a substructure containing free variables in order to confirm or refute it require a process similar to the one used to find an item described by some findspec. Such general substructures may contain some items which are connected to the head item by several different paths. If these items are constant items, any instantiation of the general substructure will contain them at the end of similar paths from the head item. If, however, they are variable items or items heading substructures containing variable items, the instantiation substructures will have different items in their place and we must be sure that no item in the general substructure is substituted for by more than one item in any instantiation substructure. This is done in the say way as evaluating a findspec which contains vnames which originally appeared preceded by "%" or in the findprefix.

10. Summary

In retrospect, we can see several significant facets of the MENS structure and the MENTAL system. First, the work has been developed with a unified viewpoint grounded in the theoretical basis represented by the six motivating factors discussed in section 2. Underlying these have been the desires to maintain complete generality and to keep the executive routines as simple and general as possible. Thus the number of ad hoc features have been kept to a minimum. The only departure from building just a structure and those routines necessary to manipulate the structure ignoring what information might be stored in the structure was the establishment of the system relations and item relations used to store deduction rules and the executive routines to interpret them. Once that was done, however, no further constraints were placed on the deduction rules so that generality was maintained to a large degree.

Another significant facet of MENS is the two levels of relations — system relations and item relations. System relations are the basic organizational mechanism of the structure, yet the user is allowed to define the ones he wants to use and thus may experiment with different semantic structures . Item relations are the conceptual, meaningful relations that hold between other concepts, yet the fact that they are relations is preserved only by the way they are connected in the structure, which is determined and interpreted by the user. Item relations, as conceptual entities, may have stored information about them as well as information using them.

A very important facets of MENS and MENTAL is the ability to enter, retrieve and manipulate deduction rules the same way specific facts are entered, retrieved and manipulated, yet deduction rules are used by the system to deduce information that was not previously explicitly stored in the structure. Thus one may explain to the system what a concept means by giving, in general terms, the implications of the concept, and one may give this explanation Just like he gives the system any other information.

The system and structure as presented in this paper provide an environment in which important problems in question-answering and computer understanding may productively be investigated. Also MENS and MENTAL may be used as an experimental vehicle for further research in semantic structures.

References

(I) Bach, E. Nouns and noun phrases . Universals in Linguistic Theory, Bach, E. and Harms, R. T. (Eds.), Holt, Rinehart and Winston, New York, 1968, 90-122.

(2) Carnap, R. Empiricism, semantics, and ontology. in[I0], 205-221. Originally in Revue Intern, de Phil. 4 (1950) 20-40.

(3) Elliott, R. W. A model for a fact retrieval system, unpublished Ph. D. dissertation, University of Texas, Austin, Texas, 1965.

(4) Fillmore, C. J. The case for case. Universals in Linguistic Theory, Bach, E. and Harms, R. T. (Eds.), Holt, Rinehart and Winston, New York, New York, 1968, 1-88.

(5)_____Lexical entries for verbs. Foundations of Language, 4, 4(Nov. 1968), 373-393.

(6) Green, C. C, Raphael, B. Research on intelligent question-answering systems . AFCRL-67-0370, Stanford Research Institute, Menlo Park, Calif., May, 1967.

(7) Kaplan, R. M. The MIND system: a grammar-rule language. RM-6265/1-PR, The RAND Corporation, Santa Monica, Calif., April, 1970.

(8) Kay, M. The MIND system: a powerful parser, (forthcoming).

(9)_____, Martins, G. R. The MIND system: the morphological-analysis program. RM-6265/2-PR, The RAND Corp., Santa Monica, Calif., April, 1970.

(10)_____, Su, S. Y. W. The MIND system: the structure of the semantic file. RM-6265/3-PR, The RAND Corp. , Santa Monica, Calif., June, 1970.

(11) Quillian, M. R. Semantic memory. Semantic Information Processing, Minsky, M. (Ed.), MIT Press, Cambridge, Mass., 1968, 227-270.

(12)_____The teachable language comprehender: a simulation program and theory of language. Comm. ACM 12, 8 (Aug. , 1969), 459-476.

(13) Quine, W. V. O. Notes on existence and necessity. J. Phil. 40 (1943) 113-127.

(14) Schwarcz, R. M , Burger, J. F. , Simmons, R. F. A deductive question-answerer for natural language inference. Comm. ACM 13, 3 (March, 1970), 167-183.

(15) Shapiro, S. C. The list set generator: a construct for evaluating set expressions. Comm. ACM 13, 12 (Dec. , 1970), 741-744.

(16)_____A data structure for semantic information, processing. Unpublished Ph.D. dissertation, University of Wisconsin, Madison, Wisconsin, 1971.

(17)_____and Woodmansee, G. H. A net structure based relational question answerer: description and examples. Proc. Int. Tt. Conf. Art. Intel., Walker, D. E. and Norton, L. M. (Eds.), Washington, D. C, 1969, 325-345.

(18) Shapiro, S. C, Woodmansee, G. H., Krueger, M. W. A semantic associational memory net that learns and answers questions (SAMENLAQ). Technical Report #8, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, Jan., 1968.

(19) Simmons, R. F., Burger, J. F. A semantic analyzer for English sentences. SP-2 987, System Development Corporation, Santa Monica, Calif., Jan., 1968.

(20)_____,_____, Schwarcz, R. M. A computational model of verbal understanding. SP-3132, System Development Corporation, Santa Monica, Calif., April, 1968.

(21) Woods, W. A. Semantics for a question-answering system. Mathematical Linguistics and Automatic Translation Report No. NSF-19 to the National Science Foundation, The Aiken Computation Laboratory, Harvard University, Cambridge, Mass., September, 1967.