BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

AFCRL-69-0438 6 October 1969

CAPTURING CONCEPTS IN A SEMANTIC NET

by

Anthony Bell

M. Ross Quillian

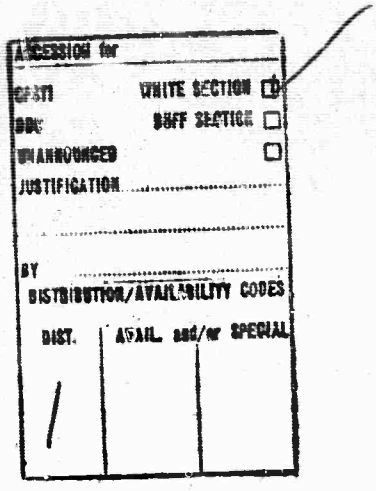
Scientific Report No. 13
Contract No. F19628-68-C-0125
Project No. 8668
Contract Monitor: Hans Zschirnt
Data Sciences Laboratory

Prepared for:

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
Office of Aerospace Research
United States Air Force
Bedford, Massachusetts 01730

This research was supported by the Advanced Research Projects Agency under ARPA Order No. 627.

54



Qualified requestors may obtain additional copies from the Defense Documentation Center. All others should apply to the Clearinghouse for Federal Scientific and Technical information.

AFCRL-69-0438

6 October 1969

CAPTURING CONCEPTS IN A SEMANTIC NET

by

Anthony Bell

M. Ross Quillian

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Scientific Report No. 13
Contract No. F19628-68-C-0125
Project No. 8668
Contract Monitor: Hans Zschirnt, Data Sciences Laboratory

Prepared for:

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
Office of Aerospace Research
United States Air Force
Bedford, Massachusetts 01730

This research was supported by the Advanced Research Projects Agency under ARPA Order No. 627.

ABSTRACT

A working memory model based on a semantic network is described in detail. Some advantages and disadvantages of such a model are discussed. An attempt is made to enable a reader to learn to perform the formidable task of representing data in the memory format. Since the actual memory is not easily read (or written), a set of LISP programs are included which make these tasks manageable.

TABLE OF CONTENTS

Report No. 1885 Bolt Beranek and Newman Inc.

CAPTURING CONCEPTS IN A SEMANTIC NET

Part I: Rationale and Overview of a Model

This paper describes a memory model, i.e., a particular format and organization for the information comprising a data store. We propose this model both as a useful way of actually storing any large body of factual information in a computer, and as a theory of the general structure of long term human memory. The model, in various earlier versions, has been used in computer programs (Quillian, 1966, 1969) and as the basis of psychological experiments (Collins and Quillian, 1968, 1969). The aim of this paper is to explain the model in considerable tedious detail, so as to make it possible for anyone either to build and use his own version in a computer program, or to generate detailed predictions for psychological experimentation. The paper thus is essentially a sort of primer on how to translate written text into this memory format.

The memory is intended to allow representation of any concept, but in a way which only deals with one facet of human memory. That is, the memory is designed to encode only descriptive information, it omits any explicit reference either to emotional meaning or to plans for action. Such plans or routines may be for muscular, for perceptual, or for cognitive action, and may be routines designed to work on other routines. We suspect that emotional meaning can eventually be handled by simply adding some sort of tags to descriptive information, and hence can be omitted for the present without terrible danger of constructing the entire memory incorrectly. Our omission of routines, however, is much more worrisome. We of course don't know the degree to which such routines and information related to them form a part of memory for descriptive information,

but our guess is, they do, to a very large degree. We suspect that things which are the functional equivalents of the names of routines, of the names of input parameters of routines, and of the names of various effects of routines form the "primitives" of descriptive memory. We believe that Piaget's "schemata" are best understood as such information, plus the routines themselves. It may in fact be that all descriptive information is initially constructed out of these primitives; Piaget has argued persuasively that only on the basis of an infant's developing plans does it become possible for him to conceive of such notions as enduring objects, as time, as space, etc. (Piaget 1950, see also Quillian, Wortman, and Baylor, 1965.) Within the descriptive material itself, there need be no primitives, everything can simply be defined in terms of pointers to other things, analogous to the way words are defined in a dictionary. It is the links leading out of this descriptive material, to action, to recognition, and to cognition that our present model omits, and whose omission worries us. For this means that we are skipping over all the underpinning on which we suspect human memories actually stand; we are attempting to model an advanced result of the human development process in abstraction from what we suspect is its schemata-related base.

What can justify such an approach? Well, essentially it is only that one must attack problems where he can; we think we have in fact been able to find out some things about the organization of memory at this high level, both by computer simulation and by psychological studies, and that a great deal more can be discovered. No one really knows how to write the routines people use to perform even their simplest actions or perceptions. (In fact, so far, more is probably known about how people perform high-level cognitive actions. See, e.g., Simon, 1969). Even less is known about how to build a machine capable of developing routines for muscular or perceptual actions, the way that people do. A few "robot projects"

(MIT, Stanford, SRI) are now making beginning efforts in this direction, but results that will enable us to deal with things like human language seem a very long way off. It has seemed to us that in this situation, it is worthwhile to try directly to work out a format and an organization for descriptive memory. This format optimally would be rich enough to encode the meaning of any natural language text, in a format that yet is uniform enough to be managed by specifiable procedures (debuggable computer routines), and at the same time not become impossibly cumbersome or redundant. These conflicting goals are not easily reconciled, and we will try to point out remaining problems in the code as we proceed.

Neither syntactic parsing schemes nor symbolic logics are of much help in modeling such memories, since they either deal with only one small aspect of language, or they become impossibly cumbersome, or both. It also seems to us that "ontological reality" is of very little concern for a model of human memory. The memory structure here thus is completely phenomenalistic, it attempts to represent concepts, period. Any concern about the relationship of these concepts to "the real world" seems to us beside the point. (Cf., for example, works such as Quine, 1960.)

Thus this paper is directed toward anyone who may be interested in the pragmatic details and problems of how to simulate a memory for descriptive information, to the degree that this may be possible without explicit incorporation of schemata. Our hope is of course that such a memory can have routines and schemata-like links added to it in the future.[1]

---

[1] Some work toward a memory that is both descriptive and imperative has been done at Carnegie-Mellon University. (Allen Newell, personal communication.)

The memory will be described here as it is now expressed in LISP, but this is more a convenience than a necessity, it could also be set up and handled by other list processing languages.

Overall Memory Organization

Essentially, the memory is a mass of interconnected nodes which represent conceptual elements. In general each of these nodes is itself made up of a constellation of pointers to other nodes, so the overall memory is a general graph structure, with no restriction against loops or reentries. (We will not be concerned with the mathematics of such structures.) Many nodes in this network will contain pointers to the same other node. In fact, all the nodes which use a particular concept as a compositional ingredient should contain a pointer to the same node, so that no more than one node will ever be required in the entire memory to represent explicitly any particular concept. If two nodes use the same concept but with different modifications of it, then each of them will point to separate intermediate nodes, which in turn will point to the node representing the common concept. This kind of memory organization removes redundancy, while permitting one concept to be defined in terms of others.

Such a memory organization also permits common ingredients present among any given set of concepts to be located swiftly, by a technique which effectively simulates a parallel search. This method will be recognized as that used in prior programs (Quillian, 1966, 1969). It is based on the fact that, starting from any given node in the memory, a program can easily trace to all the nodes that this node contains pointers to, and then (on a second pass) to all the nodes these nodes contain pointers to, and so on, for as many such passes as is desired. In a rich memory this breadth-first tracing will tend to fan out on each successive pass

to a greater and greater number of nodes, although certain branches of the fan will either circle back to previous nodes, or will simply die off due to reaching some NIL node, i.e., one whose meaning has not yet been specified in the memory.

Next, suppose that as a routine proceeds with such a trace, it places an "activation tag" on every node it passes through. This activation tag names the initial starting concept which led (however indirectly) to all the nodes reached and tagged.

Now, suppose that this process is initially given more than one initial starting node. Its tracing now proceeds breadth-first through all these concepts at once, moving one level deeper into each of them on each pass. Thus it simultaneously traces out a separate "fan" for each initially given unit. The processor places an activation tag on each node it reaches, identifying the particular fan it is part of by naming the initial node at the fan's head. Moreover, this process now checks every node it tags, to see if the node has already been reached during prior tracing eminating from some other initial node. This is easily determined, since any such node will have a tag showing it has already been reached, indicating its initial node(s). Whenever such a previously tagged node is found, it constitutes an ingredient common to these two initial nodes, an "intersection."

This method of locating common ingredients of concepts will, in general, find the common ingredients which are closest to the initial starting nodes before it finds those which are further away. That is, it will locate intersections reachable by short paths from the initial concepts before it finds those reachable only by longer paths. Some restriction on the number of passes to make before quitting must always be given to such a routine, whether or not the process is also terminated after some given

number of intersections have been located. Breadth-first searches to find intersections of concepts are used in a number of ways within a program such as TLC, with more elaborate tags which allow the program to distinguish an intersection node that is connected to an initial node by a path going only through supersets from one whose path at some point moves "out" through other information associated with some node(s). (Supersets are explained below.)

The most important property of a memory in which everything points to other things is that a large part of its information is implicit (Quillian, 1966). Retrieving its information therefore requires not only retrieving material that is stored in the memory, but also generating new material on the basis of retrieved explicit material. That is, such a processor must be able to take a piece of explicit material from the memory, trace to further information stored with the components used to compose that material, and produce new information by projecting implications of the explicit information onto the information stored with its components. For instance, if the memory explicitly stores the fact that John employs Bill, routines should not only be able to retrieve this, but also to derive that John probably pays Bill, that Bill probably does something John wants done, that John is somewhat likely to hold a position of more power than Bill, etc. The routines to do this must operate by combining general information stored with "employ" with the specific piece of information that John employs Bill.

Not a great deal is known about how to write such generative retrieval routines, but the reader should be aware that the memory model here is intended to facilitate such use of its information. The trade-off between what to represent explicitly and what to leave to be derived is a choice the coder will sometimes have to make.

Coding vs. Comprehension

In order to simplify our explanation of how to encode natural language into the memory, we will concentrate on text which is noun phrases and usually dictionary definitions. This is not essential, any kind of text can be encoded. But, such text does eliminate several difficult or messy problems. The first of these is deciding what is important enough to encode; in dictionary definitions everything can be assumed to be, while in other text relatively little is. Second, in dictionary definitions we don't have to decide where to store encoded information; it is clearly to be stored with the word being defined, and there only. In other text, this is a very big problem. (Quillian, 1969). Third, in dictionary definitions tense is usually irrelevant, and hence need not be coded. Fourth, in dictionary definitions there is rarely any need to be concerned with who believes a given concept or assertion that is coded in the memory, or with his confidence in its validity. Such information clearly is part of our more general knowledge, and must be added to the encoding of much text.

In this paper we will take two other important shortcuts strictly for convenience in teaching the code. The first is to code all English as close as possible to the English itself. For example, we here will code a phrase like "lawyer of the client" simply as:

(lawyer(of client))

In contrast, our computer program simulating language comprehension, TLC, comprehends and encodes "lawyer of the client" as meaning that:

"this lawyer is representing or advising this client in a legal matter" (Quillian, 1969)

The information here added by TLC's interpretation is of the sort we believe people actually add in comprehending language. We believe they, like TLC, do this by relating text they read to many parts of their entire store of knowledge of the world. Doing this greatly enriches their understanding of such text. However, it also moves an individual's comprehension toward this own memory, built up over his entire past history, and hence reflects his personal, idiosyncratic view of the world. By its very individuality, then, such interpretative coding is ineffective for one person to use in teaching the format to another. Thus we here will encode text as similarly as possible to the text itself, even though this is not what we believe people actually do in comprehending language.

The second shortcut we will take here is to pretend that all words only have one meaning. This is especially false for prepositions, and anyone who really wants to build a memory, or to conduct psychological experiments based on one will probably find he must look up every word he encodes, decide which meaning of it he wishes to refer to, and then do so. The method of referring to a particular meaning of a word is described in the last section of this paper. However, all the principles of the model can be illustrated without this arduous labor, so we will omit it.

Internal Memory is Not Printable

Although a memory with circles and reentries can easily be set up within a computer, it cannot be directly written out in linear form. Therefore, there are actually three forms of the memory format. One of these is the "internal" form in which information is stored within a computer or hypothetical human memory, another is a linear form which can be printed out to illustrate any segment of that internal memory, still another is a slightly different linear form which a person can write, and which then can be translated into the internal form by a small computer program. We will call these three formats respectively the internal

or actual memory format, the output readable format, and the input readable format. The actual memory "model" is of course the internal form.

Figure 1 illustrates a piece of information encoded in four forms. Fig. 1A is the English. Fig. 1B is in the input readable form that a coder might create to represent this English. Fig. 1C is a picture illustrating the piece of actual internal memory that the input translation program would create to represent 1B. Finally, Fig. 1D shows the output readable form that our output translation program would produce if asked to translate 1C out into readable form. The main difference between these coded forms is that whereas words represent concepts in the readable forms, lists represent these concepts in the internal form. (In general, one of these lists represents the first definition of the corresponding word that appears in the readable form.) Thus, data in the input readable form must be translated in order to produce sections of internal memory. We will discuss this translation more thoroughly below, after the input format of the memory is clear. Listings of LISP routines for translation in and out of internal memory are given as appendix 2.

The most important principle of all the memory formats is that all factual information is encoded as either a "unit" or as a "property." A unit represents the memory's concept of some object, event, idea, assertion, etc. Thus a unit is used to represent any of the kinds of thing which can be represented in English by a single word, a noun phrase, a paragraph, or some longer body of text. A property on the other hand encodes any sort of predication, such as might be stated in English by a verb phrase, a relative clause, or by any sort of adjectival or adverbial modifier.

FIGURE 1: Information Coded in 4 Forms

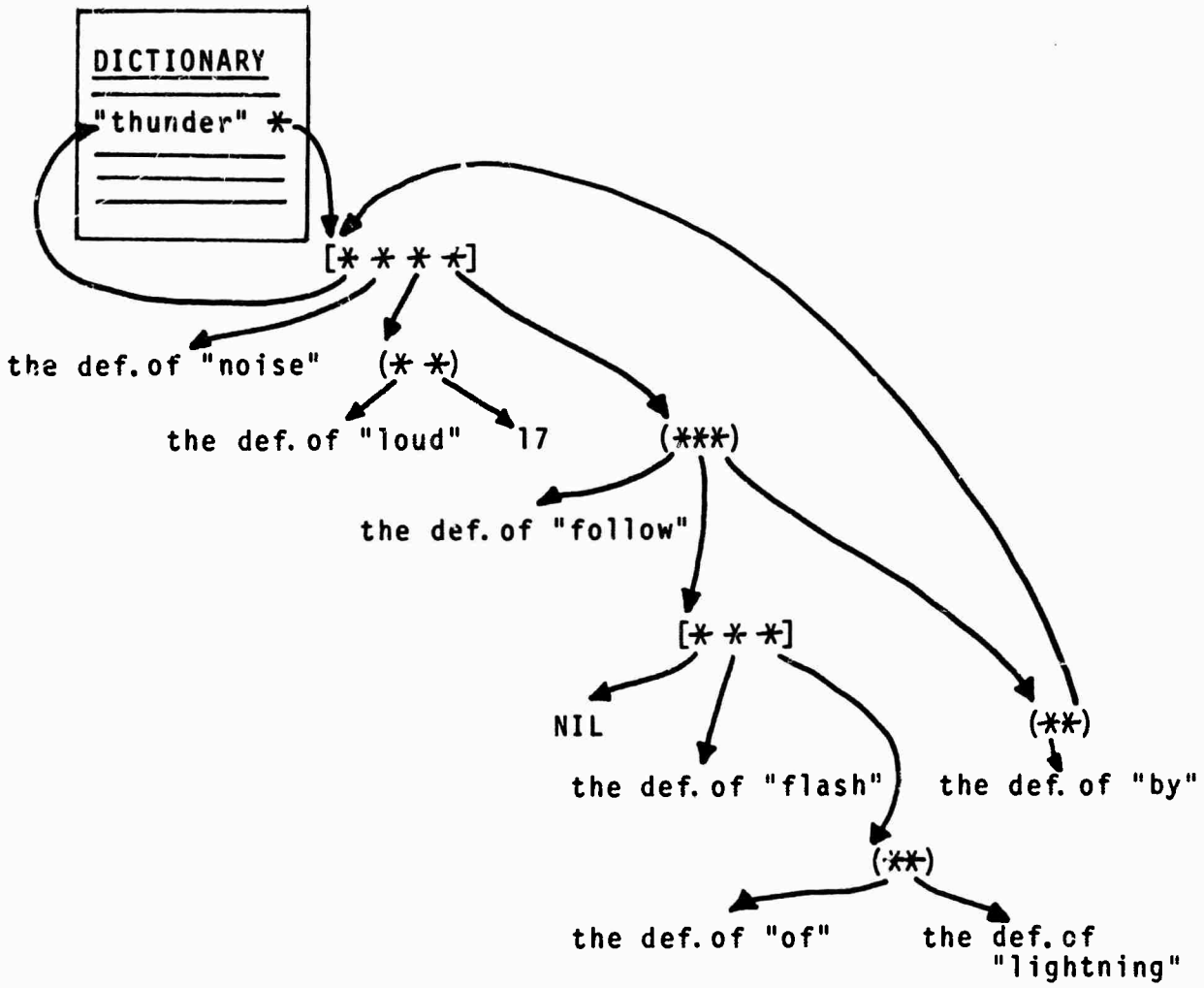
1A: ENGLISH

"Thunder is a very loud noise following a flash of lightning."

1B: INPUT READABLE FORMAT

THUNDER:
(NOISE (LOUD 17) (FOLLOW (FLASH (OF LIGHTNING)) (BY THUNDER)))

1C: INTERNAL MEMORY FORMAT



1D: OUTPUT READABLE FORMAT

THUNDER:
(NOISE (LOUD 17) (FOLLOW (FLASH (OF LIGHTNING)) (BY (*THIS* THUNDER))))

In Fig. 1C the word "thunder" —— which itself is outside the memory in a "dictionary" —— is associated with one pointer to a unit in the memory. This unit is shown as delimited by two square brackets. Also shown in Fig. 1C is another unit in brackets, and four properties, each of the properties being delimited by a set of parentheses. Distinguishing units from properties by the use of brackets vs. parentheses is only an aid to our description, there is no corresponding distinction in the actual memory format, and, as in Figures 1B and 1D, both units and properties actually are delimited with parentheses.

In the following description we will always be referring to the input readable form, unless otherwise stated. The reader may wish to refer frequently to the schematic drawing of this format in appendix 1. A generative description of the syntax of the input format, plus a schematic of the internal memory format, are also included in appendix 1.

Units

Any unit's first element (reading left to right) must always be a pointer to some other unit, referred to as the unit's "superset." A unit's superset will in general represent some more generic concept than the unit itself represents. Thus the superset of a unit JOE-SMITH might be MAN, that of MAN might be PERSON, that of PERSON might be ANIMAL, etc. (Any of these could also be the LISP atom NIL, used throughout to represent a lack of further information.) After its first element, a unit can contain either nothing or any number of pointers, but each of these must be to a property, not to a unit. Thus, Fig. 1 shows the superset of the unit representing "thunder" to be NOISE, followed by two pointers to properties.

Each property pointed to in a unit represents some assertion that is somehow associated with that unit. When all these properties are simultaneously associated with the unit's superset, the resultant is the concept the unit represents. In other words, a concept is always represented in our format by a list of pointers. The first points to the concept's superset, of which the concept can be considered a special instance, and the rest point to properties which together state how that superset must be modified and related to other units in order to constitute the concept intended. Properties are therefore the means by which refining modifications or changes of state are encoded. The relationship implicit between the properties of a unit is conjunction.

Note that new units can be freely constructed by creating an empty unit and using a pointer to some prior unit as the new unit's superset. Thus, suppose one wished to construct a new unit to represent Joe Smith as a boy, or one to represent Joe Smith from the point of view of his wife, or one to represent Joe Smith when angry. Each of these could be constructed as a new unit having as superset a pointer to the previous JOE-SMITH unit, followed by whatever refining properties were necessary to compose the appropriate particular concept. Suppose, further, that after creating these three new units, one wished to construct a unit representing Joe Smith at age eleven, and that one wished this to include all the information stored with the unit representing Joe Smith as a boy. This is done by simply creating another unit, using as its superset a pointer to the JOE-SMITH-AS-A-BOY unit, and then attaching further refining properties to this newest unit. This kind of free creation of new units which "include" old units is a basic step in building up of new structures to represent new material.

Properties

A property is always an attribute-value pair (and may also have subproperties.) However, the notion of an attribute-value pair is used in separate ways. First, words and phrases serving as adjectives or verb modifiers are encoded by using the adjectival or adverbial as the property's attribute and using a numeral indicating the degree, amount, or number of that adverbial as the property's value. (For example, the adjective "white" would be encoded as the property (WHITE 15), while the phrase "very white," would be encoded as the property (WHITE 17). In Fig. 1, (LOUD 17) is a property of this type. (The meaning of numerals will be explained below). For convenience we will refer to all such properties as "adjectival" properties. Second, any preposition and its object, and any verb and its (direct) object is also encoded as a property, but in this case the preposition or verb is used as the property's attribute, and the word or phrase that would normally be that word's grammatical object is used as the property's value. Thus, a property such as (ON HILL) can be encoded, as can the property in Fig. 1, (FOLLOW (FLASH ...)...). We will refer to these respectively as "prepositional" and as "verbal" properties. In all cases the notion of an attribute-value pair is the core of any property. This fact introduces an important uniformity into the data structure, without having either to give up expressive power or to introduce needless redundancy into the data.[1]

[1] until recently, we would have encoded the adjective "white" as the property (COLOR WHITE). This is redundant, since the dimension COLOR should be available in memory anyway as a superset of the concept WHITE. However, the attribute in a property such as (ON HILL) or (FOLLOW (FLASH...) ...) ...) is not similarly redundant, and the redundancy in adjectival properties seemed justified by the uniformity that this produced in the format. The key to removing this redundancy was clear once it occurred to us that it is only adjectives and verb modifiers which one in general wished to qualify——one often wishes to say "very white" or "slightly white," but one will very rarely wish to say "very much on a hill," or "slightly follow a flash." Thus, it is natural to encode the amount, degree or number of all properties representing adjectives and verb modifiers, but not to do so for prepositional or verbal properties. The second kind of property, one whose attribute is a verb or a preposition, may also have additional qualifying information showing its amount, degree or number, but then this must be encoded as a modification of either the property's attribute or of its value, or with a special property having N as its attribute, as will be explained below.

To summarize, a unit has one obligatory element, its superset, and a property has two, its attribute and its value. All of these are represented by pointers, each of which must point to some other unit. In both units and properties the obligatory element(s) must come first, and may be followed by any number of pointers to other properties, which supply the modification necessary to refine the unit or the property adequately.

Since there is no limit on the number or nesting of properties which can be associated either with any unit or with any property, concepts and predicates of unlimited complexity can be represented in the memory format.

The Only Way to Tell a Unit from a Property is by Location

In the internal memory, both units and properties are normally represented by a list, or by the atom NIL, indicating a lack of further information. Thus there is in general no way to tell, by looking at an arbitrary piece of data in the memory, whether it is a unit or a property. So, whenever any routine follows some pointer into the memory, it is absolutely essential that it know whether that pointer leads to a unit, or to a property, and that all further processing from that point keep track of which of these it is dealing with. It is possible to do this without ambiguity, since the syntax of the format is rigorously defined on this point. Namely, supersets, attributes, and values must always be units while these obligatory elements are followed optionally by any number of pointers to other properties, but only to properties. Like the properties helping to comprise a unit, additional properties of a property represent refinements, in this case refinements of the assertion stated by the property's attribute-value pair. By using such "sub-properties" a property's meaning is refined or modified as necessary.

The basic format is illustrated in Fig. 1B: First, the unit representing the memory's concept of "thunder" has NOISE as its superset, and two refining properties. The attribute and the value of the second property assert simply that some flash of lightning is followed. (The value being the unit with FLASH as superset and (OF LIGHTNING) as modifying property). However, a refining sub-property of this property then further specifies that this following of a flash is done by the thunder itself. (The value of the attribute BY is a pointer back to the unit representing the concept THUNDER). Note that this pointer is to the whole THUNDER unit, not to its superset NOISE. In total, then, Fig. 1B simply represents a concept to the effect that thunder is a very loud noise which follows a flash of lightning.

To further extend the memory's expressive power, it is necessary to be able to encode quantifier-like modifications of units. This will be described below, for the moment let us only note that such quantifying information is omitted from any unit representing a singular thing.

Part II: How to Encode

The foregoing concludes our relatively general statements about the memory, from here on we will move into details, and will assume that a reader is really interested in developing an ability to encode text into the format.

At this point the reader should be able to code some simple phrases and sentences. To check himself, it would be good to code, say: "A boy on a hill." Since overall this is a noun phrase, the piece of data we want to build to represent it will be a unit. The first element of this unit will be its superset, so we first locate that word or phrase which will serve as superset of the unit. This superset will be the phrase's syntactic head, which might be defined as the answer to the question, what is the piece of text talking about? Since "a boy on a hill," obviously is talking about "a boy," and since the singular quantifier "a" can be omitted, we put "boy" in the superset position, producing:

(boy...

Being on a hill is clearly a property of this boy, so we will represent this prepositional phrase as a property. Referring to the first schematic drawing in appendix 1 we see that a property has three kinds of elements: its attribute, its value, and its subproperties. Only the attribute and value are required; the subproperties are optional elements which may or may not appear.

The attribute is usually either a quality, a preposition, or a verb. Here we use the preposition "on". The value is then the object of this preposition, "a hill," and we can again drop the singular qualifier "a". The data encoded thus is simply:

(boy (on hill))

Suppose the input phrase had been, "The happy boy on the hill." There is now one additional property. Since properties may be listed in sequence (without regard to order, incidentally), the

simplest way to represent this data is by adding one property to (boy (on hill)). In this new property however the adjective "happy" will be coded with HAPPY as the attribute, and some numeral as its value. It is therefore necessary to understand numerals.

Numerals

In the first place, numerals will appear at two places in the format. One has already been described; numerals appear as the value of adjectival properties:

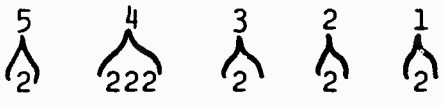
(man (tall 17)) = a very tall man

A second place numerals appear is as the value of the special attribute #. A property with # as attribute must have a numeral as value. Such a property is the way of quantifying a unit. For example:

(foot (# 1002000)) = two feet

(sugar (# 16)) = a large amount of sugar

Thus the use of properties with # as attribute allows units to be created which represent concepts of individual things, of some number of a thing, of substances, etc. This parallels the use of numerals to quantify properties, and in fact a numeral per se has the same interpretation whether it is used as the value of an adjectival property or of a # property.

Every numeral must be a seven digit octal number, where leading zeros may be omitted. This number is considered broken up into five fields, as follows: field: 

Field 1 contains a number which represents the subjectively judged degree or amount of some unit or property. This number has a range from 1 to 7. A 4 is an indication of neutrality, while

5, 6, and 7 indicate a degree or amount judged progressively more positively, while 3, 2, and 1 indicate progressively more negative judgements. However, the meaning of this number must be taken in conjunction with that in field 2, since field 2 is a cue stating how the number in field 1 is to be interpreted. The range of this cue is 0, 1, 2, 3. Respectively, these indicate that the number in field 1 is irrelevant, that it represents an absolute judgement, that it represents a minimum judgement, and that it represents a maximum judgement. Thus, fields 1 and 2 together can refer either to a judged degree or amount, to the lower bound of a judged degree or amount, or to the upper bound of a judged degree or amount. If field 1 is zero then the judged degree or amount is either irrelevant or unknown.

Some examples follow:

(psycholinguist (silly 16)) = the fairly silly psycholinguist
(book(interesting 31)) = the book which is far from interesting
(guerrilla(friendly 25)) = the guerrilla who is friendlier than not

Field 3 is used to encode criterality or frequency. A number with a value in this field provides information about the judged likelyhood that this property or unit is as the coding states. Its range is from 0-7, and the numbers have the same interpretation as the field 1 number, with 4 again serving as neutral point. Thus:

(psychologist (dull 516)) = a psychologist who is often dull.

"Often," "never," "hardly," "not at all," are all phrases which usually describes criteriality; so these words and phrases will not themselves become units or properties during encoding, but instead will be represented by an appropriate value in field 3 of a numeral attached to the units they modify. Thus:

(telephone (black 617)) = a telephone that is usually black

Field 4 is reserved for real numbers, with field 5 as its cue. Field 5 always has a value of 1, 2, or 3 if a number is indicated in field 4, a 0 again indicating irrelevance. Respectively, these

values indicate that field 4 contains: an absolute number, a numerical lower bound, and a numerical upper bound. The number itself may take up as much of field 4 as necessary and thus may be as high as 777Q. Some examples follow:

centipede(leg (# 2067000))) = the centipede has more than 67Q legs

(army(kill (infant(communist 17) (# 1067000)) (by (*this* army))))=
the army which kills exactly sixty-seven communisit infants

It should be apparent that various indicators may be used in combination in the same numeral:

(semantics (good 217)) - semantics which is rarely very good

(spider (leg (# 1008700))) - A spider which always has 8 legs

By this time it should be apparent how to encode "the happy boy on the hill":

(BOY (HAPPY 16) (ON HILL))

Sets

It is also essential to be able to represent a set of diverse units or properties, aggregated in some particular way.

A set is indicated by a list the first element of which is either AND, EOR, AOR, or SEQ, followed by the members of the set. The initial marker indicates the relationship between the members of the set that is pertinant in it. The markers indicate relationships as follows: AND = and, EOR = exclusive-or, AOR = and/or, SEQ = sequence, which is indeterminate as to being temporal, spatial, or both. These are all we have needed so far, but others may of course prove convenient in the future. Sets provide examples such as:

(AND MAN WOMAN) = The man and the woman

(GIRL ((AOR LIKE KNOW) JOHN)) = The girl who likes and/or knows John.

((OR GIRL BOY) (ON STREET)) = a boy or a girl on a street

(SEQ (FLASH (BRIGHT 16)) (NOISE (LOUD 18))) = A bright flash.
A loud noise.

(GIRL (EOR (IN CAR) (NEAR TRUCK))) = a girl who is either in a car or near a truck

Readers unfamiliar with LISP may have some difficulty becoming accustomed to using complex elements as single items in the data structure. That is, an attribute or a value must always occupy only a single space by being enclosed in parentheses. Thus in the second example above a set is used as an attribute, in the third a set is used as a superset, in the fourth modified objects serve as the elements of a set, and in the fifth there is a set of properties. (Since the relationship assumed between a string of properties is AND, one will not normally form an AND'ed set of properties.)

Nested Properties

Let us move on to another example. Consider: "The suit which is dark grey." To begin with, this is like our earlier examples, the superset is obviously a suit, and it has what seems intuitively like one property, that it is dark grey. However, this property itself is a modified one; thus we must create a slightly more embedded structure:

(SUIT (GREY 16 (DARK 14)))

Here there is one property of SUIT which itself has one sub-property. It is important to note that all sub-properties modify the attribute-value pair of their parent property. Thus it is important to distinguish a case like the last one from:

(SUIT(ON (MAN (DARK 14)))) = a suit on a dark man

or

(SUIT ((NEAR (N 17))CLOSET = a suit very near a closet

In these two cases the nested properties are not subproperties of the property, since they modify in the first case its value, MAN,

and in the second case its attribute, NEAR, rather than the whole attribute-value pair, as in the prior example.

Verbal Properties

As previously stated, adjectives, prepositions and verbs may all be used as attributes. Our examples so far have used adjectives and prepositions. When verbs are used as attributes the matter becomes a little less obvious.

For verbal properties, as for prepositional ones, the value of the property is always the logical direct object of the verb. (The logical or "deep structure" direct object is what would be the direct object if the sentence were rewritten into simple sentences. Thus it may be opposed to the "surface structure" object in sentences which linguists would describe as having undergone transformations. See e.g. Chomsky, 1966. In the case of intransitive verbs, or of transitive verbs for which there is no direct object present, the atom "NIL" must be placed in the value position. This is imperative and must not be left out, since properties must have both an attribute and a value, and both of these must be units.

Within verbal properties, modifying properties are used to encode adverbs such as "carefully." However, modifying properties are also used in verbal properties to encode indirect objects, subjects, and other things similarly related to the attribute of a verbal property. We do not assume anything about a verbal property because of what it modifies. Thus:

"The man who carefully watches the prisoner." =

(MAN (WATCH PRISONER (CAREFULLY 16) (BY *)))

The "BY" property might seem redundant, but, if we try to omit it by assuming that all verbal properties have the object they modify

as subject, then there is a problem in coding:

"the man the prisoner watches"

One could get around this by introducing inverse relations so that: "the man the prisoner watches" =

(MAN (WATCH-1 PRISONER))

However, there are several arguments against this. First, one must then store the inverse of every verb that has one, and all programs that interpret information in the memory must continuously check for such information. Second, the use of inverse relations makes it impossible to simply choose any property stored anywhere in the memory and add a pointer to it to some other unit. This is an important ability in learning or in a program like TLC. Third, as Fillmore has pointed out, all the subjects of verbs are not related in the same way to the verb —— consider:

"The man who opens the door with a key."

and

"The key that opened the door."

Therefore, it has seemed best not to assume that verbal properties are related in any particular way to something they modify, and to encode all such relationships explicitly. By doing this we remove any need for inverse relationships, and avoid the above three problems, but at the expense of having to explicitly state the subject of every verbal property, if this is known. Thus:

The man the prisoner watches = (MAN (WATCH * (BY PRISONER)))

The man who opens the door with a key =

(MAN (OPEN DOOR (BY *) (WITH KEY)))

All that is wrong with these codings is that they should use different senses of BY, to differentiate an instrumental from an agentive subject. We can only omit the subject of a verbal property

if this is not known, as in:

(MAN (SEE (FRIEND (OF *))) = the man whose friend is seen

Now let us consider a case where we might want to use a property in an illegal position, that is, where a unit is required. Suppose we want to code, "Nixon is a tricky man who goes to Washington to form an administration." The coding of this is straightforward as far as:

```
(MAN (TRICKY 16)
     (GO NIL (TO WASHINGTON)
     (BY NIXON)
     (TO ?
```

At the question-mark we want to code,

"form an administration"

This clearly should be coded:

(FORM ADMINISTRATION (BY NIXON))

However, it is illegal to put this piece of coding, a property, in place of the question-mark above, because to do so would be to use a property as the value of a property. The only solution we have been able to devise so far for this is to introduce a special dummy superset unit, *TO*. With this we form a unit:

(*TO* (FORM ADMINISTRATION (by NIXON))

This unit can be substituted for the question-mark, to complete the above coding. The use of *TO* is inelegant, and a better solution may be available. However, until one is found the inelegance of introducing "TO*'s is an argument against our whole strategy of only distinguishing units from properties by their location, since it is this that prevents use of a property where a unit would normally be expected.

Translation to Internal Memory Format

Material in input readable format is translated into material in the internal memory by a routine, RD (Read Data). A listing of this LISP routine and its sub-functions is given in appendix 2. Most of the action of RD can be seen by comparing Fig. 1B with Fig. 1C. Thus RD makes each unit and each property in a piece of input it is given into a list. It also gives every unit it reads in an additional first element, which is the atom or list serving as that unit's print-name. (If the unit has no print-name, NIL is used; (see the unit representing "flash of lightning" in Fig. 10). RD's most important action is to replace each word appearing in a piece of input with a pointer to the list stored with that word as its first definition. If a coder wishes to refer to some definition of a word other than its first, he must put in the input format not just the word, but a list of two elements. The second of these is the word, the first is the number of the definition he wants to refer to. Thus, if in input appears: (3 box), RD will replace this with the third definition of "box". By the same means, if the coder wished to refer to some unit within that third definition, he might write (3 2 5 box). RD would replace this with a pointer to the 5th element of the 2nd element of the 3rd definition of "box".

If RD attempts to find a definition (the first or any other) of some word, and cannot do so because that definition has not yet been encoded, RD creates an appropriate list, makes this the definition in question, and uses a pointer to this as the definition it requires. Thus RD will replace any later reference to the same definition by a pointer to this same list, since this now is that definition. (PD will be careful in the future to only add information to such a definition, so that any pointers to it will remain valid.)

While RD turns each ordinary unit or property into a list, it turns each set into an atom, and puts the members of the set, and

the set type marker — AOR, AND, etc. — onto the CDR of the atom. Thus in the internal memory a set is detectable by the presence of an atom — other than the atom NIL — in the place where one would expect a list, either a unit or a property. Associated with such an atom will be the elements of the set, which of course must be either all units or all properties as is required by the set's location.

A number of additional examples of encoded text appear as appendix 3.

APPENDIX I

Schematic of Input Readable Format

UNIT

(SUPERSET (PROPERTY) (PROPERTY) . . .)

UNIT	PROP	PROP
WORD	SET of PROPS	SET of PROPS
SET		

PROPERTY

(ATTRIBUTE VALUE (sub-property) (sub-property) . . .)

prop prop

1. UNIT — UNIT
2. WORD — WORD:
   a. VERB ⟹ {Direct Object, NIL}
   b. PREP. ⟹ {Object, NIL}
   c. ADJ. ⟹ NUMERAL
   d. ADV. ⟹ NUMERAL
3. Set of such units — Set of appropriate elements

SET

(settype member member . . .)

settype	member
"AOR"	Any legal datatype but numeral
"EOR"	
"AND"	
"SEQ"	

Syntax of Input Readable Format

UNIT → any English word, including NIL

UNIT → (UNIT + PROPLIST)
"superset"

UNIT → (SETTYPE + UNIT + UNITLIST)

UNIT → (RETRIEVALIST + word)

UNITLIST → UNIT + UNITLIST

UNITLIST → ϕ

PROPLIST → PROP + PROPLIST

PROPLIST → ϕ

PROP → (UNIT + UNIT PROPLIST)
"attribute" "value"

PROP → (SETTYPE + PROP + PROPLIST)

PROP → ("#" + a seven-digit octal numeral)

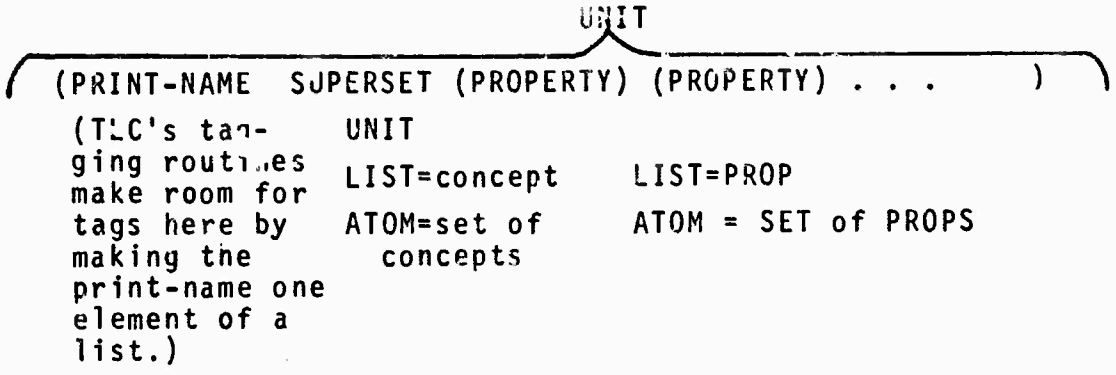
SETTYPE → "AOR," "EOR," "AND," "SEQ"

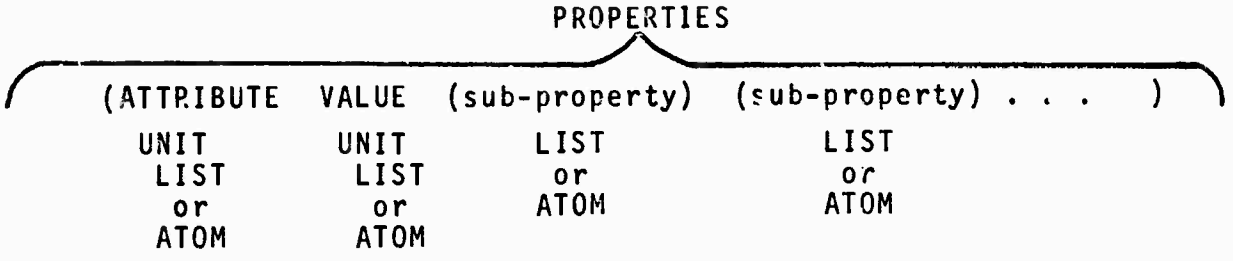
RETRIEVALIST → (NUMBER + NUMBERLIST + WORD)

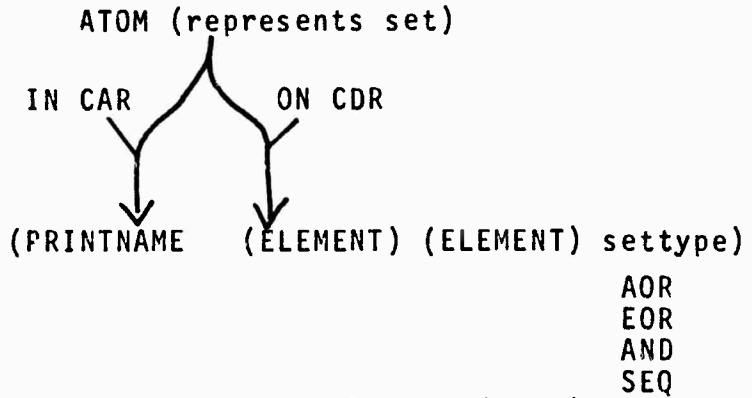
NUMBERLIST → NUMBER + NUMBERLIST

NUMBERLIST → ϕ

Schematic of Internal Memory Format







Note that the car of both an atom and of a unit (list) yields the print-name of that item. (In BBN-LISP the car of an atom is the same as its value, the cdr is the same as its description list.)

APPENDIX II

Routines for Translation In and Out of the Internal Memory

Arguments to PR

The first argument to PR is an atom. PR makes an output readable copy of all definitions associated with this atom, and binds these to the atom CWORK. It prints these out unless the second argument, NOSEE, is true.

Arguments to RD

RD takes two primary arguments. WORD is a word whose associated definition list the user wishes to expand or change. DEFLIS, the second argument is a list of elements, each of which is one of

(a) a negative number < -# > ;

(b) a positive number < # > ;

or (c) a definition in input-readable format < D >

This list is either NIL, or composed of any number of the following substrings:

(S1) < D > ;

(S2) < D # > ;

(S3) < # > ;

(S4) < -# D > ;

(S5) < -# D # > ;

(S6) < -# # > .

Note that each definition can be either a list in input-readable format or this list followed by a positive number which is used to indicate rarity of use.[1] An explanation of what each of (S1)-(S6)

[1] For instance, suppose the list of definitions HEY was:
((def1) (def2) 2 (def3) 1)
This would cause TLC not to search (def2) or (def3) until after two passes through def1 have been completed, and would cause it not to search (def3) until one more pass through (def1) and (def2) had been completed. Thus, a relatively high number would be inserted after a very rare definition to inhibit its searching until after that number of passes through more common meanings was completed.

causes is in the comments in RD's definition. Definitions are stored on the CADR of a word and are considered numbered from left to right.

The third argument to RD is NOSEE. RD calls PR in all cases, so that the atom OWORK is bound, but if NOSEE is True, no printout will occur.

```
  (PROGN (PRIN1 (QUOTE FILE" CREATED ")
      T)
    (PRIN1 (QUOTE Ø9/22/69" 1635:5Ø")
      T)
    (TERPRI T))
(DEFINEQ

(RD
  (LAMBDA (WORD DEFLIS NOSEE RETRIEVALIST)

          (* * Takes two primary arguments.
          WORD is the word whose definition wants to be
          created or changed. DEFLIS is a list of elements,
          each of which is one of: (a) a negative number <
          -# > ; (b) a positive number < # > ;
          or (c) a definition in input-readable format < D
          >. This list is either NIL or composed of any
          number of the following substrings:
          (S1) < D > ; (S2) < D # > ;
          (S3) < # > ; (S4) < -# D > ;
          (S5) < -# D # > ; (S6) < -# # >.
          Note that each definition can be either a list or
          a list and a number which is used to indicate
          rarity of use (cf text.) RD also calls the
          function MUNIT to create the proper internal
          representation of each definition.
          The results of each of the above formats are:
          (S1-S6) if word has no definitions, the definition
          list is put on the CADR of word and word is put on
          the list DATALIST; if DEFLIS is NIL, the word is
          initialized as above and its only definition
          becomes a list of its pname;
          (S1-S3) the definition or definition part is added
          to the end of the word's definition list;
          (S4-S6) -# indicates the position of an already
          existing definition which is to be replaced
          entirely (S5) or partially
          (S4,S6) "." If the third argument NOSEE is NIL,
          then (PR WORD) is performed.)
```

```
    (MAP DEFLIS (FUNCTION (LAMBDA (DEFL)
          (COND
            ((NOT (NUMBERP (CAR DEFL)))
              (ADEF WORD (MUNIT (CAR DEFL)
                  WORD)))
            ((AND (NUMBERP (CAR DEFL))
                (GREATERP (CAR DEFL)
                  -1)
                (ADEF WORD (CAR DEFL))))
            ((AND (NUMBERP (CADDR DEFL))
                (GREATERP (CADDR DEFL)
                  -1))
              (ADEF WORD (CAR DEFL)
                (MUNIT (CADR DEFL)
                  WORD)
                (CADDR DEFL))
              (RPLACD DEFL (CDDDR DEFL)))
            ((ADEF WORD (CAR DEFL)
                (MUNIT (CADR DEFL)
                  WORD))
              (RPLACD DEFL (CDDR DEFL)))))))
    (RETRIEVE)
    (PR WORD NOSEE)))

(ADEF
  (LAMBDA (WORD DEF1 DEF2 DEF3)                    (* Called by RD and does
                                                   most of its work.)
    (OR (MEMB WORD DATALIST)
      (ALPHA WORD DATALIST))
    (COND
      ((NULL DEF1)
        (CAADR (RPLACD WORD (CONS (CONS (CONS WORD))))))
      ((NULL (GDEFS WORD))
        (RPLACD WORD (CONS (CONS DEF1))))
      ((AND (NUMBERP DEF1)
          (GREATERP DEF1 -1)
          (NCONC (GDEFS WORD)
            (CONS DEF1))))
      ((AND (NOT (NUMBERP DEF1))
          (NULL (CDR (GDEF WORD))))
        (ADEF WORD -1 DEF1))
      ((NOT (NUMBERP DEF1))
        (NCONC (GDEFS WORD)
          (CONS DEF1)))
      ((NUMBERP DEF2)
        (RPLACA (CDR (NTHDEF WORD (ABS DEF1)))
          DEF2))
      ((NULL DEF3)
        (RPLACA (NTHDEF WORD (ABS DEF1))
          DEF2))
      ((RPLACA (SETQ DEF1 (NTHDEF WORD (ABS DEF1)))
          DEF2)
        (RPLACA (CDR DEF1)
          DEF3)))))
```

```
(GDEFS
  (LAMBDA (WORD)                                    (* Returns list of
                                                    definition of word.)
    (CADR WORD)))

(GDEF
  (LAMBDA (UNIT)

          (* * Returns the first element of the definition
          list of word (hopefully not a number))


    (CAADR UNIT;))

(NTHDEF
  (LAMBDA (WORD N N1)

          (* * Returns the tail of a words definition list
          begining with the NTH definition
          (where positive numbers occurring as definitions
          are considered as belonging to the immediately
          preceding definition))


    (SETQ N1 0)
    (MAPTL (GDEFS WORD)
      (FUNCTION (LAMBDA (DEF)
          (COND
            ((NUMBERP DEF)
              NIL)
            ((ZOP (SETQ N1 (ADD1 N1))
                N)))))))))

(NDEFS
  (LAMBDA (WORD N1)                                 (* Returns number of
                                                    definitions of WORD.)
    (SETQ N1 0)
    (MAPC (GDEFS WORD)
      (FUNCTION (LAMBDA (DEF)
          (COND
            ((NUMBERP DEF))
            ((SETQ N1 (ADD1 N1))))))))
    N1))
```

```
(NADEFS
  (LAMBDA (WORD N LDEF)                          (* Adds N definitions of
                                                 form (WORD) to WORD.)
    (COND
      ((ZEROP N)
        (CAR (LAST LDEF)))
      ((NADEFS WORD (SUB1 N)
          (ADEF WORD (CONS WORD)))))))

(MUNIT
  (LAMBDA (UNIT PNAME)

          (* * Makes a unit from UNIT which is in
          input-readable format, and uses PNAME as the pname
          of the unit if it is not NIL, else uses
          (NIL NIL NIL) "." Is inter-recursive with MPROP.)


    (COND
      ((NULL UNIT)
        NIL)
      ((NUMBERP UNIT))
      ((AND (ATOM UNIT)
          (NOT (EQ WORD UNIT)))
        (OR (GDEF UNIT)
          (ADEF UNIT)))
      ((OR (NUMBERP (CAR UNIT))
          (AND (EQ WORD UNIT)
            (SETQ UNIT (LIST 1 UNIT))))
        (CAR (SETQ RETRIEVALIST (CONS (LIST (REMOVE
                    (CAR (LAST UNIT))
                  UNIT)
                (CAR (LAST UNIT)))
              RETRIEVALIST))))
      ((MEMB (CAR UNIT)
          (QUOTE (AOR EOR AND SEQ)))
        (MSET UNIT))
      ((APPEND (LIST PNAME (MUNIT (CAR UNIT)))
          (MAPCAR (CDR UNIT)
            (FUNCTION MPROP)))))))
```

```
(MPROP
  (LAMBDA (PROP)                                    (* Makes a property fro-
                                                    PROP. Is recursive with
                                                    MUNIT.)
    (COND
      ((EQ (CAR PROP)
           (QUOTE #))
        PROP)
      ((MEMB (CAR PROP)
           (QUOTE (AOR EOR AND SEQ)))
        (MSET PROP T))
      ((CONS (MUNIT (CAR PROP))
           (CONS (MUNIT (CADR PROP))
             (MAPCAR (CDDR PROP)
               (FUNCTION MPROP)))))))))

(MSET
  (LAMBDA (ELEMENT PROP? SET)

          (* * Makes a set. Checks to see if this set has
          already been created by seeing if the printnames
          of existing sets in SETLIST are EQUAL to unit.
          Creates a set by calling GENSET and putting the
          printname of the set on the CAR of the setname ,
          and on its CDR a list of pointers to the units or
          properties in the set, followed by the settype.
          Called by MUNIT and MPROP.)


    (COND
      ((CAR (MAPTL SETLIST (FUNCTION (LAMBDA (SETN)
                (EQUAL (OSET SETN)
                  ELEMENT))))))
      ((RPLACD (SETQ SET (GENSET))
           (APPEND (CONS (MAPCAR (CDR ELEMENT)
                  (COND
                    (PROP? (FUNCTION MPROP))
                    ((FUNCTION MUNIT)))))
             (CONS (CAR ELEMENT))))
        (RPLACA SET (LIST NIL NIL (COND
              (PROP? (QUOTE PROPS))
              ((QUOTE UNITS)))))))))
```

```
(GENSET
  (LAMBDA NIL

          (* * Generates setnames of form SET + n by adding
          1 to the current value of the global atom LASTSET
          and puts them on the list SETLIST.
          Returns current setname. Called by MSET.)


    (CAR (SETQ SETLIST (CONS (PACK (APPEND (QUOTE (S E T))
              (UNPACK (SETQ LASTSET (ADD1 LASTSET)))))
          SETLIST)))))

(RETRIEVE
  (LAMBDA NIL

          (* * Performs retrieval of definitions or their
          parts from words that were pointed to by a list of
          numbers in the input-readable FORMAT.
          These are stored on the atom RETRIEVALIST until
          processed by RETRIEVE.)


    (MAPC RETRIEVALIST (FUNCTION (LAMBDA (UNIT DEF)
          (SETQ DEF (RETRIEVED (CAR UNIT)
              (CADR UNIT)))
          (RPLACA UNIT (CAR DEF))
          (RPLACD UNIT (CDR DEF)))))))

(RETRIEVED
  (LAMBDA (NUMS WORD DEF)

          (* * Called by RETRIEVE and locates the
          appropriate definition of WORD as specified by
          (CAR NUMS) where NUMS is the list of numbers
          pointing to some part of a definition.
          if the definition called is non-existent, will add
          as many definitions of the form *cons *word as
          needded to make the last definition be number *car
          *nums.)


    (SETQ DEF (COND
        ((CAR (NTHDEF WORD (CAR NUMS))))
        ((NADEFS WORD (DIFFERENCE (CAR NUMS)
              (NDEFS WORD))))))
    (RETRIEVED (CDR NUMS)
      DEF)))
```

```
(RETRIEVEU
  (LAMBDA (NUMS UNIT)                                    (* Retrieves from a
                                                         unit. Is inter-recursive
                                                         with RETRIEVEP.)
    (COND
      ((NULL NUMS)
        UNIT)
      ((EQP (CAR NUMS)
            1)
        (RETRIEVEU (CDR NUMS)
          (CADR UNIT)))
      ((RETRIEVEP (CDR NUMS)
          (CAR (NTH UNIT (ADD1 (CAR NUMS))))))))))

(RETRIEVEP
  (LAMBDA (NUMS PROP)                                    (* Retrieves from a
                                                         property. Is
                                                         inter-recursive with
                                                         RETRIEVEU.)
    (COND
      ((NULL NUMS)
        PROP)
      ((LESSP (CAR NUMS)
            3)
        (RETRIEVEU (CDR NUMS)
          (CAR (NTH PROP (CAR NUMS)))))
      ((RETRIEVEP (CDR NUMS)
          (CAR (NTH PROP (CAR NUMS))))))))

(PR
  (LAMBDA (WORD NOSEE USEDUNITS USEDPROPS USEDSETS)

          (* * Sets the global atom OWORK to a list of the
          definitions of WORD in output-readable FORMAT.
          If NOSEE is NIL it also prints out this list using
          PRINTDEF. Note: in the case of a setname, OWORK
          becomes a list of the pname of the set)
```

```
    (SETQ OWORK (COND
        ((MEMB WORD SETLIST)
          (CONS (OSET WORD)))
        ((MAPCAR (GDEFS WORD)
            'FUNCTION (LAMBDA (DEF)
                (OUNIT (COND
                    ((CDR DEF)
                      (CONS NIL (CDR DEF)))
                    (DEF)))))))))
    (COND
      (NOSEE (PRINT WORD)
        (TERPRI))
      (T (TERPRI)
        (TERPRI)
        (PRIN1 WORD)
        (PRINT (QUOTE :))
        (MAPC OWORK (FUNCTION (LAMBDA (WORK)
              (PRINTDEF WORK)
              (TERPRI)))))))))

(GPNAME
  (LAMBDA (UNIT)                                    (* Gets pname of UNIT)
    (COND
      ((NUMBERP UNIT))
      ((AND (ATOM (CAR UNIT))
          (NOT (NULL (CAR UNIT))))
        (CAR UNIT))
      ((CADDAR UNIT))
      ((GPNAME (CADR UNIT))))))

(OUNIT
  (LAMBDA (UNIT)

          (* * Produces output-readable FORMAT from internal
          UNIT. Is inter-recursive with OPROP)


    (COND
      ((NULL UNIT)
        NIL)
      ((MEMB UNIT SETLIST)
        (OSET UNIT))
      ((NUMBERP UNIT))
      ((AND (ATOM (CAR UNIT))
          (NOT (NULL (CAR UNIT)))
          (NULL (CDR UNIT)))
        (CAR UNIT))
      ((SUNIT UNIT))
      ((COND
          ((ATOM (CAR UNIT))
            (CAR UNIT))
          ((CADDAR UNIT))))
      ((CONS (OUNIT (CADR UNIT))
          (MAPCAR (CDDR UNIT)
            (FUNCTION OPROP))))))))
```

```
(SUNIT
  (LAMBDA (UNIT)

          (* * Called by OUNIT to check on repeated
          references to same nonatomic UNIT)


    (COND
      ((MEMB (CDR UNIT)
          USEDUNITS)
        (CONS (QUOTE *THIS*)
          (GPNAME UNIT)))
      ((NULL (CADDAR UNIT))
        (NOT (SETQ USEDUNITS (CONS (CDR UNIT)
             USEDUNITS))))))))

(OPROP
  (LAMBDA (PROP)

          (* * Produces output-readable format from internal
          PROP. IS inter-recursive with OUNIT.)


    (COND
      ((ATOM PROP)
        (OSET PROP))
      ((EQ (CAR PROP)
          (QUOTE #))
        PROP)
      ((SPROP PROP))
      ((CONS (OUNIT (CAR PROP))
          (CONS (OUNIT (CADR PROP))
            (MAPCAR (CDDR PROP)
              (FUNCTION OPROP))))))))

(SPROP
  (LAMBDA (PROP)

          (* * Called by OPROP to check on repeated
          references to same property.)


    (COND
      ((MEMB PROP USEDPROPS)
        (QUOTE *USEDPROP*))
      ((SETQ USEDPROPS (CONS PROP USEDPROPS))
        NIL))))
```

```
(OSET
  (LAMBDA (SET)

          (* * Produces output-readable format from internal
          SET. Is inter-recursive with OUNIT and OPROP.)


    (COND
      ((SSET SET))
      ((APPEND (LAST (CDR SET))
          (MAPCAR (GDEFS SET)
            (COND
              ((EQP (CADDAR SET)
                  (QUOTE UNITS))
                (FUNCTION OUNIT))
              ((FUNCTION OPROP))))))))))

(SSET
  (LAMBDA (SET)

          (* * Called by OSET to check on repeated
          references to the same setname.)


    (COND
      ((MEMB SET USEDSETS)
        (CONS (QUOTE *THOSE*)
          SET))
      ((SETQ USEDSETS (CONS SET USEDSETS))
        NIL))))

(MAPTL
  (LAMBDA (MAPTLIS MAPTFN)

          (* * Operates like other mapping functions.
          Returns the rest of its first argument if
          (MAPTFN (CAR MAPTLIS)) is not NIL, else NIL.)


    (COND
      (MAPTLIS (COND
          ((MAPTFN (CAR MAPTLIS))
            MAPTLIS)
          (T (MAPTL (CDR MAPTLIS)
              MAPTFN)))))))
```

```
(ALPHA
  (NLAMBDA (THING LIS)                                    (* Basic alphabetizes a
                                                          list.)
    (SETQ THING (EVAL THING))
    (COND
      ((NULL (EVAL LIS))
        (RPLACA LIS (CONS THING)))
      (T (PROG (NEWLIS CHAR A B)
               (SETQ CHAR Ø)
               (SETQ NEWLIS (EVAL LIS))
          LP   (SETQ CHAR (ADD1 CHAR))
          LP1  (COND
                 ((NULL NEWLIS)
                   (NCONC (EVAL LIS)
                     (CONS THING))
                   (RETURN))
                 ((LESSP (SETQ B (LOC (NTHCHAR (CAR NEWLIS)
                                               CHAR)))
                         (SETQ A (LOC (NTHCHAR THING CHAR))))
                   (SETQ NEWLIS (CDR NEWLIS))
                   (SETQ CHAR 1)
                   (GO LP1))
                 ((EQ (CAR NEWLIS)
                      THING)
                   (RETURN))
                 ((EQ A B)
                   (GO LP))
                 ((NULL (NTHCHAR (CAR NEWLIS)
                                 CHAR))
                   (ATTACH THING (CDR NEWLIS))
                   (RETURN))
                 (T (ATTACH THING NEWLIS)
                   (RETURN)))
          )))))
)
  (PRINT (QUOTE RD-PR-FNS))
  (RPAQQ RD-PR-FNS (RD ADEF GDEFS GDEF NTHDEF NDEFS NADEFS
        MUNIT MPROP MSET GENSET RETRIEVE RETRIEVED RETRIEVEU
        RETRIEVEP PR GPNAME OUNIT SUNIT OPROP SPROP OSET
        SSET MAPTL ALPHA))
  (SETQ DATALIST)
  (SETQ SETLIST)
  (SETQ LASTSET Ø)
STOP
```

APPENDIX III

Additional Examples

The man who hits the ball

(MAN (HIT (BALL) (BY *)))

The man who was hit by the ball

(MAN (HIT *) (BY BALL)))

The man who hit the tree with the ball

(MAN (HIT (TREE) (BY *) (WITH BALL)))

The man who hit the dog with his hand

(MAN (HIT (DOG) (BY *) (WITH (HAND(OF *)))))

The lion's den

(DEN (OF LION))

In the lion's den

(IN (DEN (OF LION)))

In a corner of the lion's den

(IN (CORNER (OF (DEN (OF LION)))))

The tall man who stands bravely in the corner of the lion's den

```
(MAN (TALL 16)
     (STAND NIL (BY *)
                (IN (CORNER (OF (DEN (OF LION)))))))
```

The man who buys a book

```
(MAN (BUY BOOK (BY *)))
```

The man who buys a book by Kafka

```
(MAN (BUY (BOOK (BY KAFKA)) (BY *))
```

Note here that each "by" is different. Thus the word "by" alone is an insufficient pointer to the correct meaning that must be used. In entering the data to RD it might look like this:

```
(MAN (BUY BOOK ((4 BY) *) ((2 BY) KAFKA)))
```

The man who runs the race

```
(MAN (RUN RACE (BY *)))
```

The man who runs quickly

(If there is no Direct Object available enter NIL in the value position.)

```
(MAN (RUN NIL (BY *) (QUICK 16)))
```

The man who gives John the book

(Always put Direct Object in value position)

```
(MAN (GIVE BOOK (TO JOHN) (BY *)))
```

The man who goes to the store

(Intransitive verbs always have NIL in value position)

(MAN (GO NIL (BY *) (TO STORE)))

The man who helps the girl (to) buy a necklace

(MAN (HELP (GIRL) (BY *)) (TO (*TO* (BUY NECKLACE (BY *)))))

2 girls in a car

((girl (N 1002000)) (in (car)))

2 girls in cars

((girl (in (car))) (N 1002000))

Coding Active Sentences

The man who calls the play

(MAN (CALL PLAY (BY *)))

This is a unit which might be put on the CDR of "UMPIRE" and have some Printname, such as "UMPIRE".

On the other hand the sentence:

The man calls the play

will only be coded as a property, not as a unit:

(CALL PLAY (BY (MAN)))

We may wish to attach a pointer pointing to this property from one or more units, such as the unit representing this man, or a unit having as superset PLAY or *TO*. Since the location of pointer(s) to a coded assertion is not determinable from the assertion itself, sentences (except defining sentences) must be encoded as (complex) properties, with the sentence's main verb as main attribute.

In the foregoing examples, as in the paper, we have used arrows instead of sticking strictly to input readable format. In input format there can be no arrows, but arrows can always be removed by associating pieces of data with atoms. If necessary these can be purely arbitrary atoms; RD always demands some atom with which to associate the data it is given. The following examples are coded in strict input format, with atoms used which make arrows unnecessary.

Suppose there are 2 definitions of "pen":

"an instrument with which one writes"

"a place in which pigs are kept"

To encode these we give RD:

```
(PEN (INSTRUMENT (WRITE NIL (WITH PEN)))
     (PLACE (KEEP (PIG (# 2001000)) (IN (2 PEN)))))
```

Then we may code:

The farm which has seven pens

```
(FARM ((2 PEN) 1007000))
```

Pigs (that are in pens) which are usually dirty.

```
((2 2 2 PEN) (DIRTY 16))
```

The Reuben James: the ship that passed the iceberg which was mostly under water.

```
(REUBEN JAMES (SHIP(PASS (ICEBERG (UNDER WATER (  26))))))
```

X30 is the submarine that passes under that iceberg.

```
(X30 (SUBMARINE (PASS ICEBERG (UNDER (1 2 2   REUBEN-JAMES)))))
```

BIBLIOGRAPHY

Chomsky, N. (1965). Aspects of the Theory of Syntax. Cambridge, Massachusetts: The M.I.T. Press.

Collins, A.M., and Quillian, M.R. (1968). Retrieval Time from Semantic Memory. Report No. 1692. Cambridge, Massachusetts: Bolt Beranek and Newman Inc. Also in The Journal of Verbal Learning and Verbal Behavior, 8, 240-247 (1969).

Collins, A.M., and Quillian, M.R. (1969). Semantic memory and language comprehension. In L. Gregg, Cognition in Learning and Memory. New York: John Wiley & Sons. (forthcoming)

Minsky, M. (1969). Semantic Information Processing. Cambridge, Massachusetts: M.I.T. Press.

Piaget, J. (1950). The psychology of intelligence. Translated by M. Cook and D.E. Berlyne. London, England: Routledge and Kegan Paul.

Quillian, M.R. (1966). Semantic Memory. Report No. 1352. Cambridge, Massachusetts: Bolt Beranek and Newman Inc. The central part of this paper is published as, "Word concepts: a theory and simulation of some basic semantic capabilities." and is reproduced in Minsky (1969).

Quillian, M.R. (1969). The teachable language comprehender: a simulation program and theory of language. Communications of the ACM . August 1969.

Quillian, M.R., Wortman, P. and Baylor, G.W. (1965). The programmable Piaget: behavior from the standpoint of a radical computerist. Unpublished dittoed paper. Carnegie Institute of Technology.

Quine, W.V. (1960). Word and Object. Cambridge, Massachusetts: The M.I.T. Press.

Simon, H.A. (1969). The Sciences of the Artificial. Cambridge, Massachusetts: The M.I.T. Press.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
CAPTURING CONCEPTS IN A SEMANTIC NET

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Scientific Interim

5. AUTHOR(S) *(First name, middle initial, last name)*
Anthony Bell
M. Ross Quillian

6. REPORT DATE
6 October 1969

7a. TOTAL NO. OF PAGES
52

7b. NO. OF REFS
10

8a. CONTRACT OR GRANT NO.
F19628-68-C-0125/ARPA Order No. 627

b. PROJECT NO. 8668

c DoD Element 61101D

d DoD Subelement n/a

9a. ORIGINATOR'S REPORT NUMBER(S)
BBN Report No. 1885
Scientific Report No. 13

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
AFCRL-69-0438

10. DISTRIBUTION STATEMENT
1-Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

11. SUPPLEMENTARY NOTES
This research was supported by the Advanced Research Projects Agency under ARPA Order No. 627

12. SPONSORING MILITARY ACTIVITY
Air Force Cambridge Research Laboratories (CRB)
L.G. Hanscom Field
Bedford, Massachusetts 01730

13. ABSTRACT

A working memory model based on a semantic network is described in detail. Some advantages and disadvantages of such a model are discussed. An attempt is made to enable a reader to learn to perform the formidable task of representing data in the memory format. Since the actual memory is not easily read (or written), a set of LISP programs are included which make these tasks manageable.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

A-31409

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Semantics						
Semantic Network						
LISP						
English Language Comprehension						
Memory Model						