BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

AFCRL-69-0108

31 January 1969

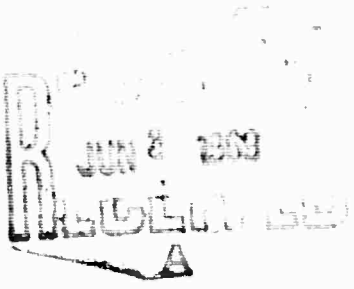
THE TEACHABLE LANGUAGE COMPREHENDER: A SIMULATION PROGRAM AND THEORY OF LANGUAGE

M. Ross Quillian

Scientific Report No. 10
Contract No. F19628-68-C-0125
Project No. 8668
Contract Monitor: Hans H. 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 sponsored by the Advanced Research Projects Agency under ARPA Order No. 627

AFCRL-69-0108

31 January 1969

THE TEACHABLE LANGUAGE COMPREHENDER: A SIMULATION PROGRAM AND THEORY OF LANGUAGE

M. Ross Quillian

BOLT BERANEK AND NEWMAN INC
50 Moulton Street
Cambridge, Massachusetts 02138

Scientific Report No. 10
Contract No. F19628-68-C-0125
Project No. 8668
Contract Monitor: Hans H. 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 sponsored by the Advanced Research Projects Agency under ARPA Order No. 627

THE TEACHABLE LANGUAGE COMPREHENDER: A SIMULATION PROGRAM AND THEORY OF LANGUAGE*

M. Ross Quillian

January 1969

*This research was supported primarily by the Advanced Research Projects Agency, monitored by the Air Force Cambridge Research Laboratories under Contract No. F19628-68-C-0125; and in part by the Aerospace Medical Research Laboratories, Aerospace Medical Division, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, under Contract F33615-67-C-1982 with Bolt Beranek and Newman Inc.

The author extends his most grateful thanks to Daniel G. Bobrow, Principal Investigator on the ARPA project, for invaluable assistance in writing the TLC program, to Tony Bell, who wrote the program's sentence generation routines; and to Allan Collins, who is attempting to test its psychological validity. Thanks also to Joseph Becker and Bruce Fraser, who along with those just mentioned provided valuable criticisms of various versions of the present report. This paper is to appear in the Computational Linguistics section of the Communications of the ACM in 1969.

ABSTRACT

The Teachable Language Comprehender (TLC) is a program designed to be capable of being taught to "comprehend" English text. When text which the program has not seen before is input to it, it comprehends that text by correctly relating each (explicit or implicit) assertion of the new text to a large memory. This memory is a "semantic network" representing factual assertions about the world.

The program also creates copies of the parts of its memory which have been found to relate to the new text, adapting and combining these copies to represent the meaning of the new text. By this means, the meaning of all text the program successfully comprehends is encoded into the same format as that of the memory. In this form it can be added into the memory.

Both factual assertions for the memory and the capabilities for correctly relating text to the memory's prior contents are to be taught to the program as they are needed. TLC presently contains a relatively small number of examples of such assertions and capabilities, but within the system notations for expressing either of these are provided. Thus, the program now corresponds to a general process for comprehending language, and provides a methodology for adding the additional information this process requires to actually comprehend text of any particular kind.

The memory structure and comprehension process of TLC allow new factual assertions and capabilities for relating text to such stored assertions to generalize automatically. That is, once such an assertion or capability is put into the system, it becomes available to help comprehend a great many other sentences in the future. Thus, adding a single factual assertion or linguistic capability will often provide a large increment in TLC's effective knowledge of the world, and in its overall ability to comprehend text.

The program's strategy is presented here as a general theory of language comprehension.

BLANK PAGE

I. INTRODUCTION

A. Goals and Sample Output

The ultimate goal of the research to be described here is to develop a computer program that could comprehend newspapers, textbooks, encyclopedias, and other written text. That is, the program should be able to extract and somehow retain meaning from natural language text it has not seen before at a level of skill comparable to that of human readers. The present paper is an overview of a program - called TLC, for Teachable Language Comprehender - which aspires to this goal.

TLC is in the development stage and so far only works on certain isolated phrases and sentences. Nevertheless, a large part of its strategy is now worked out in considerable detail, as is the structure of the memory TLC works with. Together these constitute a theory of what text comprehension is, and of how to achieve it; it is really this theory that we wish to describe here. Viewed as a theory, a good part of TLC's strategy is independent of the kind of mechanism that carries it out; one may think of this mechanism as a computer, a person's brain, or whatever else could do it.[1] Our system is implemented in BBN-LISP (Bobrow, Murphy and Teitelman, 1968) on an SDS 940.

---

1 We also happen to believe that, given the present state of psychological theories, almost any program able to perform some task previously limited to humans will represent an advance in the psychological theory of that performance. Therefore, while the reader who disagrees or who has no interest in human behavior can read TLC's strategy strictly as program specification, we choose to regard this strategy also as psychological theory, and will speak of a computer and a person interchangeably as our example of the mechanism carrying it out. Reaction time data supporting the notion that people's semantic memories have at least an overall organization like that of TLC's is reported in Collins and Quillian (1968).

"Comprehending" text is here defined as the relating of assertions made or implied in that text to information previously stored as part of the comprehender's general "knowledge of the world." Correspondingly, the central aim of TLC is the ability to appropriately relate text to the correct pieces of stored general knowledge of the world. We assume that this is not only the basic process involved in the comprehension of language, but also in a great many other perceptual and cognitive functions.

TLC's second important assumption is that all of a comprehender's knowledge of the world is stored in the same kind of memory - that is, that all the various pieces of information in this memory are encoded in a homogeneous, well-defined format. TLC's memory notation and organization constitute an attempt to develop such a format, which is both uniform enough to be manageable by definable procedures, yet rich enough to allow representation of anything that can be stated in natural language. This memory format is a further development of that used in a previous "semantic memory" program (see Quillian, 1966, 1967, or in Minsky, 1968); it amounts essentially to a highly interconnected network of nodes and relations among nodes.

If comprehension of text by TLC only meant for the program to relate the text appropriately to information in its memory, however, its comprehension would not necessarily be discernable to an outside observer, nor would it necessarily leave any trace to alter the program's future performance. Therefore, comprehension in TLC is always followed immediately by encoding <u>in TLC's memory format</u> a representation of what the program decides the meaning of the text to be. Figure 1A gives an example of the program's output.

```
READ(LAWYER'S CLIENT)


OUTPUT1:

  ((CLIENT (EMPLOY (LAWYER)
         (BY (*THIS* . CLIENT)))


OUTPUT2:

  UNDER DISCUSSION IS A CLIENT WHO EMPLOYS A LAWYER.
```

FIG. 1A An Example of TLC's Output

This figure shows a small phrase input to one version of the program and TLC's output, the machine's representation of what it decided the phrase meant. The program always expresses its output in two forms: the first showing the encoding of the input into memory format, the second a translation of this into English.

Thus Fig. 1A shows that when TLC was asked to read the phrase "lawyer's client," its comprehension was expressed by its output at the bottom of Fig. 1A.

For the moment all that needs to be noticed about the non-English version of the output is that the machine's comprehension of the input phrase has been to build a fairly elaborate internal structure - the computer's counterpart of a cognitive structure - which as the English version indicates is considerably more explicit and detailed than the piece of input text itself. The important point here is that in TLC the function of text is viewed not as explicitly stating information for a reader, but rather as directing the reader to construct for himself various cognitive structures. These structures will in large part represent assertions that go beyond anything explicit in the text itself, which is possible because in constructing such structures the reader includes pieces of information drawn from his prior memory. At the same time, a text is able to communicate something new to the reader, something which he did <u>not</u> already have in his memory, by dictating ways in which he must <u>adapt</u> and <u>recombine</u> the pieces of information he draws from memory. These pieces then are used as components for building the new cognitive structures that represent his comprehension.

In other words, given a piece of input text to comprehend, TLC first locates related pieces of information (scattered about) in its memory, and then creates and adapts copies of these to recombine into its representation of what the text means.

Defining text comprehension in this way leads to viewing the richness of a given comprehension of a piece of text as the number of related memory items that the comprehender finds. For instance, the output shown in Fig. 1B illustrates comprehension of "lawyers client" of greater richness than does Fig. 1A, since an additional related item has been found and used to create the output. TLC can presently be made to produce the Fig. 1B output, but (alas) only at the cost of overinterpreting other phrases. The reason for this difficulty and a possible extension of TLC to overcome it will be discussed below in Section 3F. For the moment, all that needs to be noticed is that one can imagine comprehensions of greater richness than TLC can safely produce with its present techniques.

Although TLC cannot necessarily recognize all the things in memory that are in any way related to a text, it is designed on the assumption that it may have to search a very sizable amount of memory. Consider a piece of text such as:

> One recalls an American president who was once involved in an incident in which he completely severed the main trunk of a small fruit tree. He went and reported his action to his father.

Most readers will recognize at some point as they read this that the earlier president mentioned is Washington, the incident being the one in which he cut down his father's cherry tree. Unless the text is related to this fact in the reader's memory, the paragraph has not been very well comprehended. Notice that native speakers can establish this relationship even though they may not have heard or thought about the cherry tree story for ten or more years, and despite the fact that the language in which the incident is described above is almost certainly different

```
READ(LAWYER'S CLIENT)


OUTPUT1:
  ((CLIENT (EMPLOY (LAWYER)
         (BY (*THIS* . CLIENT)))
      ((AOR REPRESENT ADVISE)
         (*THIS* . CLIENT)
         (BY (*THIS* . LAWYER))
         (IN (MATTER (TYPE LEGAL))))))


OUTPUT2
  UNDER DISCUSSION IS A CLIENT WHO EMPLOYS A LAWYER : HE IS A
  CLIENT WHO IS REPRESENTED OR ADVISED BY THIS LAWYER IN A LEGAL
  MATTER
```

FIG. 1B An Example of Output Illustrating Greater "Richness" of Comprehension

in terminology, in syntax, and in surrounding context, from any language in which they have ever heard (or thought about) the story before. What the reader must have, then, as he reads the text above, is an extremely versatile ability to recognize the appropriate chunk of memory information from among literally thousands of others he may since have learned about "presidents," about "fruit trees," and about "fathers." Most of the following is an effort to describe how TLC attempts to develop such an ability. As part of this description we will first describe the memory, and then trace TLC's processing of the example in Fig. 1 in some detail.

Before this, however, let us briefly indicate where TLC stands in relation to other work. First, the project is perhaps closest to efforts such as those of the TEMPO project (Thompson et al, 1965) or of the SDC project (Simmons, Burger and Schwartz, 1968.) These projects share with TLC the aim of responding to English input by using very large, very general stores of information. The fact that they are designed as question answerers rather than as language interpreters turns out not to be especially important; most of the same problems must be faced in either case. Programs designed by these three projects differ from well-known programs such as those of Bobrow or Raphael, in that the latter aim at dealing with English in one strictly limited subject matter at a time and do not attempt to use an information store of a completely general sort, even though they make some use of "global" information. (See papers by the above two authors in Minsky, 1968.) Up to now, programs aimed at dealing with such a limited subject matter have been more impressive at answering questions posed to them in English than have programs based on a general store of information. (See the survey by Simmons, 1965; a recent survey is Siklossy and Simon, 1968.)

For some text deep comprehension requires reasoning mathematically, visually, or in some other specialized way; to do this a program probably will have to incorporate techniques such as those used in the Bobrow and Raphael programs, as well as in others such as G.P.S. (Newell et al. 1962). In this regard see also the programs in Feigenbaum and Feldman, 1963, Reitman, 1965, and Weizenbaum, 1967.) However, we assume that there is a common core process that underlies the reading of all text, whether newspapers, children's fiction, or whatever, and it is this core process that TLC attempts to model.

The relation between TLC, a semantic performance model, and the syntactic "competence" models of transformational linguistics (Chomsky, 1965) is not clear. The efforts that have been made so far to attach "semantics" to transformational models seem, to this writer at least, to have achieved little success (Woods, 1968, being the most significant attempt.) Correspondingly, TLC so far works with syntax on a relatively local, primitive basis, which has very little if anything new to say to a linguist about syntax. TLC does differ from most other projects of its type by not being specifically designed to work on the output of a parsing program, although a possible way of amalgamating one particular recent parsing program with TLC will be described below.

E. Teaching TLC To Comprehend

TLC's memory contains two kinds of material: factual assertions about the world, and what will be called "form tests." Form tests constitute the syntactic part of the program's ability to recognize that a phrase or sentence of an input text is related to some particular concept stored in its memory. That is, the ability to correctly relate a piece of text to memory depends in part on overcoming whatever difficulties the syntax of that text may pose to recognizing its relations to the memory, and this specific syntactic capability is what form tests provide. As the cherry tree example above illustrates, people's ability to cut through syntactic variation in recognizing relationships to their prior knowledge is incredibly good.

TLC's memory is eventually going to have to contain as much factual information as a person's memory if the program is to comprehend as broad a range of text. However, any one particular phrase or clause will never relate to all the pieces of information in such a memory, but only to something on the order of five or ten such pieces. Similarly, while TLC must eventually have as good an overall syntactic ability as a person has in order to recognize paraphrases of things in its memory, comprehension of any one phrase or sentence will never require all of this knowledge of syntax. Therefore, TLC is designed to fragment this ability into a great many separate form tests, so that both the program's memory for factual assertions and its syntactic ability to recognize relationships can be built up a piece at a time as each piece becomes necessary for the comprehension of some particular fragment of text.

To facilitate this, TLC is built to work in close interaction with a human monitor, its teacher, who provides each piece of factual knowledge and each form test as the program needs it. Using an on-line teletype, the monitor can oversee the program's attempts to read text, approve or disapprove of each step it takes, and

provide it with additional factual information or form tests as these are required. The principle components of the TLC system are shown in Fig. 2. Any piece of knowledge or form test that the monitor provides for the program is permanently retained in its memory, which thus gets built up piece by piece.

Our plan is to begin with, say, 20 different children's books dealing with firemen, and have TLC read all of these under supervision of the monitor. We anticipate that although the program will require a great deal of input by the monitor as it reads the first book, it will tend to require less and less on successive books as it accumulates knowledge and form tests pertinent to the comprehension of such material. Thus the hope is that by the 20th book TLC will be requiring very little aid, and that, over a long (but finite) period of general reading, the monitor (teacher) will become completely unnecessary, at least within given subject matter areas. Reasons for expecting this are brought together in Section III D. below. However, to date no serious attempt has been made to actually amass either factual data or form tests in this way, since almost all our effort has been concerned with developing the executive TLC program. Our method so far has been simply to give TLC sample inputs and memory structures, attempting to devise counterexample cases to its successful performances.

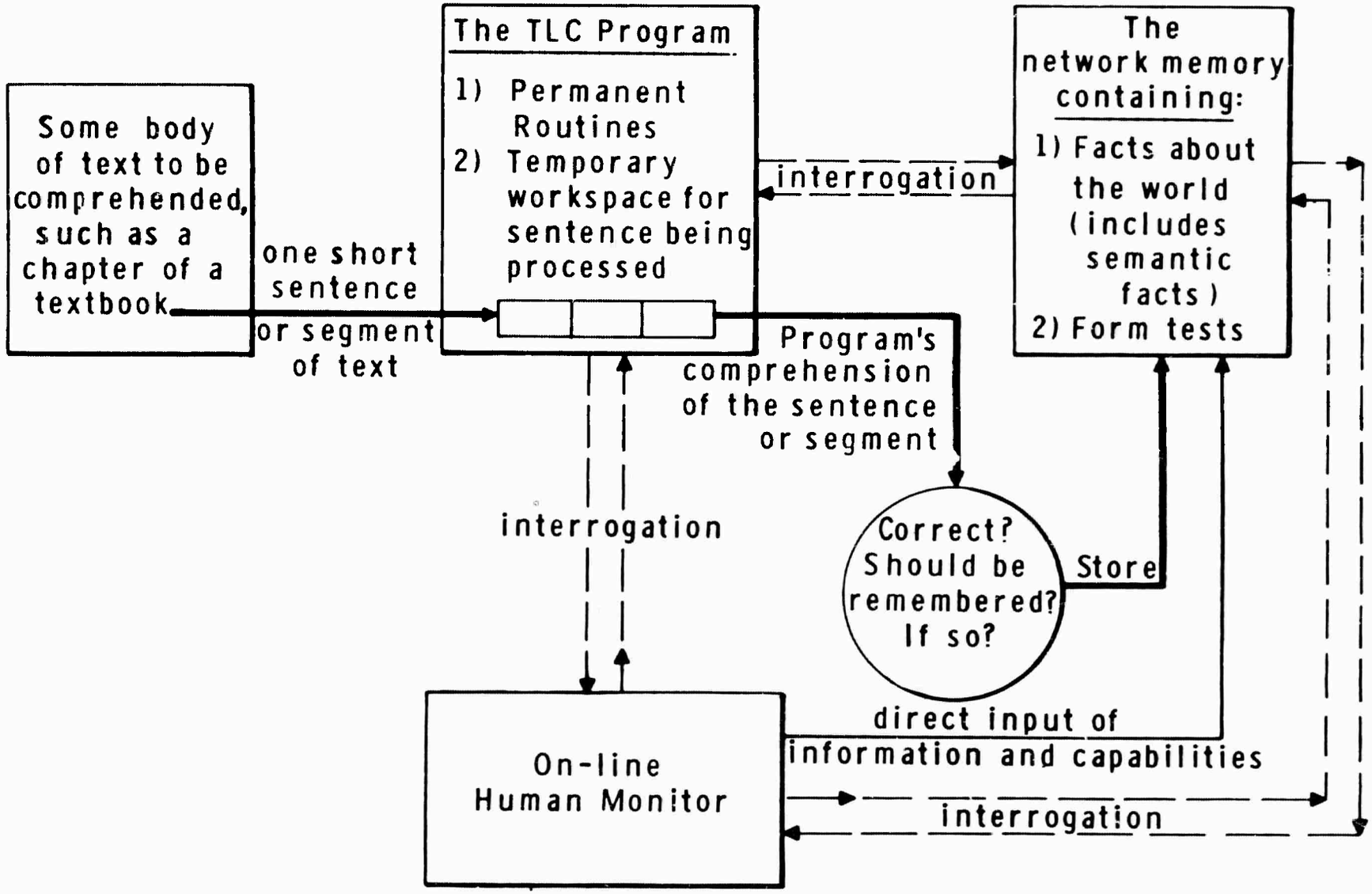


FIG.2 Principle Components and Interactions of the TLC System

II. THE MEMORY

A. The Format For Factual Information

It will be necessary to describe the memory format in some detail. The general aim of this format is to allow representation of everything uniformly enough so that it can be dealt with by specifiable procedures, while at the same time being rich enough to allow encoding of natural language without loss of information. Reconciling these aims is not a trivial undertaking, and a great amount of trial, error and effort has gone into the design of the format.

First, and foremost, <u>all</u> 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 use to represent any of the kinds of thing which can be represented in English by a single word, a noun phrase, a sentence, or some longer body of test. 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 3 illustrates a piece of factual information encoded in the memory. This figure differs from the actual memory network in that, for the sake of readability, some pointers are shown as going to words written all in capitals. Actually, these pointers in the memory always go to other <u>units</u> within the memory. For example, the pointer shown as going to PERSON would actually point to another unit. However, since this other unit will also be one meaning of the word "person," we can refer to it as PERSON, rather than showing its actual form. Each word shown in Figure 3 in all capitals should be understood as standing for some unit; in the actual memory there are no such words, only units and properties. Hereafter in this report, words in all capitals will be only used to represent units. Actual English words are stored outside the memory proper, in a "dictionary." Each word in this dictionary is associated with one or more pointers to units in the memory, each of which represents one of the word's meanings.

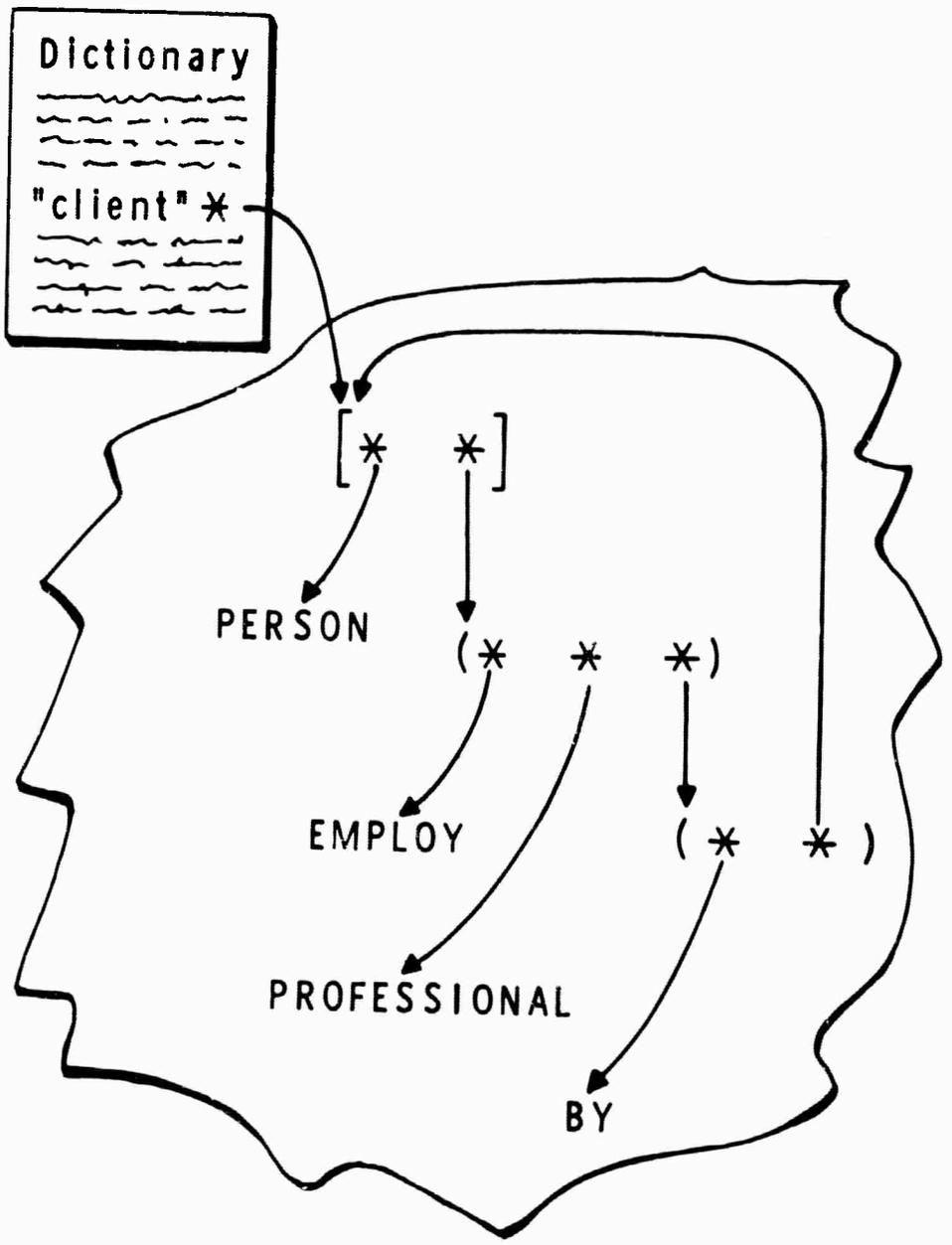


Fig. 3 A Piece of Information in Memory

In Fig. 3 the word "client" is associated with one pointer to a unit, shown as delimited by two square brackets. Also shown in Fig. 3 are two properties, each of which is delimited by parentheses.

Any unit's first element (reading left to right) must always be a pointer to some other unit, referred to as that 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 unit 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. 3 shows the superset of the unit representing "client" to be PERSON, followed by one pointer to a property.

All properties pointed to in a unit represent predicates which, when associated with that unit's superset, comprise the concept the unit represents. In other words, a concept is always represented in our format by pointing to some generic unit, its superset, of which it can be considered a special instance, and then pointing to properties stating how that superset must be modified in order to constitute the concept intended. Properties are therefore the means by which refining modifications of a superset are encoded.

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 also 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 new 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 TLC's building up of new structures to represent the meaning of text it comprehends.

A property is always essentially an attribute-value pair, but in the memory format this notion has been broadened well beyond its usual usage. That is, not only traditional dimensions and values such as (color white) are encoded as attribute-value pairs, but also any preposition and its object, and any verb and its (direct) object. Thus, a property corresponding to (on hill) could be encoded, as can the property in Fig. 3, (employ professional ...). In these latter cases the relational concept - the preposition or verb - is used as the property's attribute, with what would usually be its grammatical object serving as its value. This broadening of the notion of an attribute-value pair picks out the common feature "relational," from the many diverse ways that this feature is stated or implied in English, to always represent it uniformly. It is a major step toward the goal of this format, uniformity without loss of expressive power.

The first element of any property must always be a pointer to its attribute, and its second element must always be a pointer to its value. These two obligatory elements are followed optionally by any number of pointers to other properties. Like the properties helping to comprise a unit, these properties 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. All this is illustrated in Fig. 3: First, the unit representing the memory's concept of client has one refining property. The attribute and value of this property assert simply that some professional is employed. However, a refining sub-property of this property further specifies that this employing is done by this client himself, since the value of the attribute BY is a pointer back to the unit representing the concept CLIENT. In total, then, Fig. 3 simply represents a concept to the effect that a client is a person who employs a professional.

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. To further extend the format's expressive power, space is available in each unit (but not shown here) for storing quantifier-like modifications of it, and for allowing a unit to represent some set of other units, bound together by AND, by INCLUSIVE-OR, or by EXCLUSIVE-OR. This allows units to be created which represent concepts of individual things, of groups or classes of things, of substances, etc. This space in the unit also contains a pointer to the English word(s) associated with the unit, if such exist.

B. The Overall Organization Of Factual Information

In replacing units by capitalized words in Fig. 3, we not only made the structure more readable, but also cut off the unlimited interlinking to other units which characterizes the actual memory. Since all units and properties are made up of pointers to other units and properties, the overall memory is a large network. Many units and properties in this network will contain pointers to the same other units or properties. In fact, all the units which use a particular concept as a compositional ingredient should contain a pointer to the same unit, so that no more than one unit will ever be required in the entire memory to represent explicitly any particular concept. If two units use the same concept but with different modifications of it, then each of them will point to separate intermediate units, whose supersets will be 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 the same as that used in an earlier program (Quillian, Op Cit). It is based on the fact that, starting from any given unit in the memory, a program can easily trace to all the units that this unit contains pointers to, and then (on a second pass) to all the units these units 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 greater and greater numbers of units, even though certain branches of the fan will either circle back to previous units, or will simply die off due to reaching some NIL unit, 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 unit it passes through. This activation tag names the initial starting concept which led (however indirectly) to all the units reached and tagged.

Now, suppose that this process is initially given more than one initial starting unit. 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 unit it reaches, identifying the particular fan it is a part of by naming the initial unit at the fan's head. Moreover, this process now checks every unit it tags, to see if the unit has already been reached during prior tracing eminating from some other initial unit. This is easily determined, since any such unit will have a tag showing it has already been reached, indicating its initial unit(s). Whenever such a previously tagged unit is found, it constitutes an ingredient common to these two initial units, an "intersection."

This method of locating common ingredients of concepts will, in general, find the common ingredients which are closest to the initial starting units 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 TLC, with more elaborate tags which allow the program to distinguish an intersection unit that is connected to an initial unit by a path going only through supersets from one whose path at some point moves "out" through an attribute or value of some property.

With this general picture of the memory we are ready to plunge into the considerably more difficult process of how TLC comprehends text.

III. HOW TLC WORKS

A. Finding Memory Properties Related To The Text

Natural language text communicates by causing a reader to recall mental concepts which he already has. It refers him to such already known concepts either with isolated words or with short phrases, and then specifies or implies particular relations between these. In this way the text may be said to direct the reader to form new concepts. These new concepts contain representations of the old ones, along with representations of the various relations asserted between these known concepts. We assume that such new concepts are formed at least temporarily within the head of a reader to represent his comprehension.

Therefore, TLC's plan for encoding the meaning of text is to retrieve from its memory those units that represent the concepts discussed by a text, and then create a separate new unit to "include" each of these as its superset. While the superset of each new unit will thus be a previously known unit, the new unit's refining properties must be built to represent whatever relations the text implies between this unit and others, i.e., whatever particular assertions the text makes about it. Fig. 4A indicates two initial steps TLC takes toward achieving such comprehension of an input phrase.

For each word of the input phrase TLC creates a new unit, shown in Fig. 4A as delimited by a pair of brackets still empty of content. TLC will try to add into these new units pointers to appropriate

Fig. 4A. Initial Steps

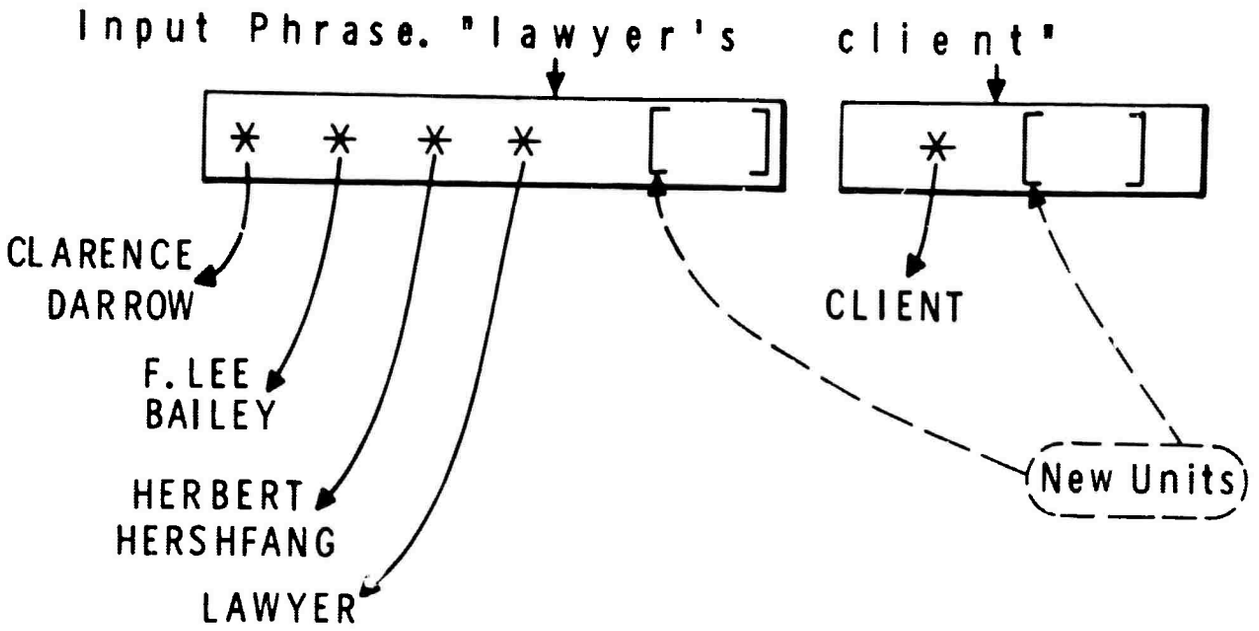


Fig. 4B. Output of Comprehension

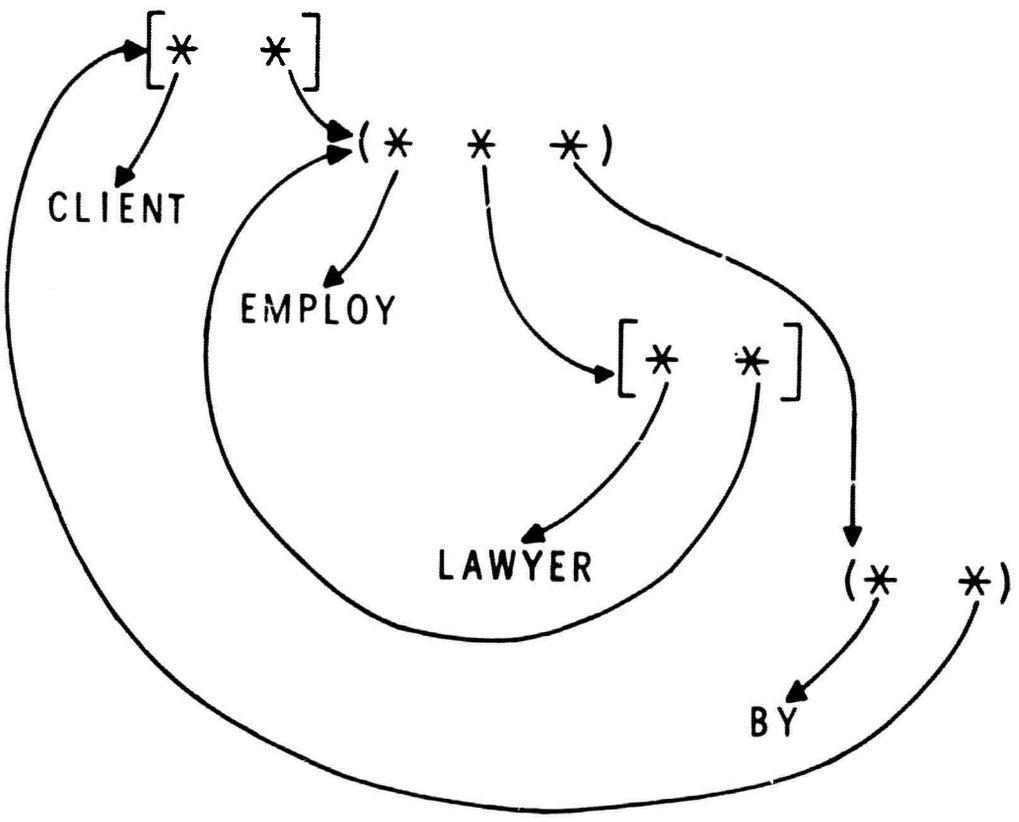


FIG. 4 Stages in the Comprehension of ". . . lawyer's client. . ."

supersets and properties, so as to compose a correct representation of what old concepts they refer to and what is asserted about these. Thus in Fig. 4B, which shows the final output of the comprehension process, these same two square bracketed units appear again, but now "filled in" with pointers to supersets and properties.

When such new units are initially set up, three of the major obstacles that stand in the way of filling them in are:

1. Words usually have multiple meanings; how is one (or some set) of these to be selected to serve as superset of the word's new unit? (In other words, precisely which old concepts are referred to in this piece of text?)

2. How is TLC to compose properties adequate to express what the text asserts about these concepts?

3. In continuous text a great many words refer to things discussed earlier in the same text; how is TLC to know when this occurs, so that it can somehow amalgamate all the statements made about some given thing throughout a text? (For example, in the Fig. 4A text "lawyer" might or might not refer to some particular lawyer the text has been discussing. Such previous occurrences of something referred to later in text are called "anaphoric" or "generalized pronominal" referents of the later reference.)

TLC's strategy is to combine all three of these obstacles, and attempt to overcome them all as part of a single process. This process is its detailed theory of language comprehension.

The program begins by setting up for each word of the text it is working on not only the word's initially empty new unit but also a list of pointers. These pointers enumerate all the candidates for that word's meaning; there is one pointer to each of the word's possible dictionary definitions and one pointer to each anaphoric referent the word might possibly have in this text. In

Fig. 4A these lists of candidates are indicated as lists of asterisks; we have assumed that three different lawyers have been discussed previously in the text, and since there is also one dictionary meaning of the word, four pointers appear in its candidate list. The program must select one of each word's candidates to fill in as superset of that word's new unit; this will constitute its decision about that word's present meaning, including whether or not this meaning is anaphoric.

A word's possible anaphoric referents, having previously been comprehended, will now each be represented by the unit created to represent it, and hence will be of the same format as a dictionary definition. Thus, TLC can process any candidate unit uniformly, whether it is an anaphoric referent or a standard dictionary meaning. Recall especially that an unlimited number of properties can be reached by tracing through the fan of data emanating from any given unit; this is true of newly filled in anaphoric units as well as of units serving as permanent dictionary definitions, since both are made up of pointers to other units in memory.

Now, TLC must somehow compose properties to add to newly created units. It does this by adapting copies of properties already in its memory, copies being made of each property that it can find that relates to the current text. To decide which properties in memory a piece of text relates to, however, TLC first needs to know which properties from its memory to consider, and in what order. Its rule for selecting such properties is simply: consider properties associated with the candidates of its words, and consider these properties in breadth-first order. More specifically, TLC picks its way across the candidate units of words in the piece of text, investigating one at a time the properties pointed to in these units. For each property it makes a very quick check for intersection tags to see whether or not the property may be one that is related to the current sentence. Usually this quick

check will not indicate that the property can be related to the current piece of text, so the program will leave tags on the property and pass on to consider another property. These tags will allow subsequent processing to discover any intersection with this property, in which case TLC will return to consider it again. Whenever an intersection is found, either in the initial quick check or later, a more thorough investigation of this possibility is initiated.

Let us first consider what happens whenever these processes do in fact indicate that some property is related to the current text. Finding a property that is so related has two implications. First, it provides a memory property to recopy and adapt in order to represent part of the text's meaning. Second, since every property investigated is associated with some candidate, it implies that this candidate is the appropriate one to select as its word's meaning. For instance, if in Fig. 4A the first related property TLC finds is stored with the second candidate for "lawyer," the program will assume that that candidate is the appropriate meaning for the word "lawyer," and set up a pointer to it as superset of the new unit representing "lawyer."

Thus, finding properties that relate to input text simultaneously provides TLC's solution to the problems of multiple meaning, of anaphoric reference, and of what properties to copy and adapt in order to encode what the input says. We shall see below that it also tells TLC how to adapt each such property. All these results, however, hinge on TLC's ability to locate particular properties that are related to a given piece of text. To illustrate how the program does this, we will again use Fig. 3. (Notice first that the candidate set up for "client" in Fig. 4A must in fact be the unit illustrated in Fig. 3.)

In trying to comprehend "lawyer's client," TLC goes to the Fig. 3 unit, on which it finds one property (and one nested sub-property) to try to relate to this input phrase. In its quick check of these, the program first considers the property's attribute, EMPLOY, and seeks to discover any word in the input phrase that seems acceptable to identify with EMPLOY. Such a word might be "employ" itself, or it could be "hire," "work (for)," "engage," etc. (Description of the intersection technique TLC uses to see if the input contains such a word is again best postponed until we understand the role of such identification in TLC's overall scheme; it will be described in Section IIIC below.)

In this example the input phrase, "lawyer's client," contains no word that TLC can identify with EMPLOY. The program therefore turns from the attribute of the property under investigation to its value, PROFESSIONAL. It checks this unit in the same way, attempting to locate any word of the input string that seems acceptable to identify with PROFESSIONAL. This time (since a lawyer is defined in the memory as one particular kind of professional) TLC decides that the word "lawyer" is tentatively acceptable to identify with PROFESSIONAL.

Now, TLC takes this tentative identification of part of a property with some word of the text as an indication that the text may imply a particular relation between two words of the text: the identified text word and the word whose candidate contains the property. That is, the text may in part mean that the identified word, "lawyer," relates to the "source" word, "client," in the way this property, (EMPLOY PROFESSIONAL ...), would indicate if "lawyer" were identified as the professional. However, at this point this is only tentative; the program only knows that "client" and "lawyer" both appear somewhere in the input piece of text; it has not yet considered the syntactic arrangement or inflections of these two words, nor what other words appear between or around them.

These features of the local context will determine whether the input implies the kind of relationship named by the property, or whether the two words just happened to appear in the same piece of text, but with no intention of relating their meanings as indicated by the property under consideration. To decide this question entails deciphering the implications of particular syntactic (and semantic) features.

As stated above the program's syntactic recognition ability is composed of many separate form tests stored in the memory. A form test is a routine which checks for the presence of particular features in the input phrase. It also specifies how to use the property under investigation if these particular features are present. The features checked in various form tests include that "client" come before "lawyer," that a particular word or symbol (such as "of") appears between the two words, that one of the words must have some particular kind of ending (e.g.'s), or that some agreement of endings must obtain between them or between them and certain words around them. A form test of course states its features in terms of the variables "source word" and "identified word," not in terms of "client" and "lawyer" per se.

To make it easier for the person serving as TLC's monitor to specify form tests, the system provides a language, somewhat resembling a string manipulation language like COMIT, FLIP, or SNOBOL (on these see Raphael, Bobrow, Fein and Young, 1968). The job of TLC's monitor is to specify such form tests as they are needed, and to associate such tests with the correct unit used as an attribute in the memory. Whenever the program finds itself able to identify the attribute or value of some property with some word in the text, it retrieves all the form

tests associated with that property's attribute.[2] It then runs these form test routines one at a time on the piece of text it is currently trying to comprehend. These form tests continue until either one test is found whose features are present in the current piece of input text, or all tests in the set have failed. In the latter case the program concludes that this property should not be used to relate the two tentatively related words after all, and goes back to looking for other identifications of the property's attribute or value.

In the present example, one form test stored with EMPLOY specifies that the word tentatively identified with the property's value ("lawyer") have an 's as its ending, and be followed immediately by the source word ("client"). Since this is true in this input phrase, this form test will succeed, leading the program to conclude that, indeed, this property is related to the input text.

TLC next checks any property in memory that modifies the current property (in this case that starting with a pointer to BY, see Fig. 3), to see if this sub-property also can be related to the current text. In this it again succeeds, although in this case by identifying the source word, "client," with the sub-property's value.

At this point the program has succeeded in relating the input piece of text to one property (and to its nested sub-property), and is ready to use the information this provides.

---

2 Actually, every property's attribute has two sets of recognition capabilities associated with it, one to be tried whenever the attribute itself is tentatively identified with some word of the sentence; the other to be tried if the property's value is so identified, as in our current example. Only one of these sets is tried, since different form tests are appropriate in the two cases.

B. The Encoding Process

The property TLC has decided is related to the text was part of one candidate of "client." TLC assigns this candidate to be the superset of that word's new unit. (This settles on a meaning for "client," and would be a selection if there had been any other candidates.) It similarly assigns as the meaning of "lawyer" one of "lawyer's" candidate units, namely, the one the intersection program identified with PROFESSIONAL.[3] The program also revises the candidate lists for both of the related words, so that further passes looking for properties related to the current text will consider none of their candidates except the two now thought to constitute the correct meanings. (However, this restriction has a time limit, and so will disappear if the current decisions lead into a dead end for comprehending the rest of the sentence.) The program now copies over and adapts the related property and subproperty with the result shown in Fig. 4B.

As stated above, the topmost unit shown in Fig. 4B is the same new unit shown in Fig. 4A for "client;" the other bracketed unit in Fig. 4B is the new unit shown in Fig. 4A for "lawyer." In Fig. 4B these two new units have each been filled in with a pointer to

---

3 Thus, we see that the use of a property may allow TLC to select the appropriate meanings of two words of the sentence, a detail omitted for the sake of clarity in the overview of its operation above. One might argue that more than one meaning of "lawyer" can be identified with PROFESSIONAL, so that actually the set of these should be made its superset. One counterargument to this is that people in reading seem to pick one sensible meaning for a word, and ignore others unless this leads to some anomaly. (Another is that picking one has been easier to program.)

the candidate chosen as its superset and to the new property created by copying over the EMPLOY property of Fig. 3. This copy has been built using TLC's general rule for adapting memory properties: copy each property exactly but replace any pointer to a unit that has been identified with some word of the text with a pointer to the new unit for that word. Thus, the value of the EMPLOY property in Fig. 4B is a pointer not to PROFESSIONAL, but to the new unit for "lawyer," since these have been identified. Similarly, the value of the BY property is a pointer not to the memory's general unit representing a client, but to the new unit representing this client.

Overall, the data of Fig. 4B is the same as that shown (in more readable form)[4] in Fig. 1A, which TLC generates into English as: "Under discussion is a client who employs a lawyer." If, during the process of locating related properties, another distinct related property is found, a pointer to the adapted copy of this property is also added onto the new units for "client" and "lawyer" (See Fig. 1B.)

In summary, Fig. 4B shows that TLC draws information from its memory to produce a representation of a piece of input text that is:

1) Encoded in the regular memory format.
2) Much richer and less ambiguous than the input text itself.
3) A highly intraconnected structure with the various concepts mentioned in the text linked in ways supplied from a memory of facts about the world.
4) Linked to the permanent memory by many pointers to established concepts (although its creation in no way changes the permanent memory, except for the addition of temporary tags used by the intersection routines.)

---

4 In its OUTPUT1, the program replaces any repeated mention of a unit with a (*THIS* - name), and omits any repeated mention of a property.

C. Identifying Units by Information in Memory

The process described above depends on TLC being able to identify units such as PROFESSIONAL with candidate units of words in a piece of text. As stated, such an identification provides a property tentatively related to the text, and initiates syntactic checking via form tests.

The first condition that two units must meet to be identifiable is that they have what we call a "superset intersection in memory." This is said to occur if either:

1) The two units are the same unit.
2) One unit is superset of the other, or superset of the superset of the other, or superset of that, etc. In this case, we will say that one unit lies directly on the "superset chain" of the other.
3) The two units' superset chains merge at some point.

TLC is designed to locate superset intersections which occur between certain kinds of units but to ignore those which occur between others. To see the reasons for this let us consider the sentence, "John shoots the teacher." This sentence has one interpretation if John is known to be a gangster, and quite another if John is known to be a portrait photographer. TLC's problem is to make sure that, whichever of these is the case, this knowledge has the same effect on TLC that it has on a human reader. Let us suppose the second is the case, so that the unit representing "John" in the comprehender's memory has a property stating, in part: (PHOTOGRAPH SUBJECT(S) ...). The situation is indicated in Fig. 5, which shows one candidate each for "John" and for "teacher," and one property of each of these units in memory.

Now, superset intersections connecting three separate kinds of pairs of units can be picked out in Fig. 5. All of these intersect in the third way mentioned above by merging of superset chains. First, there is an intersection between the candidate units JOHN

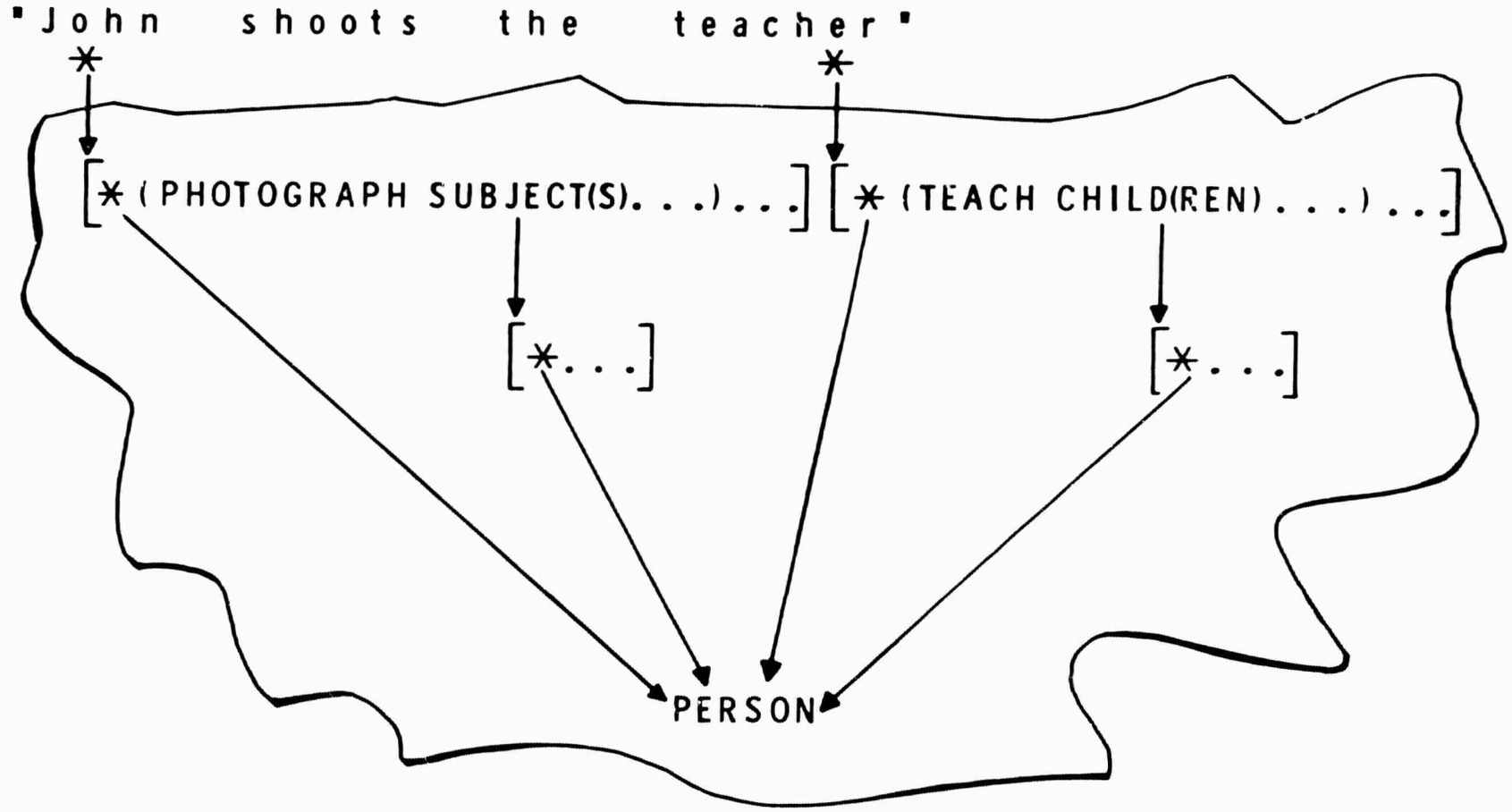


FIG.5 Three Kinds of Pairs of Intersecting Units

and TEACHER, since their superset chains merge at the unit PERSON. Clearly this intersection is not particularly useful for comprehension, however, since it does not involve any properties of either JOHN or of TEACHER. Second, there is a superset intersection between SUBJECT(S) and CHILD(REN), whose superset chains also merge at PERSON. This intersection connects part of a property of JOHN with part of a property of TEACHER. In other words, the comprehender knows something about JOHN and knows something else about TEACHER, and these pieces of his knowledge relate JOHN and TEACHER to two other things, and these two other things are semantically acceptable to identify. Such intersections, connecting parts of separate properties, are very common in the memory, but, we suspect, are not of much use in direct comprehension, TLC's only aim at present. We will indicate below how intersections connecting properties may be used in indirect comprehension, but, for the time being, TLC is designed to ignore them. The third kind of pair having a superset intersection in Fig. 5 is the mix of the first two: part of some property is connected to a candidate. Thus, the superset chain of SUBJECTS(S) merges with that of JOHN at PERSON, as does the superset chain of CHILD(REN) with that of TEACHER, of SUBJECT(S) with TEACHER, and of CHILD(REN) with JOHN. The first two of these pairs each connect part of a property to its own parent unit. Such an intersection also is of no interest, and is eliminated by TLC (except in the case of explicit self reference, as in the BY property of Fig. 3.)

The two remaining intersection pairs, connecting part of a property with some other candidate, are the kinds of intersections TLC is interested in, these being the sort which cause it to create hypotheses about the input's meaning and then to initiate form tests, as described above. In this example, if and when TLC finds the intersection between JOHN and CHILD(REN), its form tests must not succeed, so the program will reject the hypothesis that this sentence means John is being taught.

When TLC finds the intersection between SUBJECT(S) and TEACHER however, a form test should succeed, and lead to interpretation of the input sentence. This form test has to be able to recognize that one meaning of "shoot" can be identified with PHOTOGRAPH, by intersection of supersets. The interpretation would be to the effect that: "under discussion is the photographing of this teacher, done by John, ...". If John were known to be a gangster instead of a photographer, the memory would contain different information and TLC would interpret the sentence differently; but to do so it would still need to locate intersections which connect part of a property with some other candidate, but not to locate intersections which connect two candidates or parts of two properties.

In order to locate such "mixed" intersections, TLC uses a version of its intersection technique in which it puts one kind of activation tag on candidates and their supersets, and a different kind on parts of properties and their supersets. It then always checks as it marks a unit with one kind of tag to see if that unit already has been marked with the other kind of tag. Thus the first time TLC reaches a unit which is in fact a pertinent intersection it will mark that unit, and the first time after that that it reaches it as the other kind of thing - candidate or part of a property - it will recognize the unit to be a relevant intersection.

However, it is also very important that TLC locate intersections in something close to a breadth-first order. This means that the program must appropriately alternate between considering parts of properties and their supersets, and considering candidates and their supersets. Therefore, TLC processes an input piece of text in a number of passes, as follows:

At the time units are first chosen as candidates for the words in a piece of input text, these units are tagged. Then the program makes its first pass looking for properties related to the text. To do this it takes these same candidate units one at a time, and "quick-checks" all the properties associated with each. To

quick-check any one property is to mark and look for prior tags on the attribute and value of the property, and on three levels of superset of each of these units. This procedure is interrupted at any point that an intersection is found.

After all the properties of some candidate unit have been quick-checked, the program moves briefly to the superset of that candidate. It marks this superset unit to show it is in the superset chain of the candidate, and also checks to see if the unit already has a tag indicating it is also part of some relevant property, or in the superset chain of such a part. Of course, for an intersection to be found by this "checkback" step, the property involved must already have been quick-checked earlier in the processing of this piece of text. If an intersection is found by this checkback, the program returns to the property involved and considers using it exactly as it would have done if the quick-check of that property had found the intersection.

After the checkback, the program moves on to consider another candidate. If a second pass through a piece of text is necessary, TLC will quick-check properties associated with the supersets of all candidate units. Similarly, on a third pass it will quick-check properties associated with the supersets of supersets of candidates, and so on. On each step of such a pass, the program quick-checks only the properties directly associated with the unit that it has at hand, and then marks and checks back the superset of that unit to see if it intersects with any properties already quick-checked.

Overall, the program's procedure locates the relevant kind of intersecting pairs of units in an order reasonably close to that which a completely breadth-first search would produce, while ignoring all the irrelevant kinds of intersecting pairs, and minimizing the information it must hold in short term memory

during searching. Our guess is that a human's version of this search is done largely in parallel, which a breadth-first seach simulates reasonably well, until parallel searching machines become available.

At any time during this process that TLC does locate two units which have a superset intersection, either the units are identical in meaning, one is a superset of the other or both are members of some common superset; these correspond to the three possible kinds of superset intersection enumerated at the start of this section. In the first two cases TLC immediately reports the two units as semantically acceptable to identify. In the last case it checks to see if there is anything contradictory between the meanings of the two units; only if there are no contradictory properties are the two units accepted as semantically identifiable.[5]

One more important point about TLC's search for properties related to a given text is that there are a good many units in the memory which represent sets. TLC may therefore encounter such a set as the superset of some unit, or as the attribute or value of some property. Also, the multiple candidates set up for a word themselves comprise a set. In all these cases, the program first orders the members of the set so that the most pertinent members come first. That is, possible anaphoric referents are always put first, other units which are "in context" for the complete text of which the current input is a part are put next, and any other units are put last. (TLC recognizes units which

5 Roughly speaking, two units are judged contradictory only if they are found to each contain a property having the same attribute but different values. Thus, "woman," "Spaniard," "amateur," or "infant," should all intersect with PROFESSIONAL at the unit PERSON, but only the first two of these should be identifiable with it, since the latter two each conflict in one regard or another with a property of PROFESSIONAL. The routines to check for such contradiction work by another version of the tagging and intersection finding technique.

are "in context," by the presence of old activation tags left in the memory during processing of prior sentences of the same text.) This ordering of the members of sets often produces an order different from the a priori likelihood ordering of sets as they are stored in memory. It is done to insure that units expected to be most pertinent in this context will be searched slightly before others.

D. Automatic Generalization of Data and of Form Tests in TLC's Memory

The most important thing about TLC's procedure for locating identifiable units in memory is that it identifies a great many pairs of units. Thus, a node like PROFESSIONAL will be identified with many words besides obvious ones like "lawyer" or "accountant." In fact, any word referring to any kind of person will share the superset PERSON with PROFESSIONAL, and the great majority of these will prove semantically acceptable to identify with it.

Thus, the EMPLOY property shown in Fig. 3 and related to "lawyer's client" will be related in a similar way to almost any input phrase referring to some person's client; given "accountant's client" or "woman's client" the accountant or the woman will be identified with PROFESSIONAL, and hence the phrase comprehended as meaning that that person is employed by the client.

A similar effect results from the fact that TLC's first pass investigates the properties directly pointed to in candidate units, its second pass investigates the properties directly pointed to in supersets of those units, and so on.

The importance of this can be seen if TLC is given a phrase such as "young client." Correct comprehension of this phrase must supply the fact that this client's age is being judged young, which is not explicit in the text. TLC's way of supplying such information is by relating the text to a stored property having AGE as an attribute, and having a set containing YOUNG, OLD, MIDDLE-AGED, etc. as a value. However, it does not seem reasonable to suppose that TLC would have any such property stored with its unit for "client." On the other hand it is not unreasonable to suppose that such a property would be part of the memory's unit representing

PERSON, a concept for which age is very general and important. If such a property is stored with PERSON, TLC's breadth-first search through candidates will reach and relate it to "young client" on its second pass. (See Fig. 6, example 1)

Similarly, given "young Spaniard," or "middle-aged carpenter" or any such phrase mentioning some sort of person, TLC will reach and use that same AGE property stored with PERSON.

Now, the important point both about TLC's successive passes and about its ability to identify a great many pairs of units in its memory is that they allow TLC to use its memory in an inferential as well as a literal manner. Thus in comprehending "lawyer's client," the program implicitly made an inference: since a client employs a professional and a lawyer can be identified as a professional, a client can employ a lawyer. Similarly, in comprehending "young client," the inference TLC made is: since a person's age can be young, and a client is a person, a client's age can be young. Being able to use its memory inferentially means that any single property in the memory can be used by TLC to help comprehend a great many different input phrases. In other words, a relatively small memory with an inferential capability is the functional equivalent of a very much larger store of information. The advantages of this seem obvious; we propose that humans similarly use their memories inferentially, and hence are able to operate as if they knew a vastly greater amount than they in fact need to have stored. (Psychological data in Collins and Quillian, 1968 and 1969, bears directly on this hypothesis.) The two types of inferences currently made by TLC do not exhaust all the valid inferences that can be made on the basis of paths through such memories, although they do provide a good start (Longyear, 1968, discusses some others.)

The program's ability to use its memory inferentially is not to be confused with its ability to recognize various ways in which any given assertion may be stated, an ability which depends on its form tests rather than on its memory for factual information. However, these abilities are similar in regard to their generalization. That is, from the point of view of a monitor, the effect of the program's ability to use a single property in comprehending many different sentences is that, having expressed some fact in the form of a property and included this property in the memory in order to enable comprehension of one piece of text, he will not have to put the same fact in again when it is needed in the comprehension of some other piece of text. This same kind of generalization occurs when he adds a form test.

To see this generalizing effect let us imagine that an EMPLOY property similar to that of Fig. 3 is added to the data comprising the concept LAWYER. This new property states that the client that a lawyer represents or advises usually employs that lawyer. Now, suppose the program is given some example input, say, "lawyer for the client," which it previously comprehended as shown in example 13 of figure 6. TLC now locates intersections connecting words of the sentence to parts of the newly-added property and its sub-property, just as it did in comprehending "lawyer's client." However, since appropriate form tests are not associated with the attribute of this property, the program will decide it cannot really relate this property to this phase, and again give the same output shown in example 13. To correct this, the monitor will associate new form tests with EMPLOY and BY, the attributes involved. These tests must succeed whenever the source word (here "lawyer") is followed by the word "for," which is in turn followed by the word tentatively identified with the value of the sub-property (here "client"). In this case, the monitor does not need to define these new form tests, since appropriate ones have already been defined, and are associated with the property previously stored with LAWYER. Thus, he simply adds these tests to those associated with EMPLOY and BY, and reruns the example. This time TLC also uses the newly-added property, and gives the enriched output shown as example 14 of Fig. 6.

However, the important thing is that a new form test will have been associated with the unit EMPLOY. Thus whenever, in a future attempt to comprehend some other phrase, any other property having EMPLOY as an attribute is investigated, the newly added form test will again be available, with no intervention by the monitor required. For instance, if the memory contains properties stating that agents are employed by actors and that bookkeepers are employed by companies, the form test just added will provide the syntactic capability TLC needs to comprehend input phrases such as "agent for Marlon Brando," or "accountant for Bolt Beranek and Newman," by relating each to the appropriate EMPLOY property. These outputs are shown as examples 15 and 16 of Fig. 6.

In other words, TLC in effect automatically generalizes each form test it is given to all properties having the same attribute. This goes some way toward generalizing each capability given to TLC, although obviously not far enough, since each new form test should in fact be generalized to other attributes which are somehow "of the same sort." TLC allows the monitor to let a set of form tests associated with one attribute also serve as the set for any number of other attributes, although we haven't yet had enough experience as a monitor to say how effectively this potential can be used.

E. TLC And Complex Sentence Structure

In order to deal with strings of text longer than very simple phrases, TLC must employ some combination of properties, and hence some combination of its form tests, much as some combination of rules in a grammar must be employed to generate any real sentence. TLC employs several tactics not so far discussed to put together appropriate combinations of properties and form tests. These can be illustrated in its comprehension of a phrase like "lawyer's young client."

To comprehend this phrase, the program again sets up new units and candidate lists as shown in Fig. 4 for "lawyer's client." On its first pass it again investigates the Fig. 3 property, and tentatively identifies PROFESSIONAL with "lawyer." However in this case the form test that succeeded for "lawyer's client" will not succeed, since the interposed word "young" intrudes into the pattern that that test requires.

Now, whenever a form test fails, TLC checks to see if this is only because of unwanted words interposed in the input string. If so, it considers the current property as "pending" some use of the interposed word(s) and goes on attempting to find other properties to relate to the input. In this case, then, TLC holds the Fig. 3 property pending use of the word "young," and continues investigating properties. On its second pass through the candidates, it will come to the AGE property stored with PERSON, and use this f r comprehending the "young client" part of the input phrase.

At this point it needs to be recalled that each form test consists both of some pattern of features which must be found in the input piece of text and of specifications for what to do if this is found. Among other things, these specifications state which word is to be considered syntactic "head" of the words mentioned in the form test,

if any is. For instance, for 'lawyer's client," the form tests that succeed specify that "client" must be the head of that input phrase, as is indicated in the English output of Fig. 1 when TLC says, "Under discussion is a client ..." (compare other outputs in Fig. 6.) Similarly, the form test that succeeds for "young client" specifies that "client" must be the head of that input. The significance of choosing a head is that, whenever any property is related to a piece of text, TLC marks all the words matched by the successful form test as "used" except that which the test specifies as head. This means that in the present example "young" now gets marked "used," and the Fig. 3 property, held pending use of that word, is tried again. Since form tests know they can if necessary skip over any words that have been marked "used," this form test now succeeds, allowing the pending property to be related appropriately to the input phrase and comprehended as shown in Fig. 6 as example 2.

Several properties of TLC's operation can be seen in the preceding example. First, holding properties pending allows the program to adapt the order in which these properties are recovered from memory into an order more appropriate for the syntax of a particular piece of input text. Second, the specification of a head by each successful form test allows the program to nest its processing of a sentence so as to "eat up" long sentences a little at a time, amalgamating subordinate constituents into larger and larger constituents until only a single head remains unused within the sentence. Notice that as this processing proceeds, the new units created to represent words of the input string are being filled in with new adapted properties whose values (or attributes) are pointers to the new units representing other words, in accord with TLC's general rule for adapting copied over properties. (See Section IIIB.

Figure 6: Sample PLC Comprehensions

Key

The first eleven examples were run in normal mode. When the program is run in a more closely monitored mode, as in example 12, it prints out two lines of information each time it uses a property to help comprehend the input. This output always names what it will print out, followed by a colon, followed by the information named. The meaning of the names used are as follows:

USING: the attribute and value of the data property it is currently using.

ATR*: a word in the input which it has identified with the attribute of the data property.

VAL*: a word in the input which it has identified with the value of the data property.

SOURCE: the word of the input whose meaning provided the data property.

PER: The form test used. Form tests always are named T1, T2, . . ., Tn. Any words preceeding the form test name describe how it was used: ATRIB means it was used because the property's attribute was intersected; CKBACK means the intersection occurred during a "check back"; NESTED means the property has been held pending before use.

HEAD: The word chosen as the syntactic head of the words currently used.

NOW-CAN-USE: This is used in place of USING if a property's use has been dependent on the use of one of its sub-properties.

Example No.

1. READ(YOUNG CLIENT)

((CLIENT (AGE (YOUNG))))

AT THIS POINT WE ARE DISCUSSING A YOUNG CLIENT.

(continued)

Example No. (continued)

```
2.  READ(THE LAWYER 'S YOUNG CLIENT)

  ((CLIENT (AGE (YOUNG))
      (EMPLOY (LAWYER)
        (BY (*THIS* . CLIENT)))))
```

NOW WE ARE TALKING ABOUT A YOUNG CLIENT; HE IS A CLIENT WHO EMPLOYS A LAWYER.

```
3.  READ(CLIENT 'S LAWYER)

  ((LAWYER ((AOR REPRESENT ADVISE)
        (CLIENT)
        (BY (*THIS* . LAWYER))
        (IN (MATTER (TYPE LEGAL))))))
```

HERE WE ARE CONCERNED WITH A LAWYER WHO REPRESENTS OR ADVISES A CLIENT IN A LEGAL MATTER.

```
4.  READ(MAN 'S LAWYER)

  ((LAWYER ((AOR REPRESENT ADVISE)
        (MAN)
        (BY (*THIS* . LAWYER))
        (IN (MATTER (TYPE LEGAL))))))
```

AT THIS POINT WE ARE DISCUSSING A LAWYER WHO REPRESENTS OR ADVISES A MAN IN A LEGAL MATTER.

```
5.  READ(DOCTOR 'S LAWYER)

  ((LAWYER ((AOR REPRESENT ADVISE)
        (DOCTOR)
        (BY (*THIS* . LAWYER))
        (IN (MATTER (TYPE LEGAL))))))
```

NOW WE ARE TALKING ABOUT A LAWYER WHO REPRESENTS OR ADVISES A DOCTOR IN A LEGAL MATTER.

```
6.  READ(LAWYER 'S DOCTOR)

  ((DOCTOR (CURE (LAWYER)
        (BY (*THIS* . DOCTOR)))))
```

HERE WE ARE CONCERNED WITH A DOCTOR WHO CURES A LAWYER

.

Example No. (continued)

7. READ(LAWYER OF THE CLIENT)

```
((LAWYER ((AO? REPRESENT ADVISE)
          (CLIENT:
          (BY (*THIS* . LAWYER))
          (IN (MATTER (TYPE LEGAL))))))
```

AT THIS POINT WE ARE DISCUSSING A LAWYER WHO REPRESENTS OR ADVISES A CLIENT IN A LEGAL MATTER.

8. READ(LAWYER 'S REPRESENT ATION OF THE CLIENT)

```
((REPRESENT ((*THIS* . REPRESENT)
             (CLIENT)
             (BY (LAWYER))
             (IN (MATTER (TYPE LEGAL))))))
```

NOW WE ARE TALKING ABOUT THE REPRESENTING OF A CLIENT BY A LAWYER IN A LEGAL MATTER.

9. READ(THE CLIENT ADVISE ED BY THE LAWYER)

```
((CLIENT ((ADVISE)
          (*THIS* . CLIENT)
          (BY (LAWYER))
          (IN (MATTER (TYPE LEGAL))))))
```

HERE WE ARE CONCERNED WITH A CLIENT WHO IS ADVISED BY A LAWYER IN A LEGAL MATTER.

10. READ(CLIENT EMPLOY S A LAWYER)

```
((TO ((*THIS* . EMPLOY)
      (LAWYER)
      (BY (CLIENT)))))
```

AT THIS POINT WE ARE DISCUSSING THE EMPLOYING OF A LAWYER BY A CLIENT.

11. READ(THE CLIENT CURE ED BY THE DOCTOR)

```
(((AND CLIENT PATIENT)
   ((CURE)
     (*THIS* . CLIENT)
     (BY (DOCTOR)))))
```

NOW WE ARE TALKING ABOUT A CLIENT, WHO IS A PATIENT, WHO IS CURED B' A DOCTOR.

Example No. (continued)

```
12.  READ (THE CLIENT HEAL ED BY THE DOCTOR EMPLOY S THE
LAWYER)


USING: CURE PATIENT. ATR*: HEAL. VAL*: CLIENT
SOURCE: DOCTOR. PER: ATRIB T29. HEAD: CLIENT


USING: BY DOCTOR. VAL*: DOCTOR
SOURCE: DOCTOR. PER: NESTED T21. HEAD: CLIENT


USING: EMPLOY PROFESSIONAL. ATR*: EMPLOY. VAL*: LAWYER
SOURCE: CLIENT. PER: ATRIB CKBACK T17. HEAD: EMPLOY


USING: BY CLIENT. VAL*: CLIENT
SOURCE: CLIENT. PER: NESTED CKBACK T13. HEAD: EMPLOY

OUTPUT1:
  (EMPLOY ((*THIS* . EMPLOY)
       (LAWYER)
       (BY ((AND CLIENT PATIENT)
            ((HEAL)
              (*THIS* . CLIENT)
              (BY (DOCTOR)))))))

OUTPUT2:

 AT THIS POINT WE ARE DISCUSSING THE EMPLOYING OF A LAWYER
 BY A CLIENT, WHO IS A PATIENT, WHO IS HEALED BY A DOCTOR
.
 13.  READ (LAWYER FOR THE CLIENT)


 USING: BY LAWYER. VAL*: LAWYER
 SOURCE: LAWYER. PER: NESTED T32. HEAD: LAWYER


 NOW-CAN-USE: (AOR REPRESENT ADVISE) CLIENT. VAL*: CLIENT
 SOURCE: LAWYER. PER: NESTED T31. HEAD: LAWYER

 OUTPUT1:
   (LAWYER ((AOR REPRESENT ADVISE)
        (CLIENT)
        (BY (*THIS* . LAWYER))
        (IN (MATTER (TYPE LEGAL)))))

 OUTPUT2:

  NOW WE A   TALKING ABOUT A LAWYER WHO REPRESENTS OR ADVISES
  A CLIENT IN A LEGAL MATTER.
```

Note: For the following examples, we went through the sequence described in section III D, in order to illustrate automatic generalization of the use of form tests. First an EMPLOY property similar to that of Figure 3 was added to the concept LAWYER. Then the form tests needed to let TLC relate this property to "lawyer for the client", namely, T31 & T32, were associated with BY and EMPLOY respectively. Doing this enriches TLC's output over that produced in example 13 for "lawyer for the client." Then, given that the necessary facts are in the memory, TLC comprehends the next two examples with no further intervention by the monitor. Note its use of T31 and T32 as shown in its monitored output for these examples.

Example No. (continued)

14. READ (LAWYER FOR THE CLIENT)

```
USING: EMPLOY LAWYER. VAL*: LAWYER
SOURCE: LAWYER. PER: T32. HEAD: LAWYER


USING: BY CLIENT. VAL*: CLIENT
SOURCE: LAWYER. PER: NESTED T31. HEAD: LAWYER


USING: BY LAWYER. VAL*: LAWYER
SOURCE: LAWYER. PER: NESTED T32. HEAD: LAWYER


NOW-CAN-USE: (AOR REPRESENT ADVISE) CLIENT. VAL*: CLIENT
SOURCE: LAWYER. PER: NESTED T31. HEAD: LAWYER

OUTPUT1:
  (LAWYER ((AOR REPRESENT ADVISE)
      (CLIENT)
      (BY (*THIS* . LAWYER)))
    (EMPLOY (*THIS* . LAWYER)
      (BY (*THIS* . CLIENT))))

OUTPUT2:

 AT THIS POINT WE ARE DISCUSSING A LAWYER WHO REPRESENTS
 OR ADVISES A CLIENT; HE IS A LAWYER WHO IS EMPLOYED BY
 THIS CLIENT.
```

Example No. (continued)

```
15.  READ (AGENT FOR MARLON-BRANDO)


USING: EMPLOY AGENT. VAL*: AGENT
SOURCE: AGENT. PER: T32. HEAD: AGENT


USING: BY (AOR ACTOR FIRM). VAL*: MARLON-BRANDO
SOURCE: AGENT. PER: NESTED CKBACK T31. HEAD: AGENT

OUTPUT1:
  (AGENT (EMPLOY (*THIS* . AGENT)
      (BY (MARLON-BRANDO))))

OUTPUT2:

 NOW WE ARE TALKING ABOUT AN AGENT WHO IS EMPL YED BY MARLON-BRANDO
.


NIL
←RT(E41 15↑56)




16.  READ (ACCOUNTANT FOR BBN)


USING: EMPLOY BOOKKEEPER. VAL*: ACCOUNTANT
SOURCE: ACCOUNTANT. PER: T32. HEAD: ACCOUNTANT


USING: BY ORGANIZATION. VAL*: BBN
SOURCE: ACCOUNTANT. PER: NESTED T31. HEAD: ACCOUNTANT

OUTPUT1:
  (ACCOUNTANT (EMPLOY (*THIS* . ACCOUNTANT)
      (BY (BBN))))

OUTPUT2:

 HERE WE ARE CONCERNED WITH AN ACCOUNTANT WHO IS EMPLOYED
 BY BBN.
```

Therefore, once only one unused head remains in the input string, this means that the new unit created to represent that head contains new properties linking it to other new units, which in turn will in general contain new properties linking to other new units, and so on, in some way such that all of the input string's new units are interlinked in a single network. This structure represents a comprehension which encompasses all the words of this input, and so the program terminates its processing then only a single head remains.

The program's performance for longer sentences is illustrated in Fig. 6, example 12, which TLC comprehends by first amalgamating "healed by the doctor" into the head "client" and then comprehending what amounts to "the client employs the lawyer." In this example, and in the rest of Fig. 6, the output shown is that which TLC produces when it is run in a more closely monitored mode than that shown in earlier figures. That is, in this mode the program shows how it sets up the sentence during the initial comprehension step (showing that articles and endings are amalgamated into their heads immediately) and also prints out information each time that it uses a property, as described in the key to Fig. 6. The rest of Fig. 6 shows most of the other examples TLC has been taught to comprehend so far.

Although examples such as the above show that TLC can comprehend at least some longer sentences and nested constructions, there is still at least one serious flaw in its present use of form tests. No running record is kept of the sentence's syntactic structure as this structure gets deciphered during comprehension. Just how serious the consequences of this are is not clear, but without it, form tests will sometimes check for features which couldn't possibly exist given the sentence's syntactic structure as already understood. A much more serious consequence would occur if form tests, in order to take account of different

contexts in which the features they require appear, had to keep being redefined as longer and longer patterns in order to work properly. We do not really believe that this is a likelyhood, but in any case there is a good way available of keeping such a running syntactic record. This is the method used in "predictive syntactic parsers" (for example, Kuno, 1965.) In view of this, we are now considering incorporating into TLC large parts of one predictive parsing program, that written by Thorne, Bratley, and Dewars (1968).

This extremely interesting and ingenious program[6] does not output as a parsing a tree structure, but rather a set of nested strings. However, in building these strings it succeeds in "undoing" a number of syntactic transformations, replacing deleted elements and rearranging others. Most pertinently, the program is very good at avoiding unnecessary or redundant syntactic processing by keeping a record of what has been decided so far. This record is made during a single pass through an input sentence, in the form of an "analysis network" which records all acceptable parsings of the sentence as paths extending through this network.

At a time when the program's analysis has progressed as far as some given word of the input sentence, all viable analyses extending to that point will be represented by "live" nodes in its analysis network. To proceed on, the program refers to its grammar to obtain all the syntactically allowable next steps from those particular live nodes. Then it tests each such step to see if it actually is acceptable, given the next word and particular prior history of this sentence. All the steps which prove acceptable are recorded as extensions of the analysis network, thereby producing a new set of live nodes for the next stage of the analysis. The analysis

---

6 The author is indebted to Daniel Bobrow for drawing attention to the advantages of Thorne's program for TLC.

network often branches, while previous paths that have no acceptable extensions die off automatically. By only attempting to extend nodes that are live, this procedure automatically restricts syntactic processing to consider exactly and only things which may be possible, given the analyses viable up to the current word. Each path extending to the end of the sentence constitutes a syntactically acceptable parsing of it, and can be printed out as a nested group of strings. Note that steps along such an analysis path are equivalent to particular syntactic relationships between the words at the beginning and end of each step, so that the grammar may be said to propose particular syntactic relationships between words of input sentences.

As we visualize it at this point, TLC would use such a procedure by constructing a Thorne-like analysis network in addition to its comprehension. However, at most stages, instead of extending the analysis network by trying all steps enumerated in the grammar for currently live nodes, TLC's tentative comprehensions would be used to propose extensions of this network. To do this, TLC's form tests would be rewritten to name various steps (possible syntactic relationships) specified in the grammar. That is, instead of naming some pattern of features required in the input string, each form test would simply name a particular syntactic relationship (step) in the grammar, and name particular words of the input string to be related in that way. This would mean that at the point at which TLC now executes form tests, some group of Thorne's syntactic relationships would instead be proposed between the words TLC tentatively relates.

To check out any such proposed syntactic relationship (step) a first check would be made to see if the node that that relationship required as its starting point was "live" at the stage of the analysis network corresponding to the word TLC specified. If not,

that syntactic relationship would be considered impossible (at least at this time). If the beginning node was live, the same kind of checks that Thorne's program makes now of a particular step would be made to see if it was syntactically feasible at this point. If it was, the analysis network would be so extended, and TLC would consider that form test successful.

This kind of use of Thorne's procedure would provide two very desirable features. First, it would allow much faster elimination than at present of the syntactic relationships sought by form tests, since this would use a record of prior processing (is the beginning node live?) rather than having to look each time at features of the input string itself. Second, only syntactic analyses which make sense semantically would be produced. To the degree that the parser only extended the analysis networks via steps proposed by TLC's tentative semantic interpretations, these semantic interpretations would "drive" the syntactic analysis of the input. This would pare away syntactically acceptable but meaningless parsings, which not only are unwanted, but which account for most of the processing activity of present parsers.

The procedure above is not yet programmed, and it is probable that between certain words a purely syntactic analysis such as Thorne's program now makes may have to be made, at least temporarily. This might be the case, for instance, whenever no semantic information is available about a string of unfamiliar words. Also, it might in some cases prove more difficult to specify how to make the parsing step required to establish a syntactic relationship between two words than it would be to generate all syntactically acceptable steps from one to the other and then see if these included any paths having certain characteristics.

However this may turn out, though, incorporation into TLC of a parser like that which Thorne et al have built would seem to offer some attractive possibilities and is being actively explored, with Daniel Bobrow currently duplicating a version of Thorne's program at BBN.

F. Unsolved Problems

In addition to a need for keeping a better record of syntactic processing, TLC at present lacks other capabilities more or less related to its tasks. One such need is for an ability to recognize that many input phrases refer to memory information stored as other, not directly mentioned concepts. For example, "male child" should evoke the comprehender's knowledge of the concept BOY, and "old man" should evoke its knowledge of the concept OLD-MAN, even though, in English, this concept has no single word name. (There is of course no restriction in the memory format against having concepts without English names, and in fact our present memories necessarily include such concepts.)

Another ability lacking in TLC is any ability to reason spatially, or to generate visual-like imagery. Beyond this, TLC is missing capabilities which begin to shade off into things which (we assume) are less directly essential to language comprehension per se. One of these is the ability to assimilate the meaning of a piece of text it comprehends. While the monitor can add TLC's encoded output to the program's memory, the program itself makes no attempt to do so, nor to solve the problems inherent in doing so. One of these problems, for example, is where to store such information. One can see this problem in almost any sentence; is "Battleships in World War I had 16 inch guns," about battleships, about World War I, about guns, or about, perhaps, naval history? Or is it about all of these? The way that this question is answered will determine where the comprehension of the sentence is stored, and hence which words or phrases will be capable of retrieving its information in the future. The question clearly must be answered not so much on the basis of anything present in the input text itself, as on the basis of the overall interests or orientations of the memory. (For possible approaches to this

problem see Abelson and Carroll, 1965, and Tesler, Enea and Colby, 1967.) As of now TLC sets up the properties needed to express what a piece of text means, and then usually adds pointers to these properties to all the new units created to represent words in that fragment of text, as was done in Fig. 4B.

TLC also as yet makes no effort to get rid of most of the new units created during comprehension of text. Such new units represent known concepts plus things said about them in specific instances, and any adequate learning mechanism must forget most such specific instances, while extracting any important generalizations from them and adding these to the more general concepts left in the memory. Some method of achieving such generalization and forgetting must probably be programmed before a significant amount of TLC's memory can actually be built up by reading text.

It was mentioned in the introduction that currently TLC can encounter problems of overinterpretation if too much richness of comprehension is sought. Consider for instance the following phrases, all of which deal with a lawyer and some other person.

1.	2.
enemy's lawyer	lawyer's enemy
wife's lawyer	lawyer's wife
client's lawyer	lawyer's client

Assume that a property stored in memory with the unit LAWYER states that a lawyer has as occupation the representing or advising of person(s) in legal matter(s). Now, in comprehending the three phrases of column 1, it is always appropriate to consider this stored property related to the phrase: "enemy's lawyer" means this lawyer is representing or advising this enemy in a legal matter, "wife's lawyer" means this lawyer is representing

or advising this wife in a legal matter, etc. Furthermore, it appears that no matter what sort of person is substituted for the first word in such a phrase, the use of this stored property in this way remains appropriate.

However, the first two phrases in the second column are not correctly seen as related to this stored property: "lawyer's enemy" does not mean this lawyer is representing or advising this enemy in a legal matter. Moreover, if additional phrases of the column 2 type are generated, very few turn out to be correctly comprehended by use of this particular property.

If given a phrase from either column above to comprehend, TLC's intersection procedures will in all cases locate a superset intersection connecting part of the property stored with LAWYER — specifically, the unit PERSON(s) — with the person named by the other word of the phrase, "enemy," "wife," etc. However, the phrases of column 1 are distinguished from those of column 2 syntactically; the word "lawyer" comes after the possessive word only in the column 1 phrases. It is therefore easy to make TLC decide that the property stored with LAWYER is related to all phrases like those of column 1, but to decide it is not so related to any like those of column 2; one form test will succeed on all phrases where "lawyer" comes after the possessive word, while a different form test is needed for all phrases like those of column 2. It therefore seems a very good idea to store the form test necessary for column 1 phrases with the stored property, but not to store that that would succeed for column 2 type phrases.

However, for a few phrases of the column 2 type, such as "lawyer's client," use of this stored property of lawyer is correct (see Fig. 1B). TLC should recognize such relations to produce a richer comprehension. As stated in the introduction, such

"indirect comprehension" is still a problem for TLC, and all we will do here is indicate a general plan for routines to achieve this.

First, such routines locate intersections which connect parts of two properties in memory, as described in Section IIIC. Second, one of the units so connected must always be part of a property that TLC has already found it can relate directly to the piece of text being comprehended. For instance, to comprehend "lawyer's client" TLC would first find this phrase's relation to the EMPLOY property stored with CLIENT, in the manner described in previous sections. Then, the routine for indirect comprehension would consider relating other properties to the sentence indirectly. Among these properties would be the one stored with LAWYER, stating that a lawyer's occupation is to represent or to advise people in legal matters. This routine would find a superset intersection connecting OCCUPATION with EMPLOY! Having found this connection, the routines should be able to copy and adapt the OCCUPATION property in accordance with the way the EMPLOY property is copied and adapted. This will produce an output like that of Fig. 1B, but one which will not overgeneralize to phrases like "lawyer's enemy." The key element of this solution is that the OCCUPATION property is not directly related to the text, but rather is implied by another property which is so related: employing someone has implications about their occupation. Several parts of this process still involve unsolved problems.

We also believe that, ultimately, a human-like memory should relate descriptive knowledge of the world to perceptual-motor activity, in a manner like that indicated by Piaget (Piaget, 1960, Quillian, Baylor and Wortman, 1964.) This, however, is far beyond our present scope.

IV. SUMMARY

This paper does four things:

1) proposes a structure for a memory of knowledge of the world

2) describes a theory of how natural language may be comprehended using such a memory

3) offers reasons why a computer program embodying this theory may be able to be taught, in a finite length of time, to comprehend language

4) displays the outputs so far produced by one such program

The memory structure represents factual information about the world in a richly connected network. Given any set of concepts represented in this memory, it is possible for a program to locate the conceptual ingredients they have in common, their "intersections" in memory. Every concept represented in this memory, every "unit," is directly associated with some set of factual assertions, its properties, and indirectly chained to an unlimited number of other "superset" units and, hence, with the properties associated with all these other units. The meanings of natural language words are considered to be pointers to particular units in the memory.

The theory of text comprehension is more difficult to summarize. Essentially, it asserts that to read text a comprehender searches his (her, its) memory, looking for properties which can be considered related to that text. This search begins simultaneously at all the representations of concepts, all the "candidate units," which the words of a given fragment of text point to. These will include units corresponding to all of the dictionary meanings of these words, and to any possible anaphoric referents these words may have. These initial units are all considered candidates for the meaning of the word which led to them (until a property related to the text is found, which will provide for a choice among

candidates.) TLC's search through memory is intended to locate, in breadth-first order, intersections connecting properties in memory to words of the input text. Each such intersection that is found causes TLC to form a hypothesis about part of the intended meaning of the current text. Specifically, this hypothesis is that the text means to imply a relationship, somehow similar to the one the property represents, between particular words of the text. These words are the "source" word - that which supplied the initial candidate leading to this property - and the "identified" word - that which has been found to have an intersection with this property.

Having generated such an hypothesis, (on purely semantic grounds, note), the hypothesis is further checked for syntactic feasibility. That is, certain syntactic relationships between the words hypothetically related will allow the hypothesis to remain creditable, while other syntactic relationships will not. Such compatible syntactic relationships must somehow be specified and stored with the property in the memory. At present, such syntactic relationships are represented by routines, "form tests," which check the input string for features allowing particular syntactic relationships to be assumed. (We have discussed the desirability of respecifying form tests as steps in a network grammar.) But, however such relationships are specified, tests must be made at the time an hypothesis about part of the input string's meaning is formed, to see if any one of the compatible syntactic relationships is feasible in the current piece of text. If any is, TLC considers its hypothesis confirmed: the source word and the identified word are considered to be related in a way similar to that specified by the property. Thus the current memory property may be said to be related to this text. This is taken to indicate that the particular candidate of the source word and the particular candidate of the identified word that led to the current intersection

should be considered as the meanings of those words in this text. The memory property related to the text is also taken as a model to be recopied and the copy adapted to encode a part of the text's meaning.

By this technique the comprehension procedure may find a number of properties related to a piece of text, and, using adapted copies of these, create a complex, intra-linked structure, in the same format as the memory, representing the particular meaning of the current input string.

An important feature necessary to make the above strategy work on nested syntactic constructions is the "pending" procedure. This allows the order in which intersections are found within the comprehender's memory to be adjusted so as to adequately match the order required by syntactic nestings of input sentences.

Overall, the most distinctive features of this theory, as compared to other models and theories of language of which we are aware, are its explicitness and detail, and its reliance on "knowledge of the world." The theory assumes that in general very sizable amounts of memory must be searched in order to comprehend text. TLC is designed to carry out such searching with as little wasted effort as possible, and in a breadth-first order, which simulates a largely parallel search mechanism. This kind of "semantic" processing controls the entire comprehension process, with syntactic analysis used in the service of deciphering meaning, rather than, as is often suggested, the reverse.

The argument for TLC as an efficiently "teachable" computer program rests on the fact that both the program's knowledge of the world and its ability to perceive syntactic relationships are fragmented, so that they can be built up in a machine piece by piece. The memory structure allows automatic generalization of each such piece added to this memory, since TLC will recognize a given

property's relationship to text despite considerable variation in the form of that text. Specifically:

1) A property is in general reachable via many different input words.

2) Once reached via some word, either its attribute or its value can be identified with many different other input words.

3) Once part of a property is thus identified with some input word, all the form tests previously associated with any property having the same attribute are available to help determine whether or not the text's syntax implies use of this property.

Finally, examples of TLC's output to date are presented in Figures 1, 4B and 6. We have tried to point out why these begin to illustrate machine comprehension of text, as well as what would seem to be the program's most important current flaws and limitations. As a large and not at all simple program, TLC is - after more than two years of continuous debugging and redesign - still not performing nearly as well as we feel certain it can. Nevertheless, it does at least confront, in considerable detail, the central problem of how to interpret continuous text by relating it to a large memory. We suggest that only to the degree that there is some such detailed, working model of general memory and its use, can language behavior and most other cognitive processing ever be understood by psychologists or linguists, or can reasonable performance on language tasks ever be obtained from a computer.

BIBLIOGRAPHY

Abelson, R.P., and Carroll, J.D. (1965). Computer simulation of individual belief systems. The American Behavioral Scientist, Vol. VIII, No. 9, pp. 24-30.

Bobrow, D.G., Murphy, D.L., and Teitelman, W. (1968). The BBN LISP system. Cambridge, Massachusetts: Bolt Beranek and Newman Inc.

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. To appear in The Journal of Verbal Learning and Verbal Behavior, 1969.

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

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

Kuno, S.K. (1965). The predictive analyzer. Communications of the ACM. 8 (7), pp. 453-462.

Longyear, Christopher R., (1967). The semantic rule. General Electric Tempo - 67TMP-55, Santa Barbara, California.

Minsky, M. (1968). Semantic Information Processing. Cambridge, Massachusetts: M.I.T. Press.

Newell, A., Shaw, J.C., and Simon, H.A. (1962). The processes of creative thinking. In H.E. Gruber, G. Terrell and M. Wertheimer (eds.) Contemporary Approaches to Creative Thinking. New York: Atherton Press, pp. 63-119.

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. In Minsky (1968).

Quillian, M.R. (1967). Word concepts: a theory and simulation of some basic semantic capabilities. Behavioral Science, 12, pp. 410-430.

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.

Raphael, B., Bobrow, D.G., Fein, L., Young, J.W. (1968). A brief survey of computer languages for symbolic and algebraic manipulation. This paper appears in D.G. Bobrow (editor), Symbol Manipulation Languages and Techniques, 1968, North Holland Press.

Reitman, W.R. (1965). Cognition and thought: An information processing approach. New York: John Wiley & Sons.

Siklossy, L., and Simon, H.A. (1968). Some semantic methods for language processing. Complex information processing. Paper #129. Pittsburg, Pennsylvania: Carnegie-Mellon Universit

Simmons, R.F. (1965). Answering english questions by computer: a survey. Communications of the ACM. 8(1), pp. 53-70.

Simmons, R.F., and Burger, J.F. (1968). A Semantic Analyzer for English Sentences. Report No. SP 2987. Santa Monica, California: System Development Corporation.

Tesler, L., Enea, H., and Colby, K.M. (1967) A Directed Graph Representation for Computer Simulation of Belief Systems. Department of Computer Science, Stanford University.

Thompson, F.B. (1965). The deacon project. General Electric Tempo - 65TMP-69, Santa Barbara, California.

Thorne, J.P., Bratley, P. and Dewar, H. (1968). The syntactic analysis of English by machine. In Machine Intelligence 3, Michie, Donald (ed.) New York: American Elsevier Publishing Co., Inc., pp. 281-309.

Weizenbaum, J. (1967). Contextual understanding by computer. Communications of the ACM, 10(8), pp. 474-480.

BLANK PAGE

Unclassified
Security Classification

AD 687746

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
THE TEACHABLE LANGUAGE COMPREHENDER: A SIMULATION PROGRAM AND THEORY OF LANGUAGE

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

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

6. REPORT DATE
31 January 1969

7a. TOTAL NO. OF PAGES
60

7b. NO. OF REFS
18

8a. CONTRACT OR GRANT NO.
F19628-68-C-0125

b. PROJECT NO
8668

c.
DoD Element 6154501R

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
BBN Report No. 1693
Scientific Report No. 10

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
AFCRL-69-0108

10. DISTRIBUTION STATEMENT
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 sponsored by the Advanced Research Projects Agency

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

13. ABSTRACT

The Teachable Language Comprehender (TLC) is a program designed to be capable of being taught to "comprehend" English text. When text which the program has not seen before is input to it, it comprehends that text by correctly relating each (explicit or implicit) assertion of the new text to a large memory. This memory is a "semantic network" representing factual assertions about the world.

The program also creates copies of the parts of its memory which have been found to relate to the new text, adapting and combining these copies to represent the meaning of the new text. By this means, the meaning of all text the program successfully comprehends is encoded into the same format as that of the memory. In this form it can be added into the memory.

Facts and reading abilities may be taught to the program as needed. This information is generalized in TLC and hence a single addition can often provide a large increment in TLC's effective knowledge of the world, and in its overall ability to comprehend text.

The program's strategy is here presented as a general theory of language comprehension.

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
Natural Language						
Semantics						
Language Comprehension						
Computational Linguistics						
Psycholoinguistics						