LANGUAGE PROCESSING VIA CANONICAL VERBS AND SEMANTIC MODELS

Gary G. Hendrix, Craig W. Thompson
and Jonathan Slocum

June, 1973

Technical Report NL 16

Natural Language Research for CAI

Supported by

The National Science Foundation

Grant GJ 509X

The Department of Computer Sciences

and CAI Laboratory

The University of Texas

Austin, Texas 78712

LANGUAGE PROCESSING VIA CANONICAL VERBS AND SEMANTIC MODELS

Gary G. Hendrix, Craig W. Thompson, and Jonathan Slocum

The Department of Computer Sciences

and CAI Laboratory

The University of Texas

Austin, Texas 78712

Abstract

A natural language question answering system is presented. The system's parser maps semantic paraphrases into a single deep structure characterized by a canonical verb. A modeling scheme using semantic nets and STRIPS-like operators assimilates the sequence of input information. Natural language responses to questions are generated from a data base of semantic nets by "parsing" syntactic rules retrieved from the lexicon.

Keywords: Question answerer, canonical verb, semantic net, semantic model, generation, deep case, syntactic paradigm, semantic paradigm, time frame, event scenario, grammar.

Introduction

In natural language processing, the problems of representing factual material, making inferences, solving problems and answering questions may be significantly reduced if identical meanings expressed by diverse surface structures can be represented by a single conceptual construct. A major cause of surface structure diversity is the existence of a wide variety of "surface verbs" for describing basically the same situation. For example, "buy," "sell," and "cost," etc., all describe essentially the same event. Schank (1) and others have posited the existence of "deep" or "canonical" verbs which unify in the deep structure meaning common to possibly many surface verbs used in ordinary speech. Given a canonical event such as EXCHANGE, the participants (e.g. the BUYER, the SELLER) are ordered in the surface structure, but dependent upon the choice of surface verb and voice (active or passive). This paper describes a language processing system which employs semantic network depictions of canonical events as its fundamental representation of meaning.

The language processor consists of three basic modules as shown in Figure 1. The Parser is used to map English surface structures into a semantic representation utilizing canonical verbs. The sequence of events presented to the language processor over several sentences is woven into a unified knowledge structure by the Modeling System, which builds a STRIPS robot-like model of the situations described. Again, as in the Parser, the canonical event provides the basis of representation. When a question is presented to the system, the Generator is used to produce English output directly from the canonical semantic net built (sometime in the past) by the Modeling System.

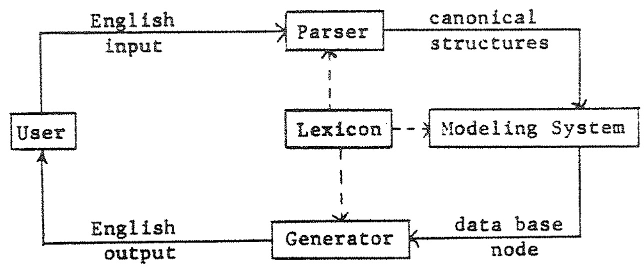

FIGURE 1

A Schematic of the Language Processing System

The Parser

The mapping from English into a canonical constituent structure is accomplished by a variant of the Augmented Finite State Transition Network system described by Woods (2). Since the operation of the system is generally well known, attention will be concentrated on the mechanism which allows for direct translation into canonical structures. The key to this operation is the inclusion of certain information in the Lexicon (Table 1). The property list of each surface verb form (infinitive, past, etc.) contains a reference to a central GENSYM-ed atom which serves as the lexical entry for all forms of a given verb. For readability, these "central" forms will be designated by L-verb, where "verb" is the usual infinitive form. The property list of each central verb contains two special entries (in addition to various others): the attribute CANON-VB indicates the canonical verb (event) associated with the given surface verb; the P-RULES attribute relates the surface noun phrases and prepositional phrases used in conjunction with the given surface verb, to the deep case structure associated with the corresponding canonical verb. This attribute is a list of triples; the first entity in each triple indicates the manner in which an event participant is specified in the surface sentence. If the participant is specified simply as a noun phrase, the first entry of the triple is "OK." If the participant is specified by a prepositional phrase, then the corresponding preposition appears in place of OK. The third entry in the triple names the deep case relation associated with the event participant. For example, consider:

JOHN BOUGHT THE CAR FROM MARY

In Table 1 it may be seen that FROM is associated with the deep case SELLER with respect to the verb BUY. The second entry in each triple is a list of semantic classes: a participant must belong to one of the semantic classes specified by the second triple entry in order to satisfy the deep case relation specified by the third triple entry. This semantic class information is used to help disambiguate the deep cases of sentence components which otherwise appear similar. (This process is also aided by the ordering of the triples.)

```
L-BUY
  WORDCLASS     VERB
  CANON-VB      EXCHANGE
  INF           BUY
  SG3           BUYS
  PAST          BOUGHT
  -EN           BOUGHT
  -ING          BUYING
  PRULES        ( (OK (HUMAN ORGANIZATION) BUYER)
                  (OK (PHYSOBJ) THINGBT)
                  (FROM (HUMAN ORGANIZATION) SELLER)
                  (FOR (MONEY) THINGGIVEN)
                  (AT (PLACE) LOC)
                  (IN (PLACE) LOC)
                  (OK (DAYPART) TIME)
                  (IN (DAYPART) TIME) )
  GRULES        ( (BUYER ACTIVE THINGBT (FROM SELLER)
                         (FOR THINGGIVEN) )
                  (THINGBT PASSIVE (FROM SELLER)
                         (FOR THINGGIVEN) ) )

L-COST
  WORDCLASS     VERB
  CANON-VB      EXCHANGE
  INF           COST
    .             .
    .             .
  GRULES        ( (THINGBT ACTIVE (BUYER) THINGGIVEN) )

L-PAY
  WORDCLASS     VERB
  CANON-VB      EXCHANGE
  INF           PAY
    .             .
    .             .
  GRULES        ( (BUYER ACTIVE (SELLER) (THINGGIVEN)
                         (FOR THINGBT) ) )

L-MAN
  WORDCLASS     NOUN
  S             MAN
  PL            MEN
  SPOSS         MAN'S
  PLPOSS        MEN'S
  MKR           (HUMAN)

L-WHEN
  WORDCLASS     QWORD
  PI            WHEN
  MKR           (DAY DAYPART)

EXCHANGE
  SURF-VB       (L-BUY L-SELL L-PAY L-COST)
```

TABLE 1

Some lexical structures

Since semantic class information will be useful in determining deep cases, each noun entry in the lexicon contains a special attribute (MKR) on its property list; this specifies the semantic classes to which the noun belongs. Interrogative pronouns and adjectives (QWORDS) also have the attribute MKR.

The application of the transition network parsing grammar to an input sentence produces two (three, in the case of questions) intermediate structures. For example:

WHO BOUGHT A CAKE AT THE NEW BAKERY FROM THE BAKER.

will cause the production of the intermediate structures:

Verbal constituent:

```
(CANON-VB EXCHANGE MODAL (TENSE PAST MOOD INTERROG
                          CASE AFFIRM) )
```

NP-PP constituents:

```
( (OK (PHYSOBJ) (TOK L-CAKE DET INDEF NBR S))
  (AT (PLACE) (TOK L-BAKERY DET DEF
               NBR S MOD (AGE L-NEW)))
  (FROM (HUMAN) (TOK L-BAKER DET DEF  NBR S))  )
```

QWORD constituent:

```
(OK (HUMAN) (TOK L-WHO))
```

The Verbal constituent indicates the canonical verb underlying the sentence as well as the sentence modality. The NP-PP constituents list specifies the NP and PP components of the input sentence in the order in which they appeared. Each such component is represented by a triple: the first entry in the triple indicates the syntactic form (PP, or OK in the case of NP) in which the component appeared; the second entry indicates the semantic classes associated with the component (copied from the lexicon); the third entry is an even-order list describing the nature of the component (as given in the input sentence). The appearance of a QWORD in an interrogative sentence will cause a QWORD constituent to be produced: this is a triple having the same form as a (single) NP-PP constituent.

The final stage in mapping English sentences into canonical structures consists of correlating the entries in the NP-PP constituents list, and the QWORD constituent if present, with the PRULES associated with the surface verb. Basically, for each NP-PP constituent a search is made through the first entries of successive PRULES for a preposition or OK which matches the first entry of the NP-PP constituent; when a candidate is found, the rule's second entry is matched against the second entry of the constituent: if the two lists (sets) of semantic markers have a non-null intersection <u>and</u> the deep case indicated by the third entry of the PRULE triple is not already assigned, then that deep case is associated with the constituent as shown below. Once the NP-PP constituents have been assigned their appropriate cases, the QWORD constituent (if any) is considered; the matching process is essentially that described above, except that <u>all</u> the unassigned cases which match are collected in a list to be paired with the attribute ARGS. The resultant structure is assigned the pseudo deep case label Q. Thus the final structure ("canonical structure") produced by the parser for the indicated input sentence is:

```
(CANON-VB EXCHANGE  MODAL (TENSE PAST  MOOD INTERROG
                           CASE AFFIRM)
 SELLER (TOK L-BAKER  DET DEF  NBR S)
 THINGBT (TOK L-CAKE  DET INDEF  NBR S)
 LOC (TOK L-BAKERY  DET DEF  NBR S  MOD (AGE L-NEW))
 Q (ARGS (BUYER)  TOK L-WHO) )
```

The purpose of the parser is to provide a simple, expressive "front end" to the modeling system; the version described is (so far) limited to (possibly conjoined) simple, active sentences without embeddings. Work is in progress to extend the parser to include passives and several types of embedded sentences. Natural extensions include: (1) more complex PRULES, which explicitly express the order of NP's and PP's in the constituents list; (2) a more intelligent correlating algorithm, which is able to recognize NP complements and relative clauses; (3) PRULES for nominalized verbs and nouns modified by PP's; and (4) a richer variety of syntactic and semantic markers.

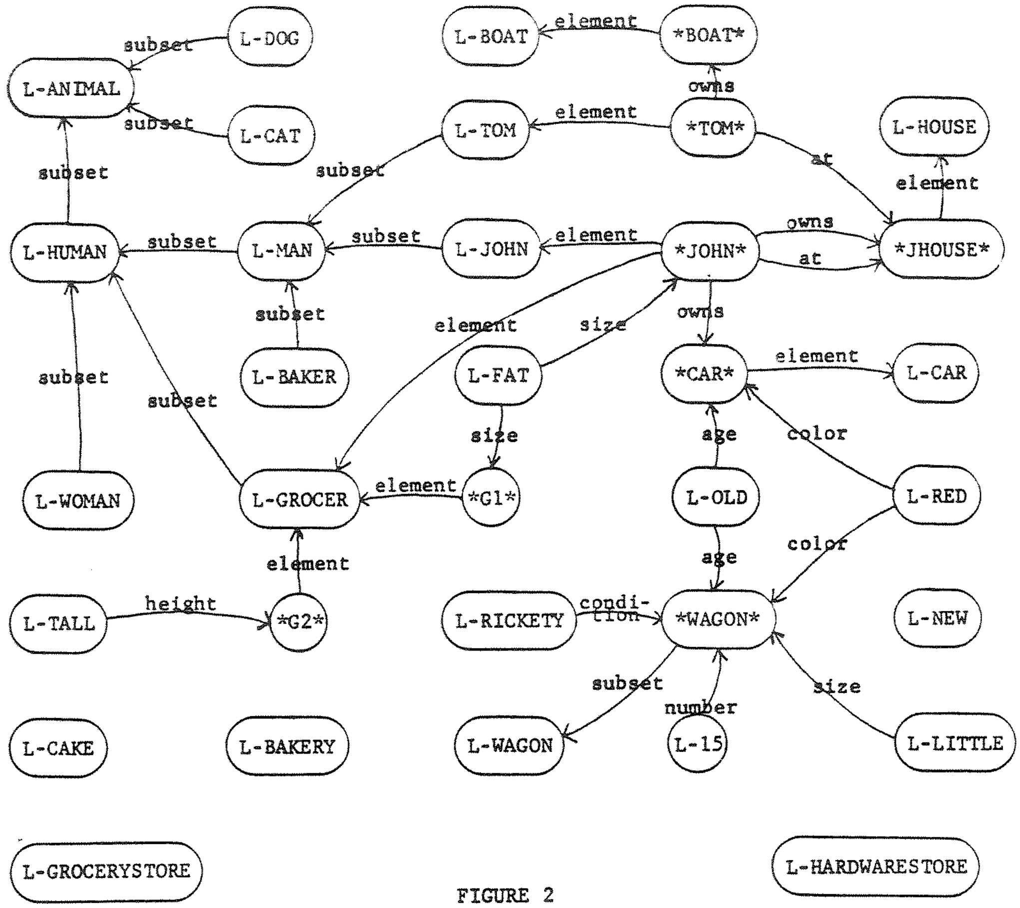

FIGURE 2

A State-of-the-World Graph

The Modeling System

The purpose of the modeling system is to integrate the collection of facts or concepts presented to the system through the parser. Three distinct types of concepts must be encoded: (1) objects; (2) relationships between objects; and (3) changes in relationships with respect to time. The modeling system uses an encoding procedure analogous to that used in motion pictures, recording changes in relationships with respect to time by a sequence of "still photographs." The "still photos" of the modeling system are graphs (nets) representing the state of the world at an instant in time. Nodes in the graph stand for objects, and arcs represent static (binary) relationships between objects. (No time relationships are included.)

Figure 2 is an example of a state-of-the-world graph (SWG). Node names beginning and ending with asterisks (depicting GENSYM atoms) represent particular objects in the real world. Other node names are words in the system's lexicon. Consider the node labeled *JOHN*. *JOHN* is an element of L-JOHN and L-GROCER. L-JOHN is the set of all Johns, a subset of L-MAN, the set of all men. L-GROCER is the set of all grocers. Hence, *JOHN* is a node representing a particular man who is a John who is a grocer. This John owns a particular house, *JHOUSE*. Further, John is at that house, owns an old, red car and is fat.

To model changes in relationships with respect to time, a sequence of SWGs, analogous to the sequence of still photographs of motion pictures, is defined. The sequence of events causing changes in the state of the world is recorded on a time line (see Figure 3). Each node on the time line corresponds to an event. Beginning with some initial state, the state of the world following any event may be determined by considering the sequence of events along the time line. Associated with each event are a list of relationships which held before the event took place but which are no longer valid after the event (the delete list) and a list of relationships which did not hold (or at least were not known to hold) before the event took place but which are valid after the event (the add list). Thus, each node on the time line models a set of changes in the state of the world produced by the occurrence of some event. When known, a canonical verb and a set of parameters which describe the event causing the changes are also associated with a time line node. Not all event parameters need be known ("John bought something at the bakery."), but the more parameters which are known, the better the event is specified.

The language processing system begins operation with a SWG depicting only lexical information. As descriptions of events are given as inputs, nodes are added to the time line. With each new event, certain relations (those on the event's delete list) are deleted from the current SWG and new relationships (those on the add list) are added, transforming the old picture of the state of the world into a new picture showing conditions after the event. To answer questions about relationships in the past, the "movie" may be run backwards by considering the sequence of events in

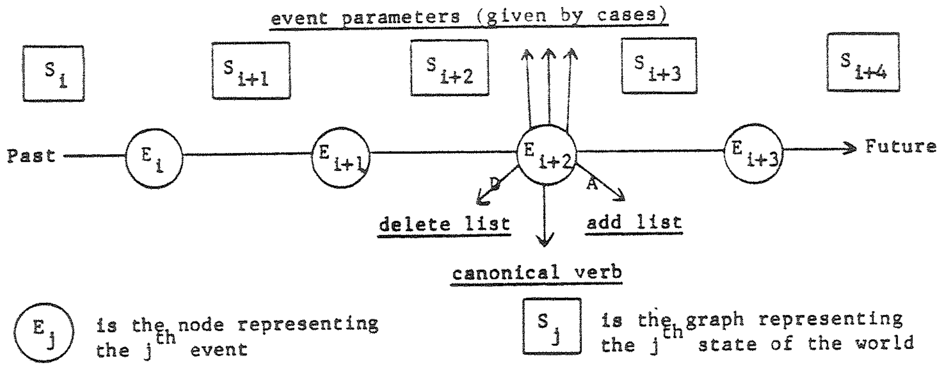

FIGURE 3

Abstract Diagram of the Time Line

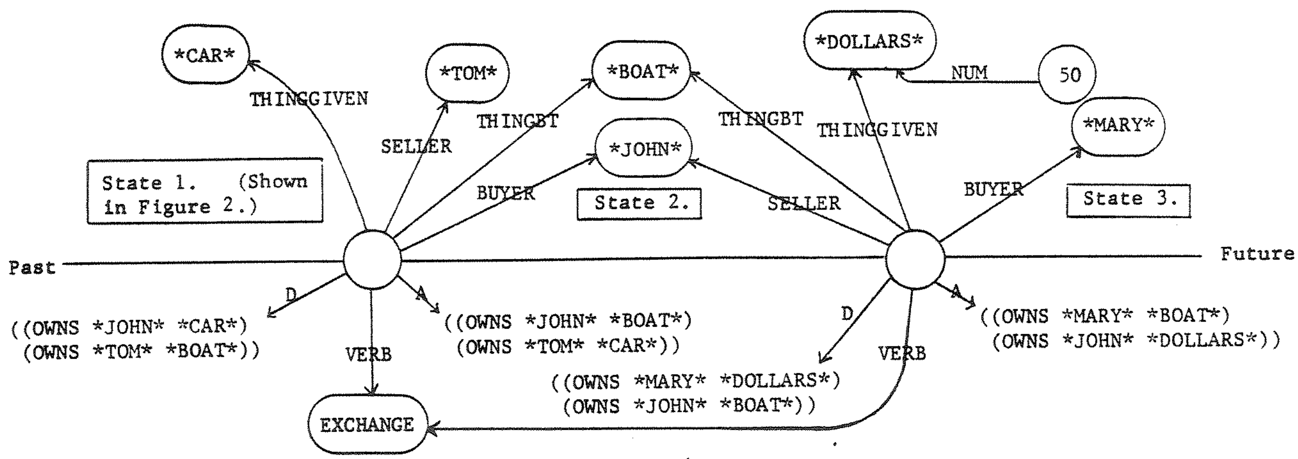

FIGURE 4

Time Line Record of Two EXCHANGE Events
(Note, "intermediate" nodes deleted for simplicity)

reverse order and deleting the adds and adding the deletes. Once a question is answered, an analogous forward process resets the SWG to reflect current conditions.

Questions regarding the nature of events themselves (as opposed to questions concerning relationships which are altered by events) are answered by consulting the record of events on the time line.

As an example of a time line node, consider how the event "John traded his car to Tom for Tom's boat" would change the SWG of Figure 2. The relationships (OWNS *JOHN* *CAR*) and (OWNS *TOM* *BOAT*) would no longer be true. (OWNS *JOHN* *BOAT*) and (OWNS *TOM* *CAR*) would become true. Hence, a time line node like the left node of Figure 4 would be produced.

Events as operators

Since an event transforms one state of the world into another, it is reasonable to think of events as operators. Pursuing this notion, a canonical verb becomes the name of an event procedure (or operation) which transforms the state of the world (an implicit parameter) and a set of explicit event parameters into a new state of the world. The event procedures to be presented here closely parallel the operators used by the STRIPS robot (3).

An integral part of any STRIPS operator is the "precondition list" which the robot uses in determining a legal sequence of operations. Since our system receives a chronological accounting of events, the necessity of precondition lists for determining event sequence is eliminated. There is, however, a related problem: suppose that part of the input to the language processor is "John bought a clock at the hardware store. Then John bought a cake at the new bakery." Note that John was at the hardware store when he bought the clock. The next event has him buying a cake at the new bakery. Before the event at the bakery could take place (as a precondition to the bakery event), John must have gone (perhaps by a complex route) from the hardware store to the bakery. Thus, the necessity of satisfying preconditions clearly remains a problem even when the event sequence is given. The precondition problem has become the problem of determining the nature of unspecified intermediate events which set the stage for the accomplishment of specified events.

The nature of these unspecified intermediate events may, to a large extent, be determined by the preconditions of the specified events. Returning to the example, before "John bought a cake at the new bakery" some event must have occurred which deleted the relation that he was at the hardware store and added the relation that he was at the bakery. (In general, EXCHANGE events must be preceded by events which bring the participants in the event to the place

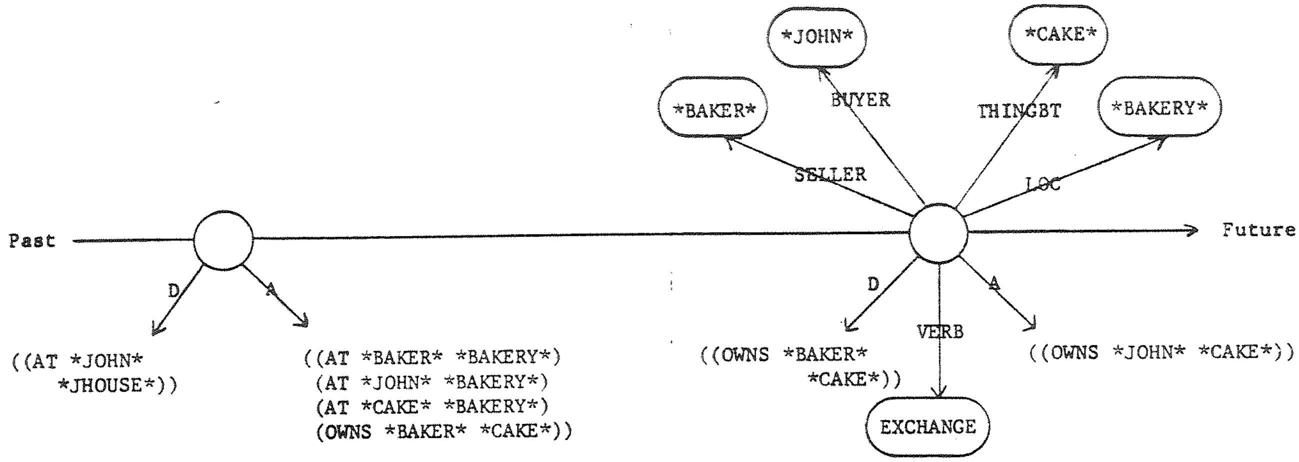

FIGURE 5

Time Line Nodes Created by an Event Procedure

of exchange, etc.)

In the place of the preconditions used by STRIPS robot operators, an event procedure substitutes the add and delete lists of an unspecified intermediate event which transforms any state of the world into a state satisfying the preconditions of the main event. That such an event must have occurred is implied by the sequence of specified events. Thus, a call to an event procedure causes two events to be incorporated into the model, creating two nodes on the time line.

An example of an event procedure, a simplified version of EXCHANGE, is shown below.

procedure call
EXCHANGE(seller buyer thingbt thinggiven loc)

intermediate event delete list
((AT seller *) (AT buyer *) (AT thingbt *)
 (AT thinggiven *)
 (OWNS * thingbt) (OWNS * thinggiven))

intermediate event add list
((AT seller loc) (AT buyer loc) (AT thingbt loc)
 (AT thinggiven loc) (OWNS seller thingbt)
 (OWNS buyer thinggiven))

main event delete list
((OWNS buyer thinggiven) (OWNS seller thingbt))

main event add list
((OWNS buyer thingbt) (OWNS seller thinggiven))

Consider the actions performed by the event procedure EXCHANGE if the system is presented with the event "At the new bakery John bought a cake from the baker" while in the state shown in Figure 2. By a mechanism to be discussed shortly, the system finds the SWG node *JOHN* which represents the particular John under discussion and creates nodes *BAKERY*, *CAKE* and *BAKER* to represent the bakery, the cake and the baker. Relationships integrating these new nodes into the original graph are also produced. With these nodes as arguments, procedure EXCHANGE is entered by the call

EXCHANGE(*BAKER* *JOHN* *CAKE* NIL *BAKERY*)

where NIL represents the unspecified parameter "thinggiven." The effect of the call on the time line is shown in Figure 5. Stepping through the procedure:

1) A time line node is created for the intermediate event. Since the nature of the event is unknown, no canonical verb or event parameters are linked to the node.

2) Working down the intermediate delete list of the event procedure, (AT seller *) is considered first. An attempt is made to match this relationship against those encoded in the SWG. The asterisk is allowed to match anything, but "seller" is bound to *BAKER*. Thus a matching relation must be of the form (AT *BAKER* --). Since no such relation exists in the SWG, no action is taken. The next relationship on the delete list is (AT buyer *) wich matches (AT *JOHN* *JHOUSE*) in the SWG. Hence, (AT *JOHN* *JHOUSE*) is deleted from the SWG and is put on the delete list of the time line node. The relationships (AT thingbt *), (AT thinggiven *), (OWNS * thingbt) and (OWNS * thinggiven) match nothing in the SWG and hence cause no action.

3) Working down the intermediate add list, (AT seller loc) causes (AT *BAKER* *BAKERY*) to be added both to the SWG and to the intermediate time line node add list. Likewise, (AT *JOHN* *BAKERY*) and (AT *CAKE* *BAKERY*) are added. Since thinggiven is bound to NIL, (AT thinggiven loc) causes no action. Proceeding down the add list, (OWNS *BAKER* *CAKE*) is added while the relation (OWNS buyer thinggiven) causes no action.

4) A time line node is set up for the main event. Labeled arcs are created to link the event node to event parameters and the canonical verb EXCHANGE.

5) By working down the main delete list of the event procedure, (OWNS *BAKER* *CAKE*) is deleted.

6) By working down the add list, (OWNS *JOHN* *CAKE*) is added.

By this procedure the original SWG is transformed into an intermediate state in which John and the baker are at the bakery and the baker owns the cake. This intermediate state is then transformed into a state in which John and the baker are still at the bakery, but John owns the cake.

Linking the parser to the modeling system

To see how parser output is utilized by the modeling system, consider the input sentence

"At the new bakery John bought a cake from the baker,"

which is parsed into

(CANON-VB EXCHANGE
 MODAL (TENSE PAST MOOD INTERROG CASE AFFIRM)
 SELLER (TOK L-BAKER DET DEF NBR SING)

```
BUYER (TOK L-JOHN   NBR SING)
THINGBT (TOK L-CAKE   DET INDEF   NBR SING)
LOC (TOK L-BAKERY   DET DEF   NBR SING
    MOD (AGE L-NEW)))
```

The CANON-VB indicates which event procedure must eventually be called. Each deep case argument cited explicitly by the parser output indicates an event parameter which must be bound to some node in the SWG. (Other event parameters are set to NIL.) If no appropriate SWG node exists, one must be created. To do this a routine, FOC (find or create), is called with a parameter's property list as its argument. If the parameter's determiner is indefinite, then FOC simply creates a new node in the SWG satisfying the other properties on the property list. Thus, when "a cake" is mentioned, FOC creates a new cake and does not concern itself with whether or not this new cake is some cake already modeled in the system. (Should it later be determined that two nodes actually model the same entity, a special COLLAPSE function is used to merge them.) If the parameter's determiner is definite (or unspecified), then FOC attempts to find an existing node in the graph satisfying the description of the noun. Thus, when FOC is to return the name of a node corresponding to "the new bakery" it assumes that a new bakery has been talked about before and it looks for a node in the SWG which could be representing that bakery. If more than one such node is found, the last one mentioned is returned. (Each time a node is used a use-time is associated with it.) If no such node is found, one is created.

The value of the attribute TOK on a parameter's property list is assumed to be the name of a node representing a set of entities of which the value of the parameter is an element. In the case of the parameter SELLER, L-BAKER is the name of a set of which the value of SELLER must be a member. Words such as L-BAKER, L-JOHN, L-CAKE, etc., are entered in both the system's lexicon (for use by the parser) and the SWG before processing begins. Certain primitive relationships among the sets named by these words are also preset. For example, the relation (SUBSET L-JOHN L-MAN) is preset in the SWG. Thus, whenever a John is mentioned he becomes an element of the set L-JOHN, a subset of L-MAN, and is therefore known to be a man.

The parameters for "At the new bakery John bought a cake from the baker" are processed from the parser output as follows:

1) The SELLER is determined. FOC looks for an x in the SWG such that (ELEMENT x L-BAKER). Finding no such x, one is created (call it *BAKER*) and it becomes the value of SELLER. (During this process (ELEMENT *BAKER* L-BAKER) is encoded in the SWG.)

2) The BUYER is determined. FOC looks for an x such that (ELEMENT x L-JOHN). It finds *JOHN* which becomes the value of BUYER.

3) The THINGBT is determined. Since the determiner is indefinite, FOC creates a new node, *CAKE* such that (ELEMENT *CAKE* L-CAKE).

4) The LOC is determined. In this instance the job of FOC is complicated by the presence of the modification. FOC looks for an x such that (ELEMENT x L-BAKERY) and (AGE L-NEW x). Finding none, FOC creates such an x and makes it the value of LOC.

Once the values of parameters have been determined, the system calls the event procedure EXCHANGE to encode the event. It is important to note that relationships encoded in the SWG during the process of defining new nodes to serve as parameters are not entered on the add or delete lists of the event's time line nodes. Thus, the new nodes are not eradicated if the model's time frame is backed up over the event. For example, the model will always know who "the baker" is.

Processing interrogative sentences

The processing of interrogative sentences closely parallels that of declarative sentences so far as the determination of parameters is concerned. Of course, rather than direct the system to change the SWG and extend the time line, questions cause the SWG and time line to be examined. There are three basic types of questions the system can answer.

An example of the first type is "What man bought a cake at the bakery?" Questions of this type are answered by examination of the time line. For the current example, a search down the time line (toward the past) is conducted until an event is found whose event class is EXCHANGE and whose LOC parameter has the value *BAKERY*. When such a node is found a test is made to see if the value of the THINGBT parameter is an element of L-CAKE. This test being passed another test is performed to determine if the value of BUYER is an element of L-MAN. If this test is passed then the value of BUYER is the answer to the question. Either the value of BUYER (a noun entity node) or the parent event node may be passed to the response generator. The generator, as will be seen shortly, produces either a noun phrase or a complete sentence answer, respectively.

An example of the second type of question is "What did the baker own before John bought something at the bakery?" The parser splits the question into two parts. A search similar to the one just described is used to find a node on the time line corresponding to the event "John bought something at the bakery." The SWG is then backed up to just before that event and the system investigates the SWG for an x such that (OWNS *BAKER* x). If such an x is found, it is the answer to the question.

An example of the third type of question is "What did the baker own before John owned a cake?" To answer this question the SWG is stepped back until a state is found in which (OWNS *JOHN* x) and (ELEMENT x L-CAKE) are true. Then the SWG is stepped back until one of these relationships is no longer true. Finally, the SWG is stepped back until (OWNS *BAKER* z) is true for some z. If such a z can be found, it is the answer to the question.

The Generator

The generation of English responses from the semantic nets produced by the modeling system is accomplished with another AFSTN grammar. In parsing, the input string (English) controls the transitions of an AFSTN parsing program which produces a canonical structure as a side effect. In generation, the transitions are similarly controlled by a "sentence;" the side effect in this case is an English string. The generator, like the parser, makes use of special lexical information: this information may be regarded as the inverse of CANON-VB and PRULES. Associated with each canonical verb is the attribute SURF-VB which links the canonical verb to surface verbs which may be used to express it; in order to relate deep case structures (semantic nets) to the syntactic patterns of surface verbs, each surface verb has the attribute GRULES on its property list. Table 1 contains some examples of generation rules (GRULES).

The use of SURF-VB and GRULES is probably best illustrated by example -- consider the second semantic net fragment (in time) in Figure 4, based on the canonical verb EXCHANGE. From the SURF-VB property of EXCHANGE (ref. Table 1), such an event may be seen to be expressible as "buying," "selling," etc. In any desired fashion (perhaps by random choice) any one of these verbs may be selected. Suppose L-BUY is chosen. The lexical fragment in Table 1 shows two GRULES associated with L-BUY: the first --

(BUYER ACTIVE THINGBT (FROM SELLER)
(FOR THINGGIVEN)) --

is chosen. This then becomes the control "sentence" to be "parsed" by the generation grammar.

The first element in the rule, BUYER, indicates that an NP is to be generated (as what is commonly called the <u>subject</u> of the sentence) from the node satisfying the deep case relationship BUYER with respect to the event node (in Figure 4): *MARY*. By some method to be discussed later, this NP generation produces the string MARY. The second element in the rule, ACTIVE, indicates the <u>voice</u> in which the sentence is to be generated. The Verb String generation -- discussed by Simmons and Slocum (4) -- produces (for instance) BOUGHT. The next element in the rule, THINGBT, indicates that an NP is to be generated from the node satisfying the deep case relationship THINGBT with respect to the event node in Figure 4: *BOAT*. This NP string might be THE BOAT. The next element, (FROM SELLER), indicates that the node *JOHN* <u>may</u> (parentheses indicate optionality) be generated, and if so, as a PP using the preposition "from." This might result in FROM JOHN. The last element in the rule, (FOR THINGGIVEN), allows the node *DOLLAR* to be generated as a PP with the preposition "for," resulting in FOR 50 DOLLARS. Since the entire GRULE has now been "parsed," the generator simply concatenates these intermediate results and returns the sentence:

MARY BOUGHT THE BOAT FROM JOHN FOR 50 DOLLARS.

Without going into detail, it can be seen that the choice of L-PAY and the rule (BUYER ACTIVE (SELLER) (THINGGIVEN) (FOR THINGBT)), when applied to the identical deep structure, would result in the output sentence:

MARY PAID JOHN 50 DOLLARS FOR THE BOAT.

It is also worth noting that the choices L-COST and the rule (THINGBT ACTIVE (BUYER) (THINGGIVEN)) might produce the sentence:

THE BOAT COST MARY 50 DOLLARS.

This obviously contains less information than the underlying structure as seen in Figure 4, but note that the verb "cost" does not <u>allow</u> the inclusion of the SELLER.

Now consider the problem of generating a sentence from an incomplete underlying structure: delete the THINGGIVEN attribute (or <u>arc</u>) from the example net. If generation is attempted with the same verb and rule choices as in the last example, a non-sentence would be returned:

THE BOAT COST MARY.

Thus the choice of surface verbs and rules is not entirely free: it is necessary that some mechanism test a tentative pattern against the data base net to insure that the required arguments (those not parenthesized) are present in the net. In this example, one may see that the last rule element -- THINGGIVEN -- is not in the (altered) net, thus eliminating this particular GRULE; this, in turn, eliminates the verb L-COST from consideration. Note that any of the other GRULES in Table 1 would be acceptable, since in all these instances the presence of THINGGIVEN in the output string is defined to be optional.

It is possible that one might not wish the system to consistently generate the maximally informative sentence allowed by a rule, even though all elements be present in the data base. For instance, since several of the rules indicate the optionality of case relations SELLER and THINGGIVEN, one might wish to allow their deletion from the sentences produced -- or, better yet, their non-generation. A sentence example (again, from Figure 4) is:

MARY BOUGHT THE BOAT FOR 50 DOLLARS.

This "deletion" could be handled through explicit storage of <u>all</u> of the variants of a rule, with some optional element(s) deleted from each rule; however, this is unnecessarily redundant. Instead, the grammar itself may be constructed so as to allow for this possibility -- perhaps by random omission of <u>optional</u> NP or PP constituents, or by any other heuristic which the grammar writer may wish to employ. (Our system does not perform any of this constituent deletion.) The problem to be recognized here is that one would prefer not to allow the possibility of generating a response (to a question) in which the desired information (the answer) has been "optionally deleted." However, there is an additional possibility for answer generation (which our system does employ) which solves this problem: "answer-only" generation.

<u>Noun Phrase Generation</u>

Most (spoken) answers to questions are not sentences, but rather (noun) phrases; thus there is no reason why a mechanical answer generator must be constrained to the production of "complete" sentences. By happy coincidence, the AFSTN system allows initial control to pass to any node in the grammar -- the language processor employs this facility in sometimes choosing to generate an NP rather than an S.

Consider Figure 4. The simplest answer to the question "Who sold the boat?" is the NP "John." "Mary" is the answer to the question "Who bought the boat?" Now if the response node selected by the modeling system is an <u>event</u> node, then the generator should produce an S. But most often the response is an <u>entity</u> node -- since the modeling system is biased to reply with an entity node if possible. In this case, an NP is to be generated. The generation of NP's as answers to questions is in every way identical to the generation of NP's within sentences. Now since the generation of JOHN from the node in Figure 4 labelled *JOHN* would not particularly clarify any problems in NP generation, we shall consider the example network in Figure 2 and see how an NP is produced in some "worst case."

The node labelled *WAGON* in Figure 2 is an example of an <u>entity</u> node: it corresponds on a one-to-one basis with some particular object in the real world known as <u>wagon</u> -- or, more accurately, it corresponds to a <u>set</u> of (15) wagons. OLD, 15, RED, LITTLE, and RICKETY are predicated about this set. Thus this entity has attributes AGE, NUMBER, COLOR, SIZE, and CONDITION. While all of these predicates may be thought of as MODifiers, there is a good reason for being more precise, as we shall see. Now Figure 2 indicates that these propositions about *WAGON* appear much like event nodes (see Figure 4) -- having directed, labelled arcs which point to an entity node. These labels might even be considered "case relations." But there is one important distinction: <u>events</u> are by nature <u>one</u>-time objects -- they "happen," then they are over; a proposition, on the other hand, is static, it "goes on and on," until something (an event) occurs which changes its "truth value."

If one were to randomly generate the modifiers of *WAGON* in the course of generating that node as an NP, the result might be:

THE OLD RICKETY 15 RED LITTLE WAGONS,
or, THE RED LITTLE 15 RICKETY OLD WAGONS,
or, THE 15 LITTLE OLD RICKETY WAGONS.

Only the last is recognized as being acceptable. What makes it different from the others, obviously, is the ordering of the modifiers. (Winograd (5) ordered his modifiers.) Thus it is seen that the presence of the "case relations" between propositions and their referents can aid in controlling NP generation -- especially in view of the fact that one might posit a control string which would control NP generation much like those used to control S generation. The acceptable (3rd) example above would indicate that one proper control string is:

(NUMBER SIZE AGE CONDITION COLOR)

Now the NP grammar has the relatively simple task of "parsing" such a control string in order to generate modifiers in an acceptable order. For simplicity, proper nouns and certain others (like mass nouns) do not normally take determiners; other nouns take the definite determiner "the" by default.

Conclusions

It might be argued that certain information is lost when a sentence is mapped into canonical form. For example, given

JOHN SOLD THE CAR TO BILL

it might seem that JOHN is in some sense the initiator of the action -- but this is not necessarily true. Instead, it is more likely that the speaker chose to "foreground" JOHN for reasons of discourse development (or whatever). The choice of "subject" and "object" in a sentence is apparently important to thematic development and anaphoric resolution (6, 7), yet it is not at all clear that such syntactic information need appear in the final representation of the meaning of the sentence.

The strict chronological sequence demanded by the modeling system causes input texts to make very boring reading. Rather than being restricted to time line growth on the right, tense and other time clues provided by the input sentences (8) should be able to guide the insertion of events into the history portion of the time line. Even with such extension, the system would still be unable to account for simultaneous events, or the occurrences of events in an unknown order. Both of these problems could be solved, however, by generalizing the time line to be a partially-ordered graph.

It is apparent that the generation of a reasonable number and variety of English sentences is indeed a simple task, when using the AFSTN system and "parsing" a control string drawn from the lexicon. Yet unimplemented extensions of this scheme would allow imbedded sentences and occasional "fronting" of certain (prepositional) phrases -- typically those expressing time.

This language processing system has been implemented in GROPE, a graph processing language (9, 10) on the CDC 6600 at the University of Texas. The system has proven to be quite satisfactory in answering questions relating sequences of events -- indeed, for sequences which follow no particular pattern, it is generally faster and more accurate in answering questions than a human competitor. (Response time for most questions is in the neighborhood of two seconds on a time-sharing system, and the actual processing time is of course less than that.) More extensive documentation of the system and its support is available in Matuszek & Slocum (11), Thompson (12), Hendrix (13), and Slocum (14).

Bibliography

1. Schank, R. C., Goldman, N., Rieger, C. J. and Riesbeck, C. K., "Primitive Concepts Underlying Verbs of Thought," Memo AIM-162, Computer Science Department, Stanford University, Stanford, California, February 1972.

2. Woods, W. A., "Transition Network Grammars for Natural Language Analysis," Comm. ACM, 13, 10 (October 1970), pp. 591-606.

3. Fikes, Richard E. and Nils J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, II, 1971, 189-208.

4. Simmons, R. F., and J. Slocum, "Generating English Discourse from Semantic Networks," CACM, XV, 1972, 891-905.

5. Winograd, T., Understanding Natural Language, Academic Press, New York, 1972.

6. Chafe, W. L., Meaning and the Structure of Language, University of Chicago Press, Chicago, Illinois, 1970.

7. Baranofsky, S., "Some Heuristics for Automatic Detection and Resolution of Anaphora in Discourse," Master's Thesis, The University of Texas, Austin, Department of Computer Sciences, January 1970.

8. Bruce, Bertram C., "A Model for Temporal References and Its Application in a Question Answering Program," Artificial Intelligence, III, 1972, 1-25.

9. Friedman, D. P., "GROPE: A Graph Processing Language and its Formal Definition," University of Texas at Austin, Department of Computer Sciences, Dissertation, June, 1973.

10. Slocum, J. "The Graph Processing Language GROPE," University of Texas at Austin, Department of Computer Sciences, Thesis (in preparation).

11. Matuszek, D. and Slocum, J., "An Implementation of the Augmented Transition Network System of Woods," University of Texas at Austin, Comp. Sci. Tech. Report NL9, October 1972.

12. Thompson, C. W., "Question Answering Via Canonical Verbs and Semantic Models: Parsing to Canonical Verb Forms," University of Texas at Austin, Comp. Sci. Tech. Report NL11, January 1973.

13. Hendrix, G. G., "Question Answering Via Canonical Verbs and Semantic Models: A Model of Textual Meaning," University of Texas at Austin, Comp. Sci. Tech. Report NL12, January 1973.

14. Slocum, J., "Question Answering Via Canonical Verbs and Semantic Models: Generating English from the Model," University of Texas at Austin, Comp. Sci. Tech. Report NL13, January 1973.