# TOWARD ASSOCIATING SLOT NAMES WITH MEANINGS

by

Tsun-dah Shih

Submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1977

APPROVED:

Vesko Marinov, Assistant Professor
Department of Computer Science
in Charge of Major

Date presented: June 1977

July 1977

# TOWARD ASSOCIATING SLOT NAMES WITH MEANINGS

Tsun-dah Shih

Department of Computer Science
Oregon State University
Corvallis, Oregon

## ABSTRACT

The paper describes research on the representation of knowledge. The goal is to develop a formalism which can be used for the testing of hypotheses on the nature of human understanding and as a foundation for artificial intelligence programs. The ideas expressed herein are implemented in a program which converses with the user in natural language about the objects in a room and their relationships.

The representation is within the "frame" paradigm. Slots are utilized for storing the properties of a concept. The same representation is used for both the physical objects and the meaning of the sentences. A novel technique in our approach is to associate a slot name with a procedure describing the meaning of the property represented by the slot. In addition, the objects in our representation are identified by description, i.e. there are no proper names associated with the individual objects as in all other approaches known to us.

CONTENTS

## Section 1. Introduction

The issue of representation of knowledge has established itself as the main problem in artificial intelligence at the present time. It appears to be the greatest obstacle towards building intelligent machines. The problems of representation are important for natural language understanding as well as for machine vision, speech understanding, and most other directions of work in artificial intelligence.

As a result, representation of knowledge has attracted considerable attention on the part of artificial intelligence researchers. On the extreme one can distinguish two major trends: declarative vs. procedural representations. Each approach has its advantages and drawbacks. In a declarative representation it is easy to add new knowledge but hard to express transfer of control. In a procedural representation on the other hand the transfer of control is very convenient but it is hard to modify the knowledge base. Currently there are many attempts to combine procedural and declarative representations.

Many contemporary efforts on representation attempt to utilize the notion of frames. In such a representation customarily there is a slot for each property of a concept. Both in a frame-like as well as in other representations such as semantic nets, there is a lot of information

implicit in the name of the slot. The name means a lot to the user but for the machine is just another string.

A novel technique in our approach is to define slot names to be procedures embodying the meaning of the property represented by the slot. In the LISP implementation of our ideas, a slot name coincides with a function (procedure) name, while the filler coincides with the functional arguments.

In order to test our ideas on representation we have incorporated them in a simple natural language understanding program. The program converses with the user in English about a few simple objects in a room and their relationships. For the meaning of sentences and phrases we utilize the same representation formalism used for the meaning of physical objects and actions.

There are six sections in this paper. Section 2 and 3 survey briefly the developments of knowledge representation in artificial intelligence and of natural language understanding systems, respectively.

Section 4 gives a sample dialogue with the ROOM program on which we test our representation ideas. Section 5 describes knowledge representation in our program. Section 6 shows the implementation of our formalism.

Section 7 discusses some future directions of improvement to the representation formalism and the linguistic analysis,

which we believe will improve the program's performance and bring it closer to that of a human.

## Section 2. Representation of Knowledge

The "state space" and "problem reduction" frameworks [13] in the early work on problem solving involved some sort of "artificial" representation. For example, the solutions to the classic problems of Monkey and Bananas or Tower of Hanoi seemed to require a human translation of the initial problem into the "right" representation so that heuristic search could be applied to obtain the "goal states" for the problem.

Another more general approach is the Predicate Calculus framework [9,13,18]. Basic facts in Predicate Calculus are expressed in the form of predicates. Complex facts can be deduced from the basic ones through axioms. With the incorporation of the more powerful resolution-based theorem proving techniques, high hopes were associated with this approach. Since the inference procedure is based completely on syntax, it is somewhat difficult to incorporate semantic heuristic knowledge denoting how to derive the inference.

The procedural approach to knowledge representation is based on the assumption that knowledge is primarily "knowing how" [5,18]. The knowledge is in the form of programs, which makes it easy to provide extensive guidance of the inference but limits it to a particular problem and a specific aspect of this problem as well. Implementations using this approach made use of a general inferential scheme which is based on Planner-like languages.

The "production systems" approach is a pattern-based representation [4,12], which consists of three components in operation: a linearly ordered set of production rules, a data base, and an interpreter. In general, a rule is a string of symbols with a left and right hand side. A data base is a collection of symbols. An interpreter scans the left hand side of each rule until a rule is found, which matches against some sequence of symbols in the data base. The matched symbols in the data base are then replaced by the string of symbols in the right hand side of the rule. The process of scanning continues either with the next rule or from the beginning of the set of rules.

Semantic network is a type of representation that is broadly used in natural language programs or speech understanding systems [11,15,21]. It has been formulated and utilized in various ways. The network, in general, consists of nodes and links. A node sometimes represents the word sense meaning, while a link represents the relationship held between nodes, and the network represents the common knowledge in a specific domain.

Minsky's "frame theory" set up a new approach to the representation of knowledge problems. Minsky says: [10]

> "Whenever one encounters a new situation (or makes a substantial change in one's viewpoint) he selects from memory a structure called a <u>frame</u>: a remembered

framework to be adapted to fit reality by changing details as necessary . . . ."

"Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed . . . ."

Minsky himself did not follow through this idea to an actual working system. Some of the recent work, however, is attempting to implement frames.

Currently, a language called KRL (Knowledge Representation Language) is under development [3]. It is designed as a formalism to be used by understander systems. It utilizes frames and also integrates the procedural and declarative knowledge to some extent. The representation for declarative knowledge is based upon descriptions. An object is described by its properties, while a complex event is frequently described by means of multiple perspectives. The procedural knowledge is incorporated into the representation by "procedural attachment". Other extended control structures are also included in KRL: multiprocessing and variable depth processing; process frameworks, procedure directories, and directory modules.

## Section 3. Natural Language Understanding--an important area of AI research

A great deal of research on natural language understanding has been done during the past fifteen years within the field of artificial intelligence. Winograd [16,17] discussed in some detail the existing language understanding systems. From his point of view, the evolution of mechanical understanding can be divided into three stages: the early AI systems, the second generation AI systems, and the current developments.

A number of computer systems were implemented in the first period. One such system was Bobrow's STUDENT system [1]. It represented the meanings of sentences in terms of a set of linear equations and could store other linear equations so that the program using them could solve problems in high school algebra.

Another approach characterized as "limited logic" was taken for answering questions. SIR [14], for example, made use of word associations as well as pattern matching techniques to examine input sentences and could answer such questions as "Is X a part of Y?", etc. In SIR the meaning of a concept is stored on the property lists of the LISP atom corresponding to the concept's name.

"Second generation" AI systems spanned the late sixties and the early seventies. These systems differed from the

early ones in that they hoped to find solutions to a large range of problems of language. Two distinct largely well-developed dialogue systems are discussed.

One of them was Woods' LUNAR system which was designed as a natural language information retrieval system for lunar geology [20]. The system understood a large set of questions. It gave responses with a very limited structure, but generally satisfying the needs. Augmented Transition Networks [6,19] were used for accepting input questions and for translating them into a special query language which was designed to interface with an information retrieval system.

An Augmented Transition Network is a finite state recognizer with augmented features so as to allow recursive calls to the network itself or to other networks. Attached to a network is a set of states, arcs, and "naming actions" associated with each of the arcs. An arc specifies the condition for the current word in an input. A state tells what has been processed so far. A "naming action" binds the current word to a "functional name" of the syntactic structure of an input. The processor operates in a manner similar to the manner of a finite state recognizer.

The second system was Winograd's SHRDLU program [16] which converses with a person about manipulations of toy blocks. The program represents its syntactic, semantic, and reasoning knowledge as procedures. It operates

concurrently with these components rather than treating each of them as separate.

After the "second generation" AI systems, researchers have concentrated more on the development of a good formalism for the representation of knowledge than on the implementation of dialogue systems or the implementation of systems for understanding.

An example of a contemporary dialogue system is the GUS system under development in Xerox PARC [2]. It conducts a dialogue with a traveller, for planning a simple trip. The dialogue is directed by frames. The system utilizes recursive techniques in order to acquire the needed information specified in a slot before going on to another slot of the same level. GUS uses procedural attachment to facilitate the reasoning and to keep its initiative during the conversation with the traveller. The attached procedures are of two types: DEMONS and SERVANTS. A DEMONS type procedure is evaluated as soon as the slot is filled, while a SERVANT type procedure is called with respect to the condition specified in the slot.

## Section 4. A Dialogue with ROOM

The ROOM program was developed at the computer science department at Oregon State University [8]. It is a natural language understanding system. It has been used as a testing ground for different representation ideas and linguistic analysis.

The representation utilized in the program is in the spirit of the "frame" paradigm. We are aiming to develop a formalism which can be used by machines for representing knowledge.

An important aspect is that individual objects in the program are identified by description, i.e. there are no proper names associated with them [7]. This approach is similar to the nature of human understanding. Humans do not use generated symbols for identifying objects in an environment. Nevertheless natural intelligence performs quite well without them as we all know.

The system converses with the user about the objects in a room and their relationships. The room serves as the boundary of the program's world. To reduce the impact of the lack of vision we describe to the program (as we would describe to a human) a room containing a few simple objects and we talk about them. The system accepts sentences describing new events. The new events could either introduce new objects or make assertions about existing objects. The system could also answer several types of questions.

A sample dialogue with the system is given on page 12-13. The bracket numbers in the dialogue were added for reference purpose. The system responds with OK to carry out an appropriate action and with I SEE to modify a given object's properties. It says I KNOW if an event has been previously introduced. It gives an error message for an impossible event or a wrong assertion.

```
      *(CONVERSE)

[1]   ROOM:  HELLO
[2]   USER: *WHAT IS IN THE ROOM

[3]   ROOM:  NOTHING
[4]   USER: *THERE IS A RED BOX IN THE ROOM

[5]   ROOM:  OK
[6]   USER: *A BLACK BALL IS IN THE BOX

[7]   ROOM:  OK
[8]   USER: *THE BOX IS PUT ON A SMALL TABLE

[9]   ROOM:  OK,  THE SMALL TABLE NOW SUPPORTS THE RED BOX
[10]  USER: *THE BALL IS ON THE TABLE

[11]  ROOM:  IN FACT THE BLACK BALL IS IN THE RED BOX
[12]  USER: *WHAT IS IN THE ROOM

[13]  ROOM:  A SMALL TABLE ,  A BLACK BALL ,  AND A RED BOX
[14]  USER: *WHERE IS THE BOX

[15]  ROOM:  ON THE SMALL TABLE
[16]  USER: *WHICH BALL IS IN THE BOX

[17]  ROOM:  THE BLACK BALL
[18]  USER: *THE BALL IS BIG

[19]  ROOM:  I SEE
[20]  USER: *THE BALL IS PUT IN A GREEN BOX

[21]  ROOM:  OK,  THE GREEN BOX NOW CONTAINS THE BIG BLACK BALL
[22]  USER: *THE BOX IS PUT ON A BIG TABLE

[23]  ROOM:  OK,  THE BIG TABLE NOW SUPPORTS THE GREEN BOX
[24]  USER: *WHAT IS ON THE SMALL TABLE

[25]  ROOM:  A RED BOX
[26]  USER: *IS THERE A BALL IN THE RED BOX

[27]  ROOM:  NO
[28]  USER: *THERE IS A BROWN BALL IN THE RED BOX

[29]  ROOM:  OK
[30]  USER: *A GREEN BALL IS IN A SMALL BOX

[31]  ROOM:  OK
[32]  USER: *THERE IS A BIG BALL IN A BOX
```

```
[33]  ROOM:  I KNOW
[34]  USER: *THE BIG BALL IS BROWN

[35]  ROOM:  **ERROR** THE BALL IS BIG AND BLACK
[36]  USER: *WHICH BALL IS BLACK

[37]  ROOM:  THE ONE IN THE GREEN BOX
[38]  USER: *WHERE IS THE GREEN BOX

[39]  ROOM:  ON THE BIG TABLE
[40]  USER: *IS THERE A BALL ON THE TABLE

[41]  ROOM:  NO
[42]  USER: *THE GREEN BALL IS PUT ON THE TABLE

[43]  ROOM:  OK,  THE BIG TABLE NOW SUPPORTS THE GREEN BALL
[44]  USER: *WHAT IS ON THE TABLE

[45]  ROOM:  A GREEN BALL AND A GREEN BOX
[46]  USER: *THE GREEN BOX IS PUT IN THE SMALL TABLE

[47]  ROOM:  **ERROR** IT IS WRONG THAT A TABLE CONTAINS A BOX
[48]  USER: *THE SMALL BOX IS PUT ON THE SMALL TABLE

[49]  ROOM:  OK,  THE SMALL TABLE NOW SUPPORTS THE SMALL BOX
[50]  USER: *WHAT IS ON THE SMALL TABLE

[51]  ROOM:  A SMALL BOX AND A RED BOX
[52]  USER: *THE BROWN BALL IS PUT IN THE RED BOX

[53]  ROOM:  IT IS ALREADY THERE
[54]  USER: *WHICH BOX IS ON THE BIG TABLE

[55]  ROOM:  THE GREEN BOX
[56]  USER: *IS THE BIG BALL IN THE GREEN BOX

[57]  ROOM:  YES
[58]  USER: *THE BIG BALL IS BLACK

[59]  ROOM:  YOU TOLD ME BEFORE
[60]  USER: *WHAT IS IN THE ROOM

[61]  ROOM:  A GREEN BALL ,  A SMALL BOX ,  A BROWN BALL ,  A BIG TABLE ,
      A GREEN BOX ,  A SMALL TABLE ,  A BIG BLACK BALL ,  AND A RED BOX
[62]  USER: *BYE

[63]  ROOM:  THANK YOU
      NIL
      *
```

## Section 5. Knowledge Representation in ROOM

The system has its world, linguistic, procedural, syntactic, and inferential knowledge which operate in an integrated manner to understand the context. An input sentence is parsed by its syntactic knowledge. The world, linguistic, and procedural knowledge are used to form the representation for the sentence. The inferential knowledge makes use of the representation for subsequent inferencing and processing.

Knowledge representation in the general case would necessarily be more complex than knowledge representation in the restricted case of ROOM. All comments the author makes in this paper refer specifically to ROOM and not, except as noted, to the general case.

### 5.1 World Knowledge

ROOM uses a frame corresponding to each concept occurring within our specific context. The frame is further specified according to the information provided. Given a frame, a set of slots is associated with the frame for storing its properties. The information of the slots are effective only within this frame. The slots appearing in a frame are sensitive to the context of what we are doing. For example, the concept of "ball" in our context may have several properties such as size, color, and location.

In our representation a frame consists of a frame name as well as a set of slots with slot names and fillers for describing properties of the concept corresponding to the frame name. (A frame name indicates the concept to which it corresponds.) The slot names of the slots are LISP atoms. The fillers are represented in a large variety: a LISP atom, a list of LISP atoms, another frame, a list of other frames, or a set of sub-slots. All these are illustrated by examples in this section, or in section 5.2, or in section 5.3.

For example, the representation of the physical object A BIG RED BOX is shown in Fig. 1. The frame name is BOX. There are four slots in the frame--SIZE, COLOR, CONTAINS, and LOCATION.

```
(BOX
 (SIZE BIG)
 (COLOR RED)
 (CONTAINS NIL)
 (LOCATION ((POINTER NIL)
            (RELATION NIL))))
```

Fig. 1 Representation of A BIG RED BOX

The fillers of the SIZE and COLOR slots are represented differently from the filler of the CONTAINS or the LOCATION

slot. They are simply LISP atoms, i.e. BIG and RED. Each of these slots can be viewed as a pair of attribute and value, for describing the BOX's size and color.

The filler of CONTAINS is also represented differently from the filler of LOCATION. In the CONTAINS case, it is NIL in this example. If the box contains other physical objects, then the filler becomes a list of other physical object frames. For example, if the box in the example contains a big ball and a green ball, the CONTAINS slot in turn has the following representation instead of the one shown in Fig. 1.

```
(CONTAINS ((BALL (SIZE BIG)
                 (LOCATION ((POINTER (BOX & & & &))
                            (RELATION IN))))
           (BALL (COLOR GREEN)
                 (LOCATION ((POINTER (BOX & & & &))
                            (RELATION IN))))))
```

Fig. 2 Representation of the CONTAINS slot

The filler of the LOCATION slot is a list of two sub-slots. The first one is POINTER. Its filler is a pointer to another physical object frame. The second one is RELATION which is an attribute-value pair, with a

preposition for the filler. In Fig. 2, for example, BIG BALL's location is IN BOX.

The slots in the BOX frame are further characterized as structural, functional, and existential. Structurally, a box has its size and color. Functionally, it contains other physical objects. Existentially, it has its location.

These three types of slots, included in the BOX frame, give the representation flexibility, modularity, and generality.

The flexibility is indicated by the many possible uses of the frame. The frame could be utilized to answer several questions about the box as given by the inputs: WHAT IS IN THE BOX, WHERE IS THE BOX, WHICH BOX IS IN THE ROOM, etc. It could also be used to represent some new facts as given by the inputs: A BALL IS IN THE BOX, THE BOX IS PUT ON THE TABLE, etc.

The modularity derives from the fact that each of the three types of slots is independent of the others. Modification of one of the properties does not require changes for others. Addition of new information becomes a process of accumulation of the information. For example, putting a ball in the box requires only the addition of a ball frame to the CONTAINS slot. There is nothing to be done about the box's location or its color and size.

The generality is illustrated by the moving of a box

from one location to another. All objects contained in the box to be moved, are moved without explicitly reconstructing the data base and deducing each time that the objects are moved because the box is moved.

The WORLD representation contains the physical objects in the room, with their relationships to others. The room serves as the boundary of the program's world. The WORLD representation is similar to the representation of physical objects, as shown in Fig. 3. There is only one CONTAINS

```
(ROOM
 (CONTAINS NIL))
```

Fig. 3 The initial WORLD representation

slot in it. The filler is NIL initially. The physical objects mentioned in the conversation are either identified and modified appropriately or otherwise created and added to the slot.

Notice that an individual object in WORLD is not isolated from others. Rather, it is linked to another object or simply WORLD such that the WORLD representation is a network showing the current state of the environment.

## 5.2 Linguistic Knowledge

The representation described in section 5.1 is utilized also for the meaning of the phrases and sentences.

The NPF frame is used for representing the syntactic meaning of a noun phrase. Fig. 4 shows the NPF frame for the noun phrase A BIG RED BOX. There are three slots in the

```
(NPF
 (DET A)
 (ADJ (BIG RED))
 (NOUN BOX))
```

Fig. 4 Syntactic representation of A BIG RED BOX

frame--DET, ADJ, and NOUN. The fillers of the DET, ADJ, and NOUN slots are the definite determiner, the list of adjectives, and the noun of the noun phrase, respectively. The DET and NOUN slots are attribute-value pairs. The filler of ADJ is a list of LISP atoms.

The meaning of a noun phrase is formed as soon as the NPF and physical object frames are established. Fig. 5 shows the representation of the same noun phrase A BIG RED BOX. There are two slots in the frame--SYNTAX and SEMANTICS. The filler of SYNTAX is the NPF frame in Fig. 4, while the filler of SEMANTICS is the physical object frame for BOX in Fig. 1.

```
(NP
 (SYNTAX (NPF
          (DET A)
          (ADJ (BIG RED))
          (NOUN BOX)))
 (SEMANTICS (BOX
             (SIZE BIG)
             (COLOR RED)
             (CONTAINS NIL)
             (LOCATION ((POINTER NIL)
                        (RELATION NIL))))))
```

Fig. 5 Representation of the noun phrase A BIG RED BOX

The representation of a declarative sentence is somewhat more complex than the representation for a physical object or a noun phrase. Suppose the declarative sentence A BIG RED BOX IS ON A SMALL TABLE is the first sentence described by the user, then the sentence is represented as shown in Fig. 6. The frame name is DECL-SENT, which is significant to the process of retrieval (see section 5.5). There are six slots in the frame--SUBJECT, SUBJECT-SEARCH, OBJECT, OBJECT-SEARCH, PREDICATE, and REPLY.

```
(DECL-SENT
 (SUBJECT (NP
          (SYNTAX (NPF
                   (DET A)
                   (ADJ (BIG RED))
                   (NOUN BOX)))
          (SEMANTICS (BOX
                      (SIZE BIG)
                      (COLOR RED)
                      (CONTAINS NIL)
                      (LOCATION ((POINTER (TABLE & & &))
                                 (RELATION ON)))))))
 (SUBJECT-SEARCH ((RESULT NONE) (POINTER NIL)))
 (OBJECT (NP
          (SYNTAX (NPF
                   (DET A)
                   (ADJ (SMALL))
                   (NOUN TABLE)))
          (SEMANTICS (TABLE
                      (SIZE SMALL)
                      (SUPPORTS ((BOX & & & &)))
                      (LOCATION ((POINTER (ROOM &))
                                 (RELATION IN)))))))
 (OBJECT-SEARCH ((RESULT NONE) (POINTER NIL)))
 (PREDICATE ((VERB BE) (RELATION ON)))
 (REPLY 3))
```

Fig. 6 Representation of the declarative sentence

A BIG RED BOX IS ON A SMALL TABLE

The SUBJECT and OBJECT slots describe the subject and object of the sentence, i.e. A BIG RED BOX and A SMALL TABLE. Their fillers are NP frames.

The SUBJECT-SEARCH and OBJECT-SEARCH slots keep the results of the identification process (see section 5.5) for the subject and object. Their fillers contain two sub-slots: RESULT and POINTER. If an object is identified, RESULT will be either SIMILAR or FOUND, and POINTER points to a particular physical object frame. Otherwise, RESULT will be either NONE or NOTFOUND, and POINTER is NIL. Since the sentence in this example is assumed to be the first sentence described by the user, it is obvious that both the subject and object could not be identified. Therefore the fillers of RESULT for both the SUBJECT-SEARCH and OBJECT-SEARCH are none. The fillers of POINTER are NIL.

The filler of PREDICATE is a list of two sub-slots: VERB and RELATION, keeping the verb and preposition of the sentence, respectively.

The REPLY slot is an attribute-value pair with its filler being a number. The value part is no longer an inherited property of the slot name but an indicator corresponding to a specific reply stored in the list of REPLIES (see page 37).

Questions are represented in a similar way to declarative sentences, but with fewer slots and a different frame name corresponding to a specific type of question (see section 6.8).

In addition to the WORLD representation containing the physical objects in the room, there is a HISTORY representation for keeping previous sentences. It is an ordered list of the previous sentence frames, according to the order of events described by the user. All sentence frames kept in HISTORY are meaningful with respect to the context, i.e. they are not describing the same events or providing incorrect information. Objects mentioned in the sentences share structures with the WORLD objects.

## 5.3 Procedural Knowledge

Section 5.1 and 5.2 described the frame representations for physical objects as well as the meaning of phrases and sentences. The information represented by the slots are declarative facts about the concepts' properties--the color of the box is RED (in Fig. 1), the determiner of the noun phrase is A (in Fig. 4), the subject of the sentence is A BIG RED BOX (in Fig. 6), etc. However, there is no specification of how to use such information. The information makes sense to us because of our knowledge. It may not be so obvious for our machine to have the same interpretation.

For example, the slots of the NPF frame in Fig. 4 are used for keeping the words of the noun phrase in three catagories, according to their grammatical features. When we talk about the adjectives in the ADJ slot, we assume the

adjectives modify the noun in the noun slot. Also when we talk about the definite determiner in the DET slot, we know that the noun is possibly referred to an existing object. With our knowledge of the noun phrase, we notice that there are relationships held among the slots. But how does our machine know that? Solution of this question is accomplished by using an order slot which is filled with procedure names. The author believes that this is an original contribution to the technique of frame theory.

In this example, the procedural knowledge needed to manipulate the relations with the slots is introduced in an ORDER slot. The syntactic meaning of the noun phrase A BIG RED BOX is now represented as shown in Fig. 7 instead of the previous one in Fig. 4, by adding an ORDER slot.

```
(NPF
 (DET A)
 (ADJ (BIG RED))
 (NOUN BOX)
 (ORDER NOUN ADJ DET))
```

Fig. 7 Syntactic representation of A BIG RED BOX

The ORDER slot does not contain any word which belongs to the words of the noun phrase but all slot names instead.

The slot names appearing in the ORDER slot are defined as procedural. In the LISP implementation of this idea, a procedural slot name is a function (procedure) name which takes the filler as its functional arguments. The procedure would accomplish the task of processing whatever has been specified in advance.

The ORDER slot further specifies the order of using the information in the frame, i.e. the order for the evaluation of the procedures. In Fig. 7 the order of evaluation happens to be NOUN, ADJ, and DET. The procedural knowledge for the given noun phrase is used to form a physical object frame for the concept of BOX, to insert all structural properties (BIG and RED) in the BOX frame, and to identify the BIG RED BOX (see section 5.5).

As for the declarative sentence frame in Fig. 6, the information attached to the PREDICATE and REPLY slots again has implicit meaning to us. We know that the verb and preposition in the PREDICATE slot indicate the relationship between the subject and object of the sentence. Also it makes sense assuming that the number in the REPLY slot corresponds to one of the replies stored in the list of REPLIES, although the number is no longer an inherited property of the slot name. But our machine may not notice these facts so easily. Therefore, PREDICATE as well as REPLY are defined as procedural slot names so that procedural knowledge is incorporated to perform the same analysis as

we did.

The meaning of the declarative sentence is now represented as shown in Fig. 8 instead of the one given in Fig. 6, with the addition of an ORDER slot. The ORDER slot specifies the order for the evaluation of the procedures to be PREDICATE and REPLY. The procedural knowledge associated with the declarative sentence frame is needed to utilize all kinds of information in the frame and the WORLD representation. It is also used to form the communication to the user interacting with the program.

Notice that in Fig. 8 the ORDER slot does not include all slot names in the frame. Those that are not in ORDER are defined as descriptive. Slot names are also defined as descriptive if a frame does not contain an ORDER slot. For example, all slot names of the BOX frame in Fig. 1 as well as those of the NP frame in Fig. 5 are descriptive. Descriptive slots express declarative facts of a concept's properties.

In summary, the purpose of introducing an ORDER slot is to incorporate procedural knowledge into declarative data structures, giving our frame representation efficiency. The procedural knowledge specifies how to utilize the information in the frames. It integrates different levels of knowledge such as providing guidance of inferences, setting up responses, forming actions, etc. It also deals

```
(DECL-SENT
 (SUBJECT (NP
          (SYNTAX (NPF
                   (DET A)
                   (ADJ (BIG RED))
                   (NOUN BOX)
                   (ORDER NOUN ADJ DET)))
          (SEMANTICS (BOX
                      (SIZE BIG)
                      (COLOR RED)
                      (CONTAINS NIL)
                      (LOCATION ((POINTER (TABLE & & &))
                                 (RELATION ON)))))))
 (SUBJECT-SEARCH ((RESULT NONE) (POINTER NIL)))
 (OBJECT (NP
          (SYNTAX (NPF
                   (DET A)
                   (ADJ (SMALL))
                   (NOUN TABLE)
                   (ORDER NOUN ADJ DET)))
          (SEMANTICS (TABLE
                      (SIZE SMALL)
                      (SUPPORTS ((BOX & & & &)))
                      (LOCATION ((POINTER (ROOM &))
                                 (RELATION IN)))))))
 (OBJECT-SEARCH ((RESULT NONE) (POINTER NIL)))
 (PREDICATE ((VERB BE) (RELATION ON)))
 (REPLY 3)
 (ORDER PREDICATE REPLY))
```

Fig. 8 Representation of the declarative sentence

A BIG RED BOX IS ON A SMALL TABLE

with different types of the fillers. The ORDER slot furthermore serves as the top level control of a frame, i.e. it specifies the order of using the information.

## 5.4 Syntactic Knowledge

In order to obtain the representations of phrases and sentences described in section 5.2, the system makes use of the explicit ordering of words in an English sentence. The syntactic knowledge is represented in the form of a list of patterns corresponding to the types of input sentences, as shown in Fig. 9. This analysis is similar to the Augmented Transition Network approach.

In the current representation a pattern consists of a pattern name together with a set of states. For example, the pattern name of the first pattern in Fig. 9 is DECLARATIVE. There are five states in the DECLARATIVE pattern. A state contains a state number as well as one or more choices. A choice consists of a procedure together with a number "next" specifying the next state to be processed within the pattern. The procedure appearing in a choice is used for recognizing the current leftmost word of a sentence and for keeping such a word someplace (in one of the frames), provided the syntactic definition of the word satisfies the condition specified by the procedure. For example, given the procedure ?ADJ and the word RED with its syntactic feature ADJECTIVE being kept in the dictionary,

```
((DECLARATIVE (1 (?NPS 2))
              (2 (IS 3))
              (3 (?VERB 4) (?P 5) (?ADJ 6) (?NPO 6))
              (4 (?P 5))
              (5 (?NPO 6)))
 (THERE (1 (THERE 2))
        (2 (IS 3))
        (3 (?NP-A 4))
        (4 (?P 5))
        (5 (?NPO 6)))
 (IS (1 (IS 2))
     (2 (THERE 4) (?NPS 3))
     (3 (?P 5) (?ADJ 6))
     (4 (?NP-A 7))
     (5 (?NPO 6))
     (7 (?P 5)))
 (WHAT (1 (WHAT 2)) (2 (IS 3)) (3 (?P 4)) (4 (?NPO 5)))
 (WHERE (1 (WHERE 2)) (2 (IS 3)) (3 (?NPS 4)))
 (WHICH (1 (WHICH 2))
        (2 (?NP-N 3))
        (3 (IS 4))
        (4 (?P 5) (?ADJ 6))
        (5 (?NPO 6)))
 (NP (1 (?DET 2)) (2 (?ADJ 2) (?NOUN 3)))
 (NP-A (1 (A-AN 2)) (2 (?ADJ 2) (?NOUN 3)))
 (NP-N (1 (?ADJ 1) (?NOUN 2))))
```

Fig. 9 Sentence patterns

?ADJ accepts the word RED. The procedure occasionally refers to another pattern in the list. For example, given ?NPS and the phrase THE RED BALL, ?NPS transfers its control to the NP pattern. The NP, NP-A, and NP-N patterns are defined as a noun phrase with a definite determiner, with an A (AN) determiner, and without any definite determiner, respectively.

When a pattern is used to parse a sentence, the state 1 is processed first. The next state to be processed is indicated by a number in a choice provided the choice procedure succeeds in recognizing the leftmost word or a syntactic group of words of the sentence. In case the choice procedure fails, other choices at the same state are attempted. If none of the choice procedures succeeds, then the sentence is not acceptable, i.e. the system cannot understand the sentence at the present time. If a state referred by a number does not exist, it is defined as an accepting state. The state 6 in the DECLARATIVE pattern, for example, is the accepting state for a declarative sentence. The sentence is grammatically accepted if an accepting state is entered and also the sentence is ended.

For example, the first pattern (DECLARATIVE) in the list is used to parse the sentence on line 4 of the sample dialogue (p. 12). The pattern indicates that the sentence should begin with a noun phrase (?NPS stands for a noun phrase

which is the subject of the sentence) followed by the word IS, and IS is followed by a verb, a preposition, an adjective, or another noun phrase (?NPO stands for a noun phrase which is the object of the sentence). In case a verb follows IS, then the subsequent word may possibly be a preposition, and the preposition is followed by another noun phrase. If an adjective follows IS, then the sentence is supposed to be ended. Other patterns in the list are interpreted in a similar way.

## 5.5 Inferential Knowledge

In the program's domain of discourse inferences are made through either one of two perspectives on the knowledge base: the WORLD and HISTORY representations.

WORLD is a network of physical object frames with their relationships. Each frame corresponds to an object described by the user. The network represents the current state of the environment. Its information is updated as a new event is introduced. In other words, it is an abstract model of the contents of the conversation.

HISTORY is the program's memory of the conversation. It is an ordered list of previous sentence frames. It indicates a sequence of events described by the user. Individual objects mentioned in the sentences share structures

with the world objects. This suggests that HISTORY also represents the program's world. In effect WORLD grows or is being modified as a result of adding information to HISTORY.

With the introduction of these two perspectives, the program identifies individual objects in a similar way to that of a human. When a person uses an indefinite article "a" for the concept of "ball", for example, he may refer to any one of the balls in his environment. The program therefore looks at the current world for any one of the balls. When a person uses the definite article "the", he possibly refers to a particular ball in his environment. The ball usually corresponds to the one in his memory, i.e. he has already seen or heard of the ball. The program in this case looks at its history (memory) for that particular ball. In case there is more than one ball in HISTORY, the one most recently recorded is meant.

There are three types of inferences. The first type is to identify an object mentioned in an input sentence from the objects either in WORLD or in HISTORY, depending upon whether the definite determiner in a noun phrase is A (AN) or THE. In case it is A (AN), the description of some object is retrieved from WORLD for matching. The retrieval process is based on the name of a physical object frame. A match succeeds when the current object's structural

properties are a subset of those of the retrieved one. On the other hand, if the definite determiner is THE, then the description of some object is retrieved from the SUBJECT or OBJECT slot of the previous declarative sentence frames stored in HISTORY, according to a "last in first out" order, i.e. the most recent declarative sentence is examined first. The retrieval process is based on the name of a sentence frame, i.e. DECL-SENT provides such information. Matching is similar to the A (AN) case. The search result for the A (AN) case is NONE or SIMILAR and for the THE case is FOUND or NOTFOUND. This result is kept in the RESULT sub-slot of either the SUBJECT-SEARCH or the OBJECT-SEARCH slot of a sentence frame.

The second type of inference is to detect whether the currently described event is possible. It deals mainly with the existence of a specific functional property slot. For example, the sentence THE GREEN BOX IS PUT IN THE SMALL TABLE on line 46 of the sample dialogue would not cause the system to carry out any action because there is no CONTAINS slot in the TABLE frame.

The third type of inference checks whether the current event has been previously described. It utilizes a "brute-force" search. For each existing physical object which matches with the subject of a sentence, its LOCATION slot is examined to see if the location of the physical object matches with the object of the sentence. Because of

the nature of the program there cannot be more than one match (see p. 23). Although the search method used is "brute-force", it still represents a significant improvement in the level of understanding by the machine.

## Section 6. Implementation

The ROOM program is written in UCI LISP and run under the PDP-10 time sharing system at the University of Oregon. The program consists of the following components:

1. A set of stored data containing a small dictionary, a number of frames, sentence patterns, and partial responses.
2. A top-level control communicating with the user and monitoring processes.
3. A syntactic analysis component for processing an input string.
4. A semantic framework for representing physical objects and the meaning of sentences.
5. A set of inference procedures for making deductions.
6. An action-response structure for carrying out appropriate actions and for setting up correct responses.

### 6.1 Stored Data

The stored data are initially kept in LISP property lists under the property indicators GRAM, MEAN, or FRAME.

The grammatical features of the words being used during the conversation are stored on the property lists of the LISP atoms corresponding to the words, within the property

indicator GRAM. For example, TABLE is represented as:

```
(DEFPROP TABLE
 ((SYNT NOUN))
GRAM)
```

It indicates that the filler of the SYNT slot is NOUN and that the list is kept under GRAM. The same representation is used for other words.

This same notation is used with GRAM replaced by MEAN for the meanings of the words kept in MEAN. The meaning of the concept of TABLE is represented as follows:

```
(DEFPROP TABLE
 ((SUPPORTS NIL)
  (LOCATION ((POINTER NIL)
             (RELATION NIL))))
MEAN)
```

It indicates that a table can support other physical objects. The location of a table would be a pointer to another object and the relation of the table's location to the location of another object.

FRAME differs from GRAM or MEAN in that it keeps the sentence frames, the NP frame, and the NPF frame. Section 5.2 described these frames. Fig. 6, 5, and 4 show the further specified declarative sentence, NP, and NPF frames, respectively.

The sentence patterns are stored in the variable PATTERNS, which are in Fig. 9. The variable REPLIES keeps a list of partial responses to be retrieved and used to form responses to the user. Fig. 10 shows two of the partial responses.

```
(2
 (OR (AND (GREATERP (LENGTH PASS-TO-REPLY) 2)
          (FORM-ANSWER PASS-TO-REPLY))
     (CDR
      (MAPCAN (FUNCTION (LAMBDA (W) (APPEND (QUOTE (AND A)) W)))
              PASS-TO-REPLY))))

(21
 (LIST (QUOTE "I DON'T KNOW WHICH")
       PASS-TO-REPLY
       (QUOTE "YOU ARE TALKING ABOUT")))
```

Fig. 10 REPLIES 2 and 21

## 6.2 Top-level Control

There are three procedures to perform the verbal communications: SAY, LISTEN, and CONVERSE. SAY translates a response into a readable form and sends it to the user. LISTEN takes in whatever the user describes to the program. CONVERSE manages to communicate to the user through SAY, to accept an arbitrary string through LISTEN, to send the input string to the procedure CONTROL, or to be used for debugging in showing the current WORLD or HISTORY representation.

The procedure CONTROL receives an input sentence from CONVERSE, looks at the first word of the sentence to make a guess what kind of sentence it is, evaluates the corresponding sentence frame when its slots are filled, and keeps the frame in HISTORY of the conversation.

To select an appropriate pattern and the frame, CONTROL matches the first word of a sentence against the names of the patterns. In case there is no match, CONTROL selects the DECLARATIVE pattern and the DECL-SENT frame; otherwise, the matched pattern and the corresponding frame are chosen.

## 6.3 Syntactic Analysis

The syntactic analysis is controlled by a LISP function SENT-ANALYSIS. Before parsing a sentence it receives a specific proposed pattern and a sentence frame from the procedure CONTROL. Syntactically, a sentence

A BIG RED BALL IS ON A BIG TABLE is processed as the following, given the patterns as in Fig. 9.

SENT-ANALYSIS enters at state 1 and finds that the next word belongs to a noun phrase, therefore it branches to the NP pattern. At state 1 of NP it expects the first word to be a definite determiner, thus the procedure ?DET is evaluated to check for the word as a definite determiner. In this example the word A is a determiner. Next, SENT-ANALYSIS enters state 2 of NP to look for either an adjective or a noun. In case an adjective appears, it repeats, looking for another adjective or a noun. On the other hand, if the definite determiner is followed by a noun, then it checks the noun and returns back to whatever called NP.

The example shows that the next word BIG happens to be an adjective. ?ADJ is called to do a similar check as in the determiner case, but in a somewhat different manner. For the determiner case the program needs only to know whether the word is A, AN, or THE. But for an adjective it has to check with the word's syntactic feature stored in GRAM. The adjective RED is treated as the same way as the word BIG. To this end SENT-ANALYSIS recognized the the first three words of the sentence of which one is a definite determiner and the others are adjectives. As before, it looks at the subsequent word. But the next word

BALL happens to be a noun, so ?ADJ fails but ?NOUN succeeds and returns back to state 1 of DECLARATIVE to resume processing there for the remainder of the sentence according to the DECLARATIVE pattern. Consequently, SENT-ANALYSIS accepts the entire sentence.

## 6.4 Semantics

The semantic framework of the program is used to represent the meaning of a sentence processed by the syntactic analysis. To represent the meaning of a sentence the program has to understand in the first place the meaning of the pieces of information described in a sentence. This in turn triggers off processing the constituents made up of the noun phrases, the verbs, or the prepositions.

The representation of the meaning of a noun phrase is accomplished once the noun phrase has been recognized as syntactically correct. The words of a noun phrase are filled into the appropriate slots in the NPF frame by the procedures ?DET, ?ADJ, or ?NOUN. Consider the noun phrase A BIG RED BALL and go back to the procedure ?NOUN mentioned in SENT-ANALYSIS to realize the entire process.

?NOUN calls another procedure SLOT-EVAL after the word BALL is filled in the NOUN slot. SLOT-EVAL evaluates the procedural slots in the NPF frame with the order specified in the ORDER slot (see page 24-25 above). Therefore, NOUN

is called to create the physical object frame which is an instance of the prototype BALL frame kept under MEAN on the property list of the atom BALL. ADJ is next evaluated to insert the structural properties such as color and size into the BALL frame by calling the procedure INSERT-IN-POF. The physical object frame of A BIG RED BALL is shown in Fig. 11. Notice that POINTER as well as RELATION in the BALL frame are NIL. They will not be filled until the

```
(BALL
 (SIZE BIG)
 (COLOR RED)
 (LOCATION ((POINTER NIL)
            (RELATION NIL))))
```

Fig. 11 Representation of the physical object A BIG RED BALL

procedure PREDICATE is evaluated (see page 26-27 above). The procedure DET in the NPF frame is used for inference purpose. The process described completes a noun phrase's syntactic and semantic representation in part. The NPF and BALL frames are put in the NP frame, and the NP frame is in turn inserted into the SUBJECT or OBJECT slot in the instance of a sentence frame.

The object phrase of a sentence is treated in the same way as the subject phrase. But the verb and preposition are filled into the PREDICATE slot directly. Representation of the meaning of a sentence is completed once the procedural slots are evaluated.

## 6.5 Action-response

To check whether a computer system understands a sentence we have to examine its action and response corresponding to the sentence. ROOM generates various types of partial responses in the process of inferencing, carrying out an appropriate action, or answering a questions. It associates each type of the partial responses with a distinct number which is filled in the REPLY slot of a sentence frame. The procedure REPLY uses the number to retrieve the data kept in the REPLIES list to form a complete response. A partial response usually is a partial noun phrase translated from an internal representation, i.e. there is no definite determiner in such a noun phrase.

The program utilizes the procedure ZOOM to translate the internal representation of a physical object or a list of physical objects into a partial noun phrase. For example, given the following BALL frame, ZOOM produces the expression "(SMALL GREEN BALL)".

```
(BALL
 (SIZE SMALL)
 (COLOR GREEN)
 (LOCATION ((POINTER (ROOM &))
            (RELATION IN))))
```

ROOM's actions involve the creation of physical objects in the room or the movement of physical objects. It does not carry out an action for each sentence since the action of a sentence may be either previously described or physically impossible. Therefore, the program makes use of its reasoning ability to avoid redundancy.

To create a new object in the room the program simply adds it to the WORLD representation and has the new object's location pointing to WORLD and relation IN. Moving an object from one location to another is somewhat different. Computationally, the program traces back to the object's previous location to remove the object from one of the functional properties specified in its RELATION. POINTER in the object's LOCATION slot is made to point to a new location. An appropriate preposition is used for RELATION.

There are currently a number of predicates in the program such as ISON, ISIN, PUTON, and PUTIN corresponding to the types of actions. A distinct procedure manipulates each of these. The program generates such a procedure when the

PREDICATE slot is being processed. The procedure ISON is very similar to the procedure ISIN. In fact, they could be combined and share codes by passing arguments such as "supports" (or "contains") and "on"(or "in"). The reason of not doing that is to make each predicate serve as its own module so that when new predicates are introduced such as ROLLOFF and BOUNCEOFF, we do not have to worry how they would interact with others. We simply embed the knowledge into the procedures.

Consider the sentence A GREEN BALL IS IN A SMALL BOX, on line 30 of the sample dialogue, to see how ISIN is formed. The PREDICATE slot for the sentence is shown in Fig. 12.

```
(PREDICATE ((VERB BE)
            (RELATION IN)))
```

Fig. 12 ISIN

PREDICATE is evaluated to form the LISP expression "(BE IN)", and it evaluates this expression. BE looks up both the SUBJECT-SEARCH and OBJECT-SEARCH in the sentence frame in order to decide whether the expression ISIN needs to be formed. The word ISIN is produced if a sentence is of the form "AN object IS IN AN object" or "AN object IS IN THE object". The reasons for this are that the verb IS is

for existential purposes and that a sentence does not have an action if it is of the form "THE object IS IN AN object" or "THE object IS IN THE object". In the former case ROOM tells the user the actual location of the first "THE object", and in the latter case ROOM simply tries to find the "truth" of the sentence.

Once the procedure ISIN is formed it is evaluated. ISIN checks if a ball can be in a box (see section 5.5) and also if the meaning of the sentence has been previously described. In the context of the dialogue the event has not been introduced before. Therefore ISIN adds these two objects to WORLD and establishes the relationships between them.

## 6.6 Inferences

In section 5.5 it was said that the program has three types of inferences. A definite determiner in the noun phrase of a declarative or a question sentence initiates the program into the process of searching either WORLD or HISTORY for a similar or a mentioned object, respectively. This type of inference is defined as "identification".

The entire process of identification of an object is through the uses of the procedures DET, SEARCH-WORLD or SEARCH-HISTORY, and MATCH. DET is called in the first place to check the type of the definite determiner in a

NPF frame. The process of a WORLD search begins in the presence of an A or AN determiner in the DET slot in the NPF frame and with the procedure SEARCH-WORLD.

SEARCH-WORLD looks through the current WORLD for finding a physical object similar to the current one described in the SEMANTICS slot in the NP frame. In case no similar object is discovered, SEARCH-WORLD returns NONE as the search result. Otherwise, MATCH is brought in to check the structural properties of the two physical objects in order to find a match.

MATCH returns NIL provided the current object has more structural properties than the world object or one of the properties of the two objects does not match.

SEARCH-WORLD is no longer evaluated in case T is returned by MATCH, and it returns the search result as SIMILAR together with a temporary pointer to the similar world object. In case NIL is returned, SEARCH-WORLD is repeated from the name following the one just examined in the world objects list.

The procedure SEARCH-HISTORY is called by DET when the definite determiner THE appears in the NPF frame. The way to find a mentioned object in SEARCH-HISTORY is similar to the way to find a similar object in SEARCH-WORLD. They are different in two respects. SEARCH-HISTORY works on either both the subject and object or one of them in some previous declarative sentences. SEARCH-HISTORY returns the search

result as FOUND or NOTFOUND instead of SIMILAR or NONE in SEARCH-WORLD.

SEARCH-HISTORY starts with a sentence retrieved from the HISTORY representation. The retrieval of a sentence from HISTORY is performed in a "last in first out" order. Only the previous declarative sentence frames are used.

Then, it works in the manner described in SEARCH-WORLD to find the mentioned object in the SUBJECT slot of the frame of the last entry in HISTORY. If it is not found in the SUBJECT slot, SEARCH-HISTORY examines the OBJECT slot. If the object is not found in the last entry in either slot of HISTORY, SEARCH-HISTORY examines the next to last entry's frame and so on recursively. If the object is not found in HISTORY at all, the program indicates this fact by the REPLIES 21 of Fig. 10, page 37 above.

The second type of inference is to detect a situation which cannot exist. It utilizes the procedure INFER-BY-PROP. INFER-BY-PROP notices the type of the functional property corresponding to the given preposition and looks for the same slot in the object frame under consideration. Failure to find such a slot results in the termination of the entire process. CONVERSE gives an error message to the user. For example, the sentence THE GREEN BOX IS PUT IN THE SMALL TABLE on line 46 of the dialogue results in an error message.

The third type of inference makes use of the outcomes of the identification process and finds out whether the current event has been previously described. The search result of an individual physical object in "identification" results in one of the four possible cases, i.e. NOTFOUND, FOUND, NONE, or SIMILAR. There are sixteen possible combinations. The third type of inference is concerned only with the combination of SIMILAR and SIMILAR. Two procedures are used: BOTH-SIMILAR and SIMILAR-OBJS.

SIMILAR-OBJS once again searches WORLD for all similar objects with respect to the one encountered, and it returns a list of the similar objects (with addresses and contents). BOTH-SIMILAR takes such two lists--one for the subject and the other for the object of the sentence. BOTH-SIMILAR checks if the location of a physical object in the subject list is a member of the object list. In case it is, BOTH-SIMILAR returns NIL as the result, which indicates the current event has been described before.

For example, suppose the system receives the following sequence of three sentences:

1. A RED BALL IS ON A TABLE
2. A BALL IS ON A RED TABLE
3. A RED BALL IS ON A TABLE

SIMILAR-OBJS returns the subject list of A RED BALL and the object list of A TABLE and A RED TABLE. BOTH-SIMILAR then

finds out that the location of A RED BALL is the same as A TABLE, which is a member of the object list. Therefore, BOTH-SIMILAR returns NIL. ROOM sets up the response I KNOW.

## 6.7 Interactions

The previous sections described in detailed way the representations of the different types of knowledge and their uses. There is a syntactic component of parsing an input and of expecting the subsequent word with possibilities; a semantic framework of forming the meaning of concepts; the inferential knowledge of judging the current environment; an action-response element embedded in action type procedures. But the system will not accomplish the task if these parts are not treated as a whole. Consider the following example and observe how ROOM operates in an integrated manner.

Suppose ROOM has received the following two sentences:

1. A RED BALL IS ON A RED TABLE
2. A BIG BALL IS ON A BIG TABLE

ROOM processes the new sentence THE RED BALL IS PUT ON THE BIG TABLE as follows.

It first parses the noun phrase and forms a physical object frame for THE RED BALL. Because of the presence of the definite determiner THE which implies the current object has been mentioned, the identification process takes place to look for the most recent RED BALL from HISTORY; as a

result RED BALL is discovered and FOUND is put in RESULT in the SUBJECT-SEARCH slot. Since the noun phrase has been processed, the subsequent word that SENT-ANALYSIS expects is the verb IS, according to the DECLARATIVE pattern. In fact, IS is recognized and it is kept in VERB in the PREDICATE slot. Again, the program returns back to the pattern. The next word is another verb PUT, therefore PUT replaces the former verb.

The rest of the sentence is processed and accepted in the same way. To this end the sentence frame has all slots filled. The remaining task is to evaluate all those procedural slots.

Thus, PREDICATE is the first one to be evaluated. It returns the LISP expression "(PUT ON)" and evaluates the expression. PUT first finds out whether RED BALL can be put on BIG TABLE. In this example, the condition results in T because of TABLE's SUPPORTS slot. PUT also checks if BIG TABLE supports RED BALL. If not, PUT forms the procedure PUTON to carry out the action, resetting appropriate pointers and relations.

In general, ROOM goes back and forth to understand an input sentence using its different types of knowledge in an integrated manner, which is similar to the human way of understanding.

## 6.8 Questions Answering

Questions answering of a natural language understanding system appears to be less intricate than the modification of a system's data base. A system in general retrieves the assertions which have already been made by a programmer or throughout a conversation. In other words, the complexity of the reasoning is greatly reduced.

ROOM can answer several types of questions such as WHAT, WHERE, IS, and WHICH. There is a specific question frame associated with each of these. The slots included in the frame are contingent upon the type.

The way of answering questions in the current program is simplified because of a limited set of predicates being used. This is done by considering only the information related to a question, i.e. the important role played by verbs is neglected at the present time.

The question WHAT IS IN THE ROOM on line 12 of the dialogue is represented in Fig. 13. The WHAT frame does not include the SUBJECT or SUBJECT-SEARCH slot as appears in the declarative sentence frame (see Fig. 8). It consists of the OBJECT, OBJECT-SEARCH, PREDICATE, REPLY, and ORDER slots. The ORDER slot specifies the order of evaluation of the WHAT frame to be OBJECT-SEARCH and REPLY. The procedure OBJECT-SEARCH is used to form a procedure name corresponding to the current type. In this case it forms QUESTION-WHAT.

```
(WHAT
 (OBJECT
  (ROOM
   (CONTAINS
    ((TABLE (SIZE SMALL) (SUPPORTS &) (LOCATION &))
     (BALL (COLOR BLACK) (LOCATION &))
     (BOX (COLOR RED) (CONTAINS &) (LOCATION &))))))
 (OBJECT-SEARCH ((RESULT FOUND) (POINTER (ROOM &))))
 (PREDICATE ((VERB BE) (RELATION IN)))
 (REPLY 2)
 (ORDER OBJECT-SEARCH REPLY))
```

Fig. 13 Representation of the question WHAT IS IN THE ROOM on line 12 of the dialogue

QUESTION-WHAT first obtains the functional slot from the object mentioned in a question, according to the preposition. It then calls ZOOM to translate the list of objects within the functional slot into an expression which can be used by REPLY to form the answer for a WHAT type question. In this example, QUESTION-WHAT gets the CONTAINS slot of ROOM and calls ZOOM to obtain the partial response:

((SMALL TABLE) (BLACK BALL) (RED BOX))

REPLY combines the partial response with another partial response (REPLIES 2) to form the answer to the question, given on line 13 of the dialogue.

The question WHERE IS THE BOX on line 14 of the dialogue is represented in Fig. 14. The WHERE frame consists

```
(WHERE
 (SUBJECT (BOX
           (COLOR RED)
           (CONTAINS ((BALL & &)))
           (LOCATION ((POINTER (TABLE & & &))
                      (RELATION ON)))))
 (SUBJECT-SEARCH ((RESULT FOUND)
                  (POINTER (BOX & & &))))
 (PREDICATE ((VERB BE) (RELATION NIL)))
 (REPLY 6)
 (ORDER SUBJECT-SEARCH REPLY))
```

Fig. 14 Representation of the question WHERE IS THE BOX on line 14 of the dialogue

of the SUBJECT, SUBJECT-SEARCH, PREDICATE, REPLY, and ORDER slots. The way of answering a WHERE type question is similar to the way of answering a WHAT type question. But QUESTION-WHERE differs from QUESTION-WHAT in that it looks for the LOCATION slot of an object rather than one of the functional slots. In this example, QUESTION-WHERE obtains BOX's LOCATION slot and calls ZOOM to form the partial response: (ON THE (SMALL TABLE))

The IS frame has the same structure as the declarative sentence frame. To answer an IS type question the program performs the same way as in a declarative sentence, but with the conversion of a response to an answer. In addition, in the latter case ROOM sometimes creates objects in the room if the event has not been mentioned earlier or otherwise looks for the correctness of a sentence. However, in the case of a question there is no creation of objects in the room at all.

The procedure TRANSLATE converts the number in the REPLY slot into another number corresponding to an answer for a specific IS type question. For example, the user may initially describe to the system the following three sentences. Assuming that the third sentence is similar to the environment of the question on line 26 of the dialogue, replies to the following three sentences are, for the first sentence OK and

1. A RED BOX IS ON A SMALL TABLE
2. A BIG BLACK BALL IS IN A GREEN BOX
3. IS THERE A BALL IN THE RED BOX

for the second sentence also OK. As for the third one, since it begins with the word IS, the IS frame is used rather than the DECL-SENT frame. TRANSLATE further converts the reply number of OK into the reply number of NO, which is an ultimate response to the question.

If the environment is not the same, the reply is different. For example, if a red box had already been put on the small table, the response to sentence 1 would be I KNOW. If a ball had already been described as being in the red box, the answer to sentence 3 would be YES.

The reasons for not using the DECL-SENT frame for IS questions are that the procedures can tell whether the creation of objects is necessary and the procedure SEARCH-HISTORY knows which sentence frame is to be retrieved and used.

The prototype WHICH frame differs from WHAT, WHERE, or IS, as shown in Fig. 15. The slots in the frame are more or less the same as those in the IS and DECL-SENT, but the ORDER slot is like the one in the WHAT frame. The reasons for this are that in the current version of the program a WHICH question may have either subject and object or simply a subject, and the subject of a WHICH question does not initially refer to any object in the room because there is no definite determiner in the noun phrase. Most information to the question is in the object. Therefore, OBJECT-SEARCH is defined as a procedure to use the search information of the object.

The program begins with the evaluation of the OBJECT-SEARCH slot. If RESULT of the OBJECT-SEARCH slot is NIL, then the question pattern is possibly to be

```
(WHICH
 (SUBJECT NIL)
 (SUBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
 (OBJECT NIL)
 (OBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
 (PREDICATE ((VERB NIL) (RELATION NIL)))
 (REPLY NIL)
 (ORDER OBJECT-SEARCH REPLY))
```

Fig. 15 The prototype WHICH frame

WHICH ?NP-N IS ?ADJ (see section 5.4). ROOM searches WORLD for an object similar to the one in the ?NP-N, with the augmented property described by the ?ADJ. For example, after the question WHICH BALL IS BLACK on line 36 of the dialogue, ROOM searches for the black ball in the room and returns its location. On the other hand, in case RESULT of the OBJECT-SEARCH slot is not NIL, ROOM looks for a similar object from the functional slot of ?NPO (see section 5.4), according to the preposition in the question. For example, after the question WHICH BALL IS IN THE BOX on line 16 of the dialogue, ROOM searches a ball from the CONTAINS slot of BOX and returns it structural properties, i.e. BLACK.

## Section 7. Remaining Problems

A large additional effort is needed on one hand to bring the representation to a state which makes it powerful and on the other hand to bring the performance of the ROOM program to a level which is comparable to human behavior in the same environment.

The use of an "answer" frame could possibly enable the system to be flexible for generating responses under different circumstances. The scheme of "memory by description" would also be used extensively to reduce the search space to some extent.

Another important direction of our research in the future is to enable the system to have the capability of defining a new concept, given another concept and some similar properties. For example, the user may happen to mention the unknown concept of TRASH CAN, then the system is expected to have the ability of defining such a concept through the following conversation:

ROOM: WHAT IS A TRASH CAN?

USER: IT IS AN OBJECT.

ROOM: CAN YOU TELL ME MORE ABOUT IT?

USER: IT IS LIKE A BOX, BUT IT IS ROUND.

As a result, the description for the concept of TRASH CAN is established--shape and BOX's properties.

We are also aiming toward representing the meaning of some difficult action verbs, in the form of procedures. For example, ROLL or BOUNCE requires the system to have the knowledge of time events in addition to the knowledge of the locations of physical objects.

As we go on, many additional issues will be raised. For each solution there probably will be a set of other problems related to it. We hope that the current status of the ROOM program will provide us with a broader view on the problems and with a powerful tool for attacking them.

## Acknowledgements

The author is grateful to the support for this work received from the Computer Science Department at Oregon State University and the computing facilities provided by the University of Oregon. Specifically, the author would like to express his gratitude to his adviser, Vesko Marinov, for bringing up the problem and for giving numerous suggestions, comments, remarks, and tremendous assistance. The author is also grateful to his minor professor, Harry Goheen, for correcting and furnishing a great deal of remarks on the initial draft. This paper will hardly have been completed without them.

Furthermore, the author would like to thank the following people: Bill Bregar and Chi Poon, for being the co-workers of the first version of the ROOM program; Bob Eifrig and Sum Lau, for supplying comments on the first version of the program; his honored parents and his elder brother Hung Shih, for always.

# Appendix--Listing of the ROOM program

The ROOM program is listed in the following order:

1. STORED-DATA, page 61-66.
2. IO-FUNCTIONS, page 67-70.
3. SENTENCE-ANALYSIS, page 71-81.
4. WORLD-FUNCTIONS, page 82-114.
5. QUESTIONS, page 115-120.

For each FUNCTION written, the following information is given according to the order:

1. FUNCTION type--FEXPR is explicitly indicated.
2. Specification of arguments passed in a FUNCTION.
3. The function of each FUNCTION.

```
(DEFPROP STORED-DATA
 (STORED-DATA GENERATED
              BALL
              BIG
              BLACK
              BOX
              BROWN
              GREEN
              IN
              IS
              ON
              PUT
              RED
              ROOM
              SMALL
              TABLE
              SENTENCE-FRAME
              NP
              NPF
              NP-N
              PATTERNS
              REPLIES)
VALUE)

NIL
*
```

```
(DEFPROP GRINPROPS
 (NIL FRAME GRAM MEAN EXPR FEXPR MACRO VALUE SPECIAL)
VALUE)

(DEFPROP BALL
 ((SYNT NOUN))
GRAM)

(DEFPROP BALL
 ((LOCATION ((POINTER NIL) (RELATION NIL))))
MEAN)

(DEFPROP BIG
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP BIG
 ((SIZE BIG) (APPLY-TO (BALL BOX TABLE)))
MEAN)

(DEFPROP BLACK
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP BLACK
 ((COLOR BLACK) (APPLY-TO (BALL BOX TABLE)))
MEAN)

(DEFPROP BOX
 ((SYNT NOUN))
GRAM)

(DEFPROP BOX
 ((CONTAINS NIL) (LOCATION ((POINTER NIL) (RELATION NIL))))
MEAN)

(DEFPROP BROWN
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP BROWN
 ((COLOR BROWN) (APPLY-TO (BALL BOX TABLE)))
MEAN)

(DEFPROP GREEN
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP GREEN
 ((COLOR GREEN) (APPLY-TO (BALL BOX TABLE)))
MEAN)

NIL
*
```

```
(DEFPROP IN
 ((SYNT PREPOSITION))
GRAM)

(DEFPROP IS
 ((SYNT VERB))
GRAM)

(DEFPROP ON
 ((SYNT PREPOSITION))
GRAM)

(DEFPROP PUT
 ((SYNT VERB))
GRAM)

(DEFPROP RED
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP RED
 ((COLOR RED) (APPLY-TO (BALL BOX TABLE)))
MEAN)

(DEFPROP ROOM
 ((SYNT NOUN))
GRAM)

(DEFPROP ROOM
 ((CONTAINS NIL))
MEAN)

(DEFPROP SMALL
 ((SYNT ADJECTIVE))
GRAM)

(DEFPROP SMALL
 ((SIZE SMALL) (APPLY-TO (BALL BOX TABLE)))
MEAN)

(DEFPROP TABLE
 ((SYNT NOUN))
GRAM)

(DEFPROP TABLE
 ((SUPPORTS NIL) (LOCATION ((POINTER NIL) (RELATION NIL))))
MEAN)

NIL
*
```

```
(DEFPROP SENTENCE-FRAME
 ((DECL-SENT
   ((SUBJECT NIL) (SUBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (OBJECT NIL)
                  (OBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (PREDICATE ((VERB NIL) (RELATION NIL)))
                  (REPLY NIL)
                  (ORDER PREDICATE REPLY)))
  (WHERE
   ((SUBJECT NIL) (SUBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (PREDICATE ((VERB NIL) (RELATION NIL)))
                  (REPLY NIL)
                  (ORDER SUBJECT-SEARCH REPLY)))
  (WHAT
   ((OBJECT NIL) (OBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                 (PREDICATE ((VERB NIL) (RELATION NIL)))
                 (REPLY NIL)
                 (ORDER OBJECT-SEARCH REPLY)))
  (WHICH
   ((SUBJECT NIL) (SUBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (OBJECT NIL)
                  (OBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (PREDICATE ((VERB NIL) (RELATION NIL)))
                  (REPLY NIL)
                  (ORDER OBJECT-SEARCH REPLY)))
  (IS
   ((SUBJECT NIL) (SUBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (OBJECT NIL)
                  (OBJECT-SEARCH ((RESULT NIL) (POINTER NIL)))
                  (PREDICATE ((VERB NIL) (RELATION NIL)))
                  (REPLY NIL)
                  (ORDER PREDICATE REPLY))))
FRAME)

(DEFPROP NP
 ((SYNTAX NIL) (SEMANTICS NIL))
FRAME)

(DEFPROP NPF
 ((DET NIL) (ADJ NIL) (NOUN NIL) (ORDER NOUN ADJ DET))
FRAME)

(DEFPROP NP-N
 ((ADJ NIL) (NOUN NIL) (ORDER NOUN ADJ))
FRAME)

NIL
*
```

```
(DEFPROP PATTERNS
 (PATTERNS (DECLARATIVE (1 (?NPS 2))
                        (2 (IS 3))
                        (3 (?VERB 4) (?P 5) (?ADJ 6) (?NPO 6))
                        (4 (?P 5))
                        (5 (?NPO 6)))
           (THERE (1 (THERE 2))
                  (2 (IS 3))
                  (3 (?NP-A 4))
                  (4 (?P 5))
                  (5 (?NPO 6)))
           (IS (1 (IS 2))
               (2 (THERE 4) (?NPS 3))
               (3 (?P 5) (?ADJ 6))
               (4 (?NP-A 7))
               (5 (?NPO 6))
               (7 (?P 5)))
           (WHAT (1 (WHAT 2)) (2 (IS 3)) (3 (?P 4)) (4 (?NPO 5)))
           (WHERE (1 (WHERE 2)) (2 (IS 3)) (3 (?NPS 4)))
           (WHICH (1 (WHICH 2))
                  (2 (?NP-N 3))
                  (3 (IS 4))
                  (4 (?P 5) (?ADJ 6))
                  (5 (?NPO 6)))
           (NP (1 (?DET 2)) (2 (?ADJ 2) (?NOUN 3)))
           (NP-A (1 (A-AN 2)) (2 (?ADJ 2) (?NOUN 3)))
           (NP-N (1 (?ADJ 1) (?NOUN 2))))
VALUE)

NIL
*
```

```
(DEFPROP REPLIES
 (REPLIES
  (2
   (OR (AND (GREATERP (LENGTH PASS-TO-REPLY) 2)
            (FORM-ANSWER PASS-TO-REPLY))
       (CDR
        (MAPCAN (FUNCTION (LAMBDA (W) (APPEND (QUOTE (AND A)) W)))
                PASS-TO-REPLY))))
  (3 (LIST (QUOTE "OK")))
  (4
   (LIST (QUOTE "OK, ")
         (APPEND (QUOTE (THE))
                 (ZOOM OBJECT)
                 (QUOTE (NOW SUPPORTS THE))
                 (ZOOM SUBJECT))))
  (5 (LIST (QUOTE "I SEE")))
  (6 (LIST PASS-TO-REPLY))
  (7 (LIST (QUOTE "NOTHING")))
  (11 (LIST (QUOTE "YES")))
  (12 (LIST (QUOTE "THE ONE") PASS-TO-REPLY))
  (13 (LIST (QUOTE "THE") (ZOOM PHYSOB)))
  (14
   (LIST (QUOTE "OK, ")
         (APPEND (QUOTE (THE))
                 (ZOOM OBJECT)
                 (QUOTE (NOW CONTAINS THE))
                 (ZOOM SUBJECT))))
  (21
   (LIST (QUOTE "I DON'T KNOW WHICH")
         PASS-TO-REPLY
         (QUOTE "YOU ARE TALKING ABOUT")))
  (22 (LIST (QUOTE "I KNOW")))
  (23 (LIST (QUOTE "IN FACT") PASS-TO-REPLY))
  (24 (LIST (QUOTE "WELL, ") PASS-TO-REPLY))
  (25 (LIST (QUOTE "NO")))
  (26 (LIST (QUOTE "WHICH") (ZOOM SUBJECT)))
  (27
   (LIST (QUOTE "I DON'T THINK THE")
         (ZOOM OBJECT)
         PASS-TO-REPLY
         (ZOOM PHYSOB)))
  (30 (LIST (QUOTE "IT IS ALREADY THERE ")))
  (31 (LIST (QUOTE "YOU TOLD ME BEFORE")))
  (32
   (LIST
    (QUOTE
     "I CAN'T UNDERSTAND SUCH A SENTENCE AT THE PRESENT TIME"))))
VALUE)

NIL
*
```

```
(DEFPROP IO-FUNCTIONS
 (IO-FUNCTIONS CONVERSE LISTEN SAY REPLY ERRMSG FORM-ANSWER)
VALUE)

NIL
*
```

```
(DEFPROP CONVERSE
 (LAMBDA NIL
  (PROG (STRING REPLY HISTORY WORLD PASS-TO-REPLY)
        (SETQ REPLY (QUOTE (HELLO)))
        (SETQ WORLD (MAKE (QUOTE ROOM) (QUOTE MEAN)))
   A    (SAY)
        (LISTEN)
        (ERRSET
         (COND ((EQ (CAR STRING) (QUOTE BYE))
                (SETQ REPLY (QUOTE (THANK YOU)))
                (SAY)
                (RETURN))
               ((EQ (CAR STRING) (QUOTE E))
                (SPRINT (EVAL (CADR STRING)) 1)
                (SETQ REPLY (QUOTE (END OF FORM EVALUATION))))
               ((MEMQ (CAR STRING)
                      (QUOTE (A AN THE THERE WHERE WHICH WHAT IS)))
                (CONTROL (CAR STRING)))
               (T (REPLY 32))))
        (GO A)))
EXPR)

NIL
*
```

Top loop of the ROOM program.

```
(DEFPROP LISTEN
 (LAMBDA NIL
  (TERPRI)
  (PRINC (QUOTE "USER: "))
  (SETQ STRING (LINEREAD)))
EXPR)

NIL
*
```

Reads the input sentence and puts in STRING.

```
(DEFPROP SAY
 (LAMBDA NIL
  (TERPRI)
  (PRINC (QUOTE "ROOM:  "))
  (MAPC (FUNCTION
         (LAMBDA(W)
          (COND ((ATOM W) (PRINC W) (PRINC (QUOTE / )))
                (T
                 (MAPC (FUNCTION
                        (LAMBDA (Z) (PRINC Z) (PRINC (QUOTE / ))))
                       W)))))
        REPLY))
EXPR)

NIL
*
```

Writes the reply.

```
(DEFPROP REPLY
 (LAMBDA (N)
  (SETQ REPLY
        (COND (T (EVAL (CADR (ASSOC N REPLIES))))
              (T (EVAL (CADR (ASSOC 32 REPLIES)))))))
EXPR)

NIL
*
```

N - Reply number

Retrieves the reply from REPLIES according to the number.

```
(DEFPROP ERRMSG
 (LAMBDA (L) (SETQ REPLY (CONS (QUOTE **ERROR**) L)) (ERR NIL))
FEXPR)

NIL
*
```

FEXPR

L - Message

Prints the message and pops to the top loop.

```
(DEFPROP FORM-ANSWER
 (LAMBDA (X)
  (APPEND (MAPCAN (FUNCTION
                   (LAMBDA (A)
                    (APPEND (QUOTE (A)) A (LIST (QUOTE ", "))))))
                  (REMOVE (CAR (LAST X)) X))
          (QUOTE (AND A))
          (LAST X)))
EXPR)

NIL
*
```

X - A partial response with more than two noun phrases

Inserts commas between the noun phrases, etc.

```
(DEFPROP SENTENCE-ANALYSIS
 (SENTENCE-ANALYSIS CONTROL
                    SENT-ANALYSIS
                    ?NPS
                    ?NPO
                    ?NP-A
                    ?NP-N
                    PARSE-&-FILL
                    ?DET
                    A-AN
                    ?ADJ
                    ?NOUN
                    ?P
                    ?VERB
                    IS
                    THERE
                    WHAT
                    WHERE
                    WHICH)
VALUE)

NIL
*
```

```
(DEFPROP CONTROL
 (LAMBDA(OPENER)
  (PROG (F-INSTANCE SUBJECT OBJECT S-RESULT PHYSOB)
        ((LAMBDA(FRAME PATTERNS)
          (COND ((MEMQ OPENER (QUOTE (A AN THE THERE)))
                 (SETQ F-INSTANCE
                       (CONS (QUOTE DECL-SENT)
                             (CADR (ASSOC (QUOTE DECL-SENT) FRAME))))
                 (AND (OR (EQ OPENER (QUOTE THERE))
                          (SENT-ANALYSIS (CAR PATTERNS)))
                      (SENT-ANALYSIS (ASSOC OPENER PATTERNS))))
                (T (SETQ F-INSTANCE
                         (CONS OPENER (CADR (ASSOC OPENER FRAME))))
                   (SENT-ANALYSIS (ASSOC OPENER PATTERNS)))))
         (MAKE (QUOTE SENTENCE-FRAME) (QUOTE FRAME))
         PATTERNS)
        (COND ((EQ (FILLER F-INSTANCE REPLY) 32) (REPLY 32))
              (T (SLOT-EVAL F-INSTANCE)
                 (OR (GREATERP (FILLER F-INSTANCE REPLY) 20)
                     (BOOKKEEPING F-INSTANCE))))))
EXPR)

NIL
*
```

OPENER - The first word of an input sentence.

Monitors the entire understanding process.

```
(DEFPROP SENT-ANALYSIS
 (LAMBDA(TYPE)
  (PROG (STRUCTURE NEXT STATE CHOICE)
        (SETQ STRUCTURE (CDR TYPE))
        (SETQ NEXT 1)
   FOLLOW
        (SETQ STATE (ASSOC NEXT STRUCTURE))
        (AND (NULL STRING) (NULL STATE) (RETURN T))
        (AND (NULL STATE) (RETURN))
        (SETQ CHOICE (CDR STATE))
   NEWENTRY
        (AND (NULL CHOICE)
             (RETURN (FILL F-INSTANCE REPLY 32 RPLACD)))
        (AND (EVAL (CAR CHOICE)) (GO FOLLOW))
        (SETQ CHOICE (CDR CHOICE))
        (GO NEWENTRY)))
EXPR)

NIL
*
```

TYPE - A sentence pattern.

Parses an input string according to the sentence pattern.

```
(DEFPROP ?NPS
 (LAMBDA(X)
  (PARSE-&-FILL (QUOTE NP) (QUOTE SUBJECT) (QUOTE SUBJECT-SEARCH)))
EXPR)

(DEFPROP ?NPO
 (LAMBDA(X)
  (PARSE-&-FILL (QUOTE NP) (QUOTE OBJECT) (QUOTE OBJECT-SEARCH)))
EXPR)

(DEFPROP ?NP-A
 (LAMBDA(X)
  (PARSE-&-FILL (QUOTE NP-A) (QUOTE SUBJECT) (QUOTE SUBJECT-SEARCH)))
EXPR)

NIL
*
```

X - State number

Each FUNCTION calls PARSE-&-FILL to process a noun phrase.

```
(DEFPROP ?NP-N
 (LAMBDA(X)
  (SETQ NEXT X)
  (PROG (NPF-INSTANCE NP-INSTANCE POF)
        (SETQ NPF-INSTANCE (COPY (GET (QUOTE NP-N) (QUOTE FRAME))))
        (SENT-ANALYSIS (ASSOC (QUOTE NP-N) PATTERNS))
        (SETQ SUBJECT POF)
        (FILL-SENTENCE)
        (FILL F-INSTANCE SUBJECT NP-INSTANCE RPLACD))
  T)
EXPR)

NIL
*
```

X - State number

Processes a noun phrase without any definite determiner.

```
(DEFPROP PARSE-&-FILL
 (LAMBDA(NP-NAME SUBJ-OBJ S-RSLT)
  (SETQ NEXT X)
  (PROG (NPF-INSTANCE NP-INSTANCE POF)
        (SETQ NPF-INSTANCE (MAKE (QUOTE NPF) (QUOTE FRAME)))
        (SENT-ANALYSIS (ASSOC NP-NAME PATTERNS))
        (SET SUBJ-OBJ PHYSOB)
        (FILL F-INSTANCE (S-RSLT) RESULT S-RESULT FPLACD)
        (OR (MEMQ S-RESULT (QUOTE (NONE NOTFOUND)))
            (FILL F-INSTANCE (S-RSLT) POINTER PHYSOB RPLACD))
        (FILL-SENTENCE)
        (FILL F-INSTANCE (SUBJ-OBJ) NP-INSTANCE RPLACD))
  T)
EXPR)

NIL
*
```

NP-NAME - Name of a noun phrase pattern

SUBJ-OBJ - SUBJECT or OBJECT

S-RSLT - SUBJECT-SEARCH or OBJECT-SEARCH

Processes a noun phrase.

```
(DEFPROP ?DET
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((MEMQ (CAR STRING) (QUOTE (A AN THE)))
    (FILL NPF-INSTANCE DET (CAR STRING) RPLACD)
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

(DEFPROP A-AN
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((MEMQ (CAR STRING) (QUOTE (A AN)))
    (FILL NPF-INSTANCE DET (CAR STRING) RPLACD)
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

NIL
*
```

X - State number

?DET checks if the leftmost word is a definite determiner.

A-AN checks if the leftmost word is an indefinite article.

```
(DEFPROP ?ADJ
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CADR (ASSOC (QUOTE SYNT) (GET (CAR STRING) (QUOTE GRAM))))
        (QUOTE ADJECTIVE))
    (COND ((EQ (LENGTH (GET (QUOTE NPF-INSTANCE) (QUOTE VALUE))) 1)
           (ADD-TO-OBJ (CAR STRING)))
          (T (FILL NPF-INSTANCE ADJ (CAR STRING) CONS)))
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

NIL
*
```

X - State number

Checks for the leftmost word as an adjective.

Checks if the adjective modifies the previous noun phrase.

```
(DEFPROP ?NOUN
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CADR (ASSOC (QUOTE SYNT) (GET (CAR STRING) (QUOTE GRAM))))
        (QUOTE NOUN))
    (FILL NPF-INSTANCE NOUN (CAR STRING) RPLACD)
    (SETQ STRING (CDR STRING))
    (SLOT-EVAL NPF-INSTANCE)
    T)))
EXPR)

NIL
*
```

X - State number

Checks for the leftmost word as a noun.

Calls SLOT-EVAL to evaluate the procedural slots in the NPF frame.

```
(DEFPROP ?P
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CADR (ASSOC (QUOTE SYNT) (GET (CAR STRING) (QUOTE GRAM))))
        (QUOTE PREPOSITION))
    (FILL F-INSTANCE PREDICATE RELATION (CAR STRING) RPLACD)
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

NIL
*
```

X - State number

Checks for the leftmost word as a preposition.

```
(DEFPROP ?VERB
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CADR (ASSOC (QUOTE SYNT) (GET (CAR STRING) (QUOTE GRAM))))
        (QUOTE VERB))
    (EVAL (LIST (READLIST (CONS (QUOTE ?) (EXPLODE (CAR STRING))))))
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

NIL
*
```

X - State number

Checks for the leftmost word as a verb.

```
(DEFPROP IS
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CAR STRING) (QUOTE IS))
    (FILL F-INSTANCE PREDICATE VERB (QUOTE BE) RPLACD)
    (SETQ STRING (CDR STRING))
    T)))
EXPR)

NIL
*
```

X - State number

Checks if the leftmost word is IS.

```
(DEFPROP THERE
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CAR STRING) (QUOTE THERE)) (SETQ STRING (CDR STRING)) T)))
EXPR)

(DEFPROP WHAT
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CAR STRING) (QUOTE WHAT)) (SETQ STRING (CDR STRING) T))))
EXPR)

(DEFPROP WHERE
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CAR STRING) (QUOTE WHERE)) (SETQ STRING (CDR STRING)) T)))
EXPR)

(DEFPROP WHICH
 (LAMBDA(X)
  (SETQ NEXT X)
  (COND
   ((EQ (CAR STRING) (QUOTE WHICH)) (SETQ STRING (CDR STRING)) T)))
EXPR)

NIL
*
```

X - State number

Each FUNCTION checks for the leftmost word as the name of the FUNCTION.

```
(DEFPROP WORLD-FUNCTIONS
 (WORLD-FUNCTIONS DET
                  ADJ
                  NOUN
                  SLOT-EVAL
                  INSERT-IN-POF
                  BOOKKEEPING
                  PREDICATE
                  FILL
                  FILLEF
                  FIND-SLOT
                  FILL-SENTENCE
                  FILL-HISTORY
                  INFER-BY-PROP
                  SEARCH-WORLD
                  SEARCH-HISTORY
                  PREVIOUS-OBJECT
                  MATCH
                  MAKE
                  PRETTYLEV
                  ZOOM
                  ZOOM1
                  BE
                  SUBJ-NOTFOUND
                  SUBJ-OK
                  BOTH
                  CHECKSUBJ-OBJOK
                  CHECK-CORRECT
                  CHECK-EXISTENCE
                  ACTUAL-LOC
                  FORM-NAME
                  ISIN
                  ISON
                  CHECK-MEMBER-OF
                  NEW-LOC
                  ACTION-PPEP
                  BOTH-SIMILAR
                  SIMILAF-OBJS
                  ?PUT
                  PUT
                  PUTIN
                  PUTON
                  DELETE-POINTER
                  ADD-TO-OBJ
                  ADD-PROP
                  ADD-COLOF
                  MODIFY
                  IS-MODIFY
                  MESSAGE
                  ASSERTION
                  TRANSLATE)
VALUE)

NIL
*
```

```
(DEFPROP DET
 (LAMBDA(ARTICLE)
  (COND ((EQ (CAR ARTICLE) (QUOTE THE)) (SEARCH-HISTORY POF))
        ((MEMQ (CAR ARTICLE (QUOTE (A AN)))
               (SEARCH-WORLD POF (FILLER WORLD CONTAINS))))))
FEXPR)

NIL
*
```

FEXPR

ARTICLE - List of a definite determiner

Initializes a WORLD or a HISTORY search.

```
(DEFPROP ADJ
 (LAMBDA(LIST-OF-ADJS)
  (COND ((NULL (CAR LIST-OF-ADJS)))
        (T (INSERT-IN-POF (REVERSE (CAR LIST-OF-ADJS))))))
FEXPR)

NIL
*
```

FEXPR

LIST-OF-ADJS - A list containing a sub-list of adjectives

Calls INSERT-IN-POF to insert structural properties in a physical object frame.

```
(DEFPROP NOUN
 (LAMBDA (OBJECT) (SETQ POF (MAKE (CAR OBJECT) (QUOTE MEAN))))
FEXPR)

NIL
*
```

FEXPR

OBJECT - List of a noun

Makes a physical object frame for the noun.

```
(DEFPROP SLOT-EVAL
 (LAMBDA(INSTANCE)
  (MAPC (FUNCTION (LAMBDA (ORDER) (EVAL (ASSOC ORDER INSTANCE))))
        (CDR (ASSOC (QUOTE ORDER) INSTANCE))))
EXPR)

NIL
*
```

INSTANCE - Frame name

Evaluates the procedural slots in INSTANCE

```
(DEFPROP INSERT-IN-POF
 (LAMBDA(ADJS)
  (SETQ POF
        (APPEND
         (LIST (CAR POF))
         (MAPCAN
          (FUNCTION
           (LAMBDA(ADJ)
            ((LAMBDA(A)
              (AND (MEMQ (CAR POF) (FILLER A APPLY-TO))
                   (LIST (CADR A))))
             (MAKE ADJ (QUOTE MEAN)))))
          ADJS)
         (CDR POF))))
EXPR)

NIL
*
```

ADJS - List of adjectives

Checks if each of the adjectives applies to the noun.

If it does, inserts properties in the physical object frame.

```
(DEFPROP BOOKKEEPING
 (LAMBDA (L) (SETQ HISTORY (CONS (EVAL (CAR L)) HISTORY)))
FEXPR)

NIL
*
```

FEXPR

L - List of the name of a sentence frame

Keeps the sentence frame in HISTORY.

```
(DEFPROP PREDICATE
 (LAMBDA(L)
  (AND (CADAAR L)
       (EVAL
        (MAPCAN (FUNCTION
                 (LAMBDA (ACTION-TYPE) (COPY (CDR ACTION-TYPE))))
                (CAR L)))))
FEXPR)

NIL
*
```

FEXPR

L - A list containing a sub-list of two slots--VERB and RELATION

Forms a list of two atoms--the verb being its CAR and the preposition being its CADR, and evaluates the expression.

```
(DEFPROP FILL
 (LAMBDA(L)
  (PROG (REMAIN)
        ((LAMBDA(SLOT FILLER ADD-TO)
          (COND (ADD-TO
                 (RPLACD SLOT
                         (LIST (NCONC (LIST FILLER) (CADR SLOT)))))
                (T (RPLACD SLOT (LIST FILLER)))))
         (FIND-SLOT (CDR L) (EVAL (CAR L)))
         (EVAL (CAR REMAIN))
         (EQ (CADR REMAIN) (QUOTE CONS)))))
FEXPR)

NIL
*
```

FEXPR

L - List of a frame name, slot names, the name of a structure to be filled, and the way to be filled.

Calls FIND-SLOT to find the slot and fills it with the new structure.

```
(DEFPROP FILLER
 (LAMBDA(L)
  (CADR
   (FIND-SLOT (APPEND (CDR L) (QUOTE (FOR FILLER))) (EVAL (CAR L)))))
FEXPR)

NIL
*
```

FEXPR

L - List of a frame name and slot names

Calls FIND-SLOT to find the right slot and returns the filler of the slot.

```
(DEFPROP FIND-SLOT
 (LAMBDA(INFO FRAME)
  ((LAMBDA(?EVAL END)
    (COND (END (AND (MEMQ (CADDR INFO) (QUOTE (CONS RPLACD)))
                    (SETQ REMAIN (CDR INFO)))
               ?EVAL)
          (T (FIND-SLOT (CDR INFO) (CADR ?EVAL)))))
   (ASSOC (CR (AND (ATOM (CAR INFO)) (CAR INFO)) (EVAL (CAAR INFO)))
          FRAME)
   (MEMQ (CADDR INFO) (QUOTE (CONS RPLACD FILLER)))))
EXPR)

NIL
*
```

INFO - List of slot names and other information specified in FILL or FILLER

FRAME - A structure--a frame or a list of sub-slots

Calls recursively to find the right slot for FILL or FILLER.

```
(DEFPROP FILL-SENTENCE
 (LAMBDA NIL
  (SETQ NP-INSTANCE (COPY (GET (QUOTE NP) (QUOTE FRAME))))
  (FILL NP-INSTANCE SYNTAX NPF-INSTANCE RPLACD)
  (FILL NP-INSTANCE SEMANTICS POF RPLACD))
EXPR)

NIL
*
```

Creates an instance of the prototype NP frame.

Fills the slots in the frame.

```
(DEFPROP FILL-HISTORY
 (LAMBDA NIL
  (FILL F-INSTANCE SUBJECT SEMANTICS SUBJECT RPLACD)
  (FILL F-INSTANCE OBJECT SEMANTICS OBJECT RPLACD))
EXPR)

NIL
*
```

Makes the objects mentioned in a sentence share structures with the WORLD objects.

```
(DEFPROP INFER-BY-PROP
 (LAMBDA (FUNCTIONAL)
  (AND (NULL (ASSOC FUNCTIONAL OBJECT))
       (EVAL
        (APPEND (QUOTE (ERRMSG IT IS WRONG THAT))
                (LIST (QUOTE A)
                      (CAR OBJECT)
                      FUNCTIONAL
                      (QUOTE A)
                      (CAR SUBJECT))))))
EXPR)

NIL
*
```

FUNCTIONAL - CONTAINS or SUPPORTS

Finds the functional slot in a physical object frame.

Calls ERRMSG if the slot is not found.

```
(DEFPROP SEARCH-WORLD
 (LAMBDA(POF WORLD-OBJECTS)
  (COND ((NOT
          (ERRSET
           (MAPC (FUNCTION
                  (LAMBDA(WORLD-OBJ)
                   (AND (EQ (CAR POF) (CAR WORLD-OBJ))
                        (ERRSET (MATCH POF WORLD-OBJ))
                        (SETQ PHYSOB WORLD-OBJ)
                        (SETQ S-RESULT (QUOTE SIMILAR))
                        (ERR NIL))))
                 WORLD-OBJECTS))))
        (T (SETQ PHYSOB POF) (SETQ S-RESULT (QUOTE NONE)))))
EXPR)

NIL
*
```

POF - Physical object frame

WORLD-OBJECTS - List of physical object frames in the WORLD's CONTAINS slot.

Searches WORLD for an object similar to POF.

```
(DEFPROP SEARCH-HISTORY
 (LAMBDA(POF)
  (COND ((EQ (CAR POF) (QUOTE ROOM))
         (SETQ PHYSOB WORLD)
         (SETQ S-RESULT (QUOTE FOUND)))
        ((NOT
          (ERRSET
           (MAPC (FUNCTION
                  (LAMBDA(SENTENCE)
                   (OR (NEQ (CAR SENTENCE) (QUOTE DECL-SENT))
                       (PREVIOUS-OBJECT (QUOTE SUBJECT))
                       (PREVIOUS-OBJECT (QUOTE OBJECT)))))
                 HISTORY)))
         (SETQ S-RESULT (QUOTE FOUND)))
        (T (SETQ PHYSOB POF) (SETQ S-RESULT (QUOTE NOTFOUND)))))
EXPR)

NIL
*
```

POF - Physical object frame

Searches HISTORY for a mentioned object with its structural properties similar to those of POF.

```
(DEFPROP PREVIOUS-OBJECT
 (LAMBDA(OBJ)
  (AND (EQ (CAR POF)
           (CAR (SETQ PHYSOB (FILLER SENTENCE (OBJ) SEMANTICS))))
       (ERRSET (MATCH POF PHYSOB))
       (ERR NIL)))
EXPR)

NIL
*
```

OBJ - SUBJECT or OBJECT

Retrieves OBJ from HISTORY.

Calls MATCH to match the structural properties between OBJ and POF.

```
(DEFPROP MATCH
 (LAMBDA(POF WORLD-OBJ)
  (AND (GREATERP (LENGTH (CDR POF)) (LENGTH (CDR WORLD-OBJ)))
       (ERR NIL))
  (MAPC (FUNCTION
         (LAMBDA(NEW-OBJ-PROP)
          (OR (EQ (CADR NEW-OBJ-PROP)
                  (CADR (ASSOC (CAR NEW-OBJ-PROP) (CDR WORLD-OBJ))))
              (MEMQ (CAR NEW-OBJ-PROP)
                    (QUOTE (CONTAINS SUPPORTS LOCATION)))
              (ERR NIL))))
        (CDR POF)))
EXPR)

NIL
*
```

POF - Physical object frame

WORLD-OBJ - WORLD object frame

Finds a match between POF and WORLD-OBJ.

```
(DEFPROP MAKE
 (LAMBDA (W I) (CONS W (COPY (GET W I))))
EXPR)

NIL
*
```

W - Concept's name

I - Property indicator

Makes a frame corresponding to the concept.

```
(DEFPROP PRETTYLEV
 (LAMBDA(S N)
  (DSKOUT JUNK
          (PROG1 (PRINC (QUOTE "(PROG NIL (SETQ JUNK @"))
                 (PRINTLEV S N)
                 (PRINC (QUOTE /])))))
  (DSKIN JUNK)
  (TERPRI)
  (SPRINT JUNK 1))
EXPR)

NIL
*
```

S - Structure

N - Number

Prints the structure (infinite list) to certain levels.

```
(DEFPROP ZOOM
 (LAMBDA(OBJECT)
  (COND ((ATOM (CAR OBJECT)) (ZOOM1 OBJECT))
        (T (MAPCAR (FUNCTION (LAMBDA (OBJS) (ZOOM1 OBJS))) OBJECT))))
EXPR)

NIL
*
```

OBJECT - List of physical object frames

Translates OBJECT into noun phrases without definite determiners.

```
(DEFPROP ZOOM1
 (LAMBDA(OBJECT)
  (APPEND
   (MAPCAN
    (FUNCTION
     (LAMBDA(SLOT)
      (AND (MEMQ (CAR SLOT) (QUOTE (SIZE COLOR)))
           (COPY (CDR SLOT)))))
    (CDR OBJECT))
   (LIST (CAR OBJECT))))
EXPR)

NIL
*
```

OBJECT - Physical object frame

Translates OBJECT into a noun phrase without a definite determiner.

```
(DEFPROP BE
 (LAMBDA(RELATION)
  (COND ((NULL (CAR RELATION)) (MODIFY))
        (T
         ((LAMBDA(DECL-SENT SUBJ-STATUS OBJ-STATUS)
           (COND ((EQ SUBJ-STATUS (QUOTE NOTFOUND)) (SUBJ-NOTFOUND))
                 ((NOT (SUBJ-OK)) (BOTH)))
           (COND ((NULL DECL-SENT) (TRANSLATE))
                 ((EQ (FILLER F-INSTANCE REPLY) 3) (FILL-HISTORY))))
          (EQ (CAR F-INSTANCE) (QUOTE DECL-SENT))
          (FILLER F-INSTANCE SUBJECT-SEARCH RESULT)
          (FILLER F-INSTANCE OBJECT-SEARCH RESULT)))))
FEXPR)

NIL
*
```

RELATION - Preposition

Monitors the process when the verb of a sentence is IS.

```
(DEFPROP SUBJ-NOTFOUND
 (LAMBDA NIL
  (COND ((EQ OBJ-STATUS (QUOTE NOTFOUND))
         (SETQ PASS-TO-REPLY
               (APPEND (ZOOM SUBJECT) (QUOTE (AND)) (ZOOM OBJECT))))
        (T (SETQ PASS-TO-REPLY (ZOOM SUBJECT))))
  (FILL F-INSTANCE REPLY 21 RPLACD))
EXPR)

NIL
*
```

The subject is already not found in HISTORY.

Checks if the object is found in HISTORY.

Sets up a reply and reply number.

```
(DEFPROP SUBJ-OK
 (LAMBDA NIL
  (COND
   ((EQ OBJ-STATUS (QUOTE NOTFOUND))
    (SETQ PASS-TO-REPLY (ZOOM OBJECT))
    (FILL F-INSTANCE REPLY 21 RPLACD)
    T)))
EXPR)

NIL
*
```

The search result for SUBJECT is not NOTFOUND.

Checks if the object is found in HISTORY.

Sets up a reply and a reply number.

```
(DEFPROP POTH
 (LAMBDA NIL
  (COND ((AND (EQ SUBJ-STATUS (QUOTE FOUND))
              (EQ OBJ-STATUS (QUOTE FOUND)))
         (CHECK-EXISTENCE))
        (T (CHECKSUBJ-OBJOK))))
EXPR)

NIL
*
```

Directs further processes when the search results for both the subject and object are not NOTFOUND.

```
(DEFPROP CHECKSUBJ-OBJOK
 (LAMBDA NIL
  (COND ((EQ SUBJ-STATUS (QUOTE FOUND)) (CHECK-CORRECT))
        (T (FORM-NAME (QUOTE IS) (CAR RELATION)))))
EXPR)

NIL
*
```

When the search results for both the subject and object are not FOUND, checks if the subject is found in HISTORY.

```
(DEFPROP CHECK-CORRECT
 (LAMBDA NIL
  ((LAMBDA(SUBJ-LOC)
    (COND ((AND (EQ (CAR OBJECT) (CAR SUBJ-LOC))
                (ERRSET (MATCH OBJECT SUBJ-LOC)))
           (FILL F-INSTANCE REPLY 22 REPLACD))
          (T (ACTUAL-LOC (FILLER SUBJECT LOCATION RELATION)))))
   (FILLER SUBJECT LOCATION POINTER)))
EXPR)

NIL
*
```

Looks for the correctness of a sentence when the subject is found in HISTORY.

```
(DEFPROP CHECK-EXISTENCE
 (LAMBDA NIL
  ((LAMBDA(SUBJ-LOC)
    (COND ((EQ OBJECT SUBJ-LOC) (FILL F-INSTANCE REPLY 22 RPLACD))
          (T (ACTUAL-LOC (FILLER SUBJECT LOCATION RELATION)))))
   (FILLER SUBJECT LOCATION POINTER)))
EXPR)

NIL
*
```

When both the subject and object are found in HISTORY, checks if the event has been described before.

Sets up a reply and a reply number.

```
(DEFPROP ACTUAL-LOC
 (LAMBDA(RELATION)
  (SETQ PASS-TO-REPLY
        (APPEND (QUOTE (THE))
                (ZOOM SUBJECT)
                (LIST (QUOTE IS) RELATION (QUOTE THE))
                (ZOOM SUBJ-LOC)))
  (FILL F-INSTANCE REPLY 23 RPLACD))
EXPR)

NIL
*
```

RELATION - Preposition

Sets up a reply telling the user the actual location of the subject.

```
(DEFPROP FORM-NAME
 (LAMBDA(VERB RELATION)
  (EVAL (LIST (READLIST (NCONC (EXPLODE VERB) (EXPLODE RELATION))))))
EXPR)

NIL
*
```

VERB - Verb

RELATION - Preposition

Forms an expression combining the verb and the preposition in one word.

Evaluates the expression.

```
(DEFPROP ISIN
 (LAMBDA NIL
  (INFER-BY-PROP (QUOTE CONTAINS))
  (COND ((EQ OBJ-STATUS (QUOTE FOUND)) (CHECK-MEMBER-OF (QUOTE IN)))
        ((EQ OBJ-STATUS (QUOTE NONE))
         (OR (NULL DECL-SENT) (NEW-LOC OBJECT WORLD (QUOTE IN)))
         (ACTION-PREP (QUOTE IN) (QUOTE CONTAINS)))
        ((EQ SUBJ-STATUS (QUOTE NONE))
         (ACTION-PREP (QUOTE IN) (QUOTE CONTAINS)))
        ((ERRSET (BOTH-SIMILAR))
         (ACTION-PREP (QUOTE IN) (QUOTE CONTAINS)))
        (T (FILL F-INSTANCE REPLY 22 RPLACD))))
EXPR)

(DEFPROP ISON
 (LAMBDA NIL
  (INFER-BY-PROP (QUOTE SUPPORTS))
  (COND ((EQ OBJ-STATUS (QUOTE FOUND)) (CHECK-MEMBER-OF (QUOTE ON)))
        ((EQ OBJ-STATUS (QUOTE NONE))
         (OR (NULL DECL-SENT) (NEW-LOC OBJECT WORLD (QUOTE IN)))
         (ACTION-PREP (QUOTE ON) (QUOTE SUPPORTS)))
        ((EQ SUBJ-STATUS (QUOTE NONE))
         (ACTION-PREP (QUOTE ON) (QUOTE SUPPORTS)))
        ((ERRSET (BOTH-SIMILAR))
         (ACTION-PREP (QUOTE ON) (QUOTE SUPPORTS)))
        (T (FILL F-INSTANCE REPLY 22 RPLACD))))
EXPR)

NIL
*
```

Carries out appropriate actions when the verb is IS and the preposition is IN or ON.

```
(DEFPROP CHECK-MEMBER-OF
 (LAMBDA(RELATION)
  ((LAMBDA(FUNCTIONAL)
    (COND
     ((ERRSET
       (MAPC
        (FUNCTION
         (LAMBDA(OBJ)
          (AND (EQ (CAR SUBJECT) (CAR OBJ))
               (ERRSET (MATCH SUBJECT OBJ))
               (ERR NIL))))
        (CADR FUNCTIONAL)))
      (ACTION-PREP RELATION (CAR FUNCTIONAL)))
     (T (FILL F-INSTANCE REPLY 22 REPLACD))))
   (ASSOC (CONVERT RELATION) OBJECT)))
EXPR)

NIL
*
```

RELATION - Preposition

Checks if the subject is on or in the object, when the subject is not found in HISTORY but the object is found.

```
(DEFPROP NEW-LOC
 (LAMBDA(FRAME-NAME NEW-LOCATION PREPOSITION)
  (FILL WORLD CONTAINS FRAME-NAME CONS)
  (FILL FRAME-NAME LOCATION POINTER NEW-LOCATION RPLACD)
  (FILL FRAME-NAME LOCATION RELATION PREPOSITION RPLACD))
EXPR)

NIL
*
```

FRAME-NAME - Physical object frame

NEW-LOCATION - New location of FRAME-NAME

PREPOSITION - Preposition

Adds FRAME-NAME to WORLD. Fills FRAME-NAME's LOCATION slot.

```
(DEFPROP ACTION-PREP
 (LAMBDA(PREP RELATION)
  (COND
   (DECL-SENT (SETQ SUBJECT (FILLER F-INSTANCE SUBJECT SEMANTICS))
              (NEW-LOC SUBJECT OBJECT PREP)
              (OR (EQ OBJECT WORLD)
                  (FILL OBJECT (RELATION) SUBJECT CONS))))
  (FILL F-INSTANCE REPLY 3 RPLACD))
EXPR)

NIL
*
```

PREP - Preposition

RELATION - Functional slot name

Establishes the relationship between the subject and object.

```
(DEFPROP BOTH-SIMILAR
 (LAMBDA NIL
  ((LAMBDA(OBJECT--S)
    (MAPC (FUNCTION
           (LAMBDA(OBJ-TARGET)
            (OR (NULL OBJ-TARGET)
                (AND (MEMQ (FILLER OBJ-TARGET LOCATION POINTER)
                           OBJECT--S)
                     (EPP NIL)))))
          (SIMILAR-OBJS (FILLER F-INSTANCE SUBJECT SEMANTICS))))
   (SIMILAR-OBJS (FILLER F-INSTANCE OBJECT SEMANTICS))))
EXPP)

NIL
*
```

Checks if the currently described event has been previously introduced.

```
(DEFPROP SIMILAR-OBJS
 (LAMBDA(OBJ)
  (MAPCAR (FUNCTION
           (LAMBDA(X)
            (AND (EQ (CAR OBJ) (CAR X)) (EPPSET (MATCH OBJ X)) X)))
          (FILLER WORLD CONTAINS)))
EXPR)

NIL
*
```

OBJ - Physical object frame

Returns a list of physical object frames similar to OBJ.

```
(DEFPROP ?PUT
 (LAMBDA NIL
  (OR (EQ (FILLER F-INSTANCE SUBJECT-SEARCH RESULT) (QUOTE FOUND))
      (FILL F-INSTANCE REPLY 26 RPLACD))
  (FILL F-INSTANCE PREDICATE VERB (CAR STRING) RPLACD))
EXPR)

NIL
*
```

Initial process for the verb PUT.

```
(DEFPROP PUT
 (LAMBDA(RELATION)
  (INFER-BY-PROP (CONVERT (CAR RELATION)))
  (OR (FILLER F-INSTANCE REPLY)
      ((LAMBDA(OBJ-STATUS)
        (COND ((AND (MEMQ OBJ-STATUS (QUOTE (SIMILAR FOUND)))
                    (ERRSET
                     (MATCH OBJECT
                            (FILLER SUBJECT LOCATION POINTER))))
               (FILL F-INSTANCE REPLY 30 RPLACD))
              ((OR (EQ OBJ-STATUS (QUOTE FOUND))
                   (NEW-LOC (FILLER F-INSTANCE OBJECT SEMANTICS)
                            WORLD
                            (QUOTE IN))
                   T)
               (FORM-NAME (QUOTE PUT) (CAR RELATION))
               (FILL-HISTORY))))
       (FILLER F-INSTANCE OBJECT-SEARCH RESULT))))
FEXPR)

NIL
*
```

FEXPR

RELATION - Preposition

Monitors appropriate operations for the verb PUT.

```
(DEFPROP PUTIN
 (LAMBDA NIL
  ((LAMBDA(OLD-LOC OLD-RELATION)
    (FILL SUBJECT LOCATION POINTER OBJECT RPLACD)
    (FILL SUBJECT LOCATION RELATION (QUOTE IN) RPLACD)
    (FILL OBJECT CONTAINS SUBJECT CONS)
    (OR (EQ OLD-LOC WORLD)
        (DELETE-POINTER SUBJECT OLD-LOC OLD-RELATION))
    (FILL F-INSTANCE REPLY 14 RPLACD))
   (FILLER SUBJECT LOCATION POINTER)
   (CONVERT (FILLER SUBJECT LOCATION RELATION))))
EXPR)

(DEFPROP PUTON
 (LAMBDA NIL
  ((LAMBDA(OLD-LOC OLD-RELATION)
    (FILL SUBJECT LOCATION POINTER OBJECT RPLACD)
    (FILL SUBJECT LOCATION RELATION (QUOTE ON) RPLACD)
    (FILL OBJECT SUPPORTS SUBJECT CONS)
    (OR (EQ OLD-LOC WORLD)
        (DELETE-POINTER SUBJECT OLD-LOC OLD-RELATION))
    (FILL F-INSTANCE REPLY 4 RPLACD))
   (FILLER SUBJECT LOCATION POINTER)
   (CONVERT (FILLER SUBJECT LOCATION RELATION))))
EXPR)

NIL
*
```

Puts the subject in or on the object.

Resets pointers, etc.

```
(DEFPROP DELETE-POINTER
 (LAMBDA(OBJ OLD-LOC OLD-RELATION)
  ((LAMBDA(F-SLOT)
    (COND ((NULL (CDR F-SLOT))
           (FILL OLD-LOC (OLD-RELATION) NIL RPLACD))
          (T (DREMOVE OBJ F-SLOT))))
   (FILLER OLD-LOC (OLD-RELATION))))
EXPR)

NIL
*
```

OBJ - Physical object frame

OLD-LOC - POINTER of OBJ's LOCATION slot

OLD-RELATION - RELATION of OBJ's LOCATION slot.

Gets rid of OBJ from the functional slot of its location.

```
(DEFPROP ADD-TO-OBJ
 (LAMBDA(ADD-ADJ)
  ((LAMBDA(DECL-SENT SIZE COLOR)
    (OR (AND SIZE COLOR (ASSERTION))
        ((LAMBDA(ADJ)
          (COND
           (SIZE
            (OR (AND (ASSOC (QUOTE SIZE) ADJ) (MESSAGE (CDR SIZE)))
                (ADD-COLOR)))
           (COLOR
            (OR (AND (ASSOC (QUOTE COLOR) ADJ) (MESSAGE (CDR COLOR)))
                (ADD-PROP)))
           (T (ADD-PROP))))
         (MAKE ADD-ADJ (QUOTE MEAN)))))
   (EQ (CAR F-INSTANCE) (QUOTE DECL-SENT))
   (ASSOC (QUOTE SIZE) PHYSOB)
   (ASSOC (QUOTE COLOR) PHYSOB)))
EXPR)

NIL
*
```

ADD-ADJ - Adjective

Performs appropriate processes--modifying a physical object's properties.

```
(DEFPROP ADD-PROP
 (LAMBDA NIL
  (COND (DECL-SENT
         (RPLACD SUBJECT (NCONC (LIST (CADR ADJ)) (CDR SUBJECT))))
        (T
         (SETQ SUBJECT
               (NCONC (LIST (CAR SUBJECT))
                      (LIST (CADR ADJ))
                      (CDR SUBJECT))))))
EXPR)

NIL
*
```

Adds size or color to a physical object frame.

```
(DEFPROP ADD-COLOR
 (LAMBDA NIL
  (COND (DECL-SENT
         (RPLACD (CDR SUBJECT)
                 (NCONC (LIST (CADR ADJ)) (CDDR SUBJECT))))
        (T
         (SETQ SUBJECT
               (NCONC (LIST (CAR SUBJECT) (CADR SUBJECT))
                      (LIST (CADR ADJ))
                      (CDDR SUBJECT))))))
EXPR)

NIL
*
```

Adds color to a physical object frame, i.e. size already exists.

```
(DEFPROP MODIFY
 (LAMBDA NIL
  (OR (FILLER F-INSTANCE REPLY)
      (COND ((EQ (CAR F-INSTANCE) (QUOTE DECL-SENT))
             (FILL F-INSTANCE REPLY 5 RPLACD)
             (FILL F-INSTANCE SUBJECT SEMANTICS SUBJECT RPLACD))
            (T (IS-MODIFY)))))
EXPR)

NIL
*
```

Directs processes for the following sentence patterns:

1. ?NPS IS ?ADJ

2. IS ?NPS ?ADJ

```
(DEFPROP IS-MODIFY
 (LAMBDA NIL
  (COND ((ERRSET (MATCH SUBJECT PHYSOB))
         (FILL F-INSTANCE REPLY 11 RPLACD)
         (FILL F-INSTANCE SUBJECT SEMANTICS PHYSOB RPLACD))
        (T (FILL F-INSTANCE REPLY 25 RPLACD))))
EXPR)

NIL
*
```

Answers questions of the pattern IS ?NPS ?ADJ

```
(DEFPROP ASSERTION
 (LAMBDA NIL
  (COND ((ERRSET
           (MAPC (FUNCTION
                   (LAMBDA(PROP)
                    (AND (EQ (CADR PROP) ADD-ADJ) (ERR NIL))))
                 (CDR SUBJECT)))
         (OR (AND DECL-SENT
                  (MESSAGE
                   (LIST (CADR SIZE) (QUOTE AND) (CADR COLOR))))
             (FILL F-INSTANCE REPLY 25 RPLACD)))
        (DECL-SENT (FILL F-INSTANCE REPLY 31 RPLACD))
        (T (FILL F-INSTANCE REPLY 11 RPLACD)
           (FILL F-INSTANCE SUBJECT SEMANTICS SUBJECT RPLACD)))
  T)
EXPR)

NIL
*
```

When size and color already exist, checks for any contradiction.

```
(DEFPROP MESSAGE
 (LAMBDA(ADJECTIVE)
  (AND DECL-SENT
       (EVAL
        (APPEND (QUOTE (ERRMSG THE))
                (LIST (CAR SUBJECT) (QUOTE IS))
                ADJECTIVE))))
EXPR)

NIL
*
```

ADJECTIVE - Adjective

Gives an error message for a wrong assertion.

```
(DEFPROP TRANSLATE
 (LAMBDA NIL
  ((LAMBDA(REPLY-NUMBER)
    (COND ((EQ REPLY-NUMBER 22)
           (FILL F-INSTANCE REPLY 11 RPLACD)
           (FILL-HISTORY))
          ((EQ REPLY-NUMBER 3) (FILL F-INSTANCE REPLY 25 RPLACD))))
   (FILLER F-INSTANCE REPLY)))
EXPR)

NIL
*
```

Converts a reply number for a declarative sentence into a reply number for an IS question.

```
(DEFPROP QUESTIONS
 (QUESTIONS SUBJECT-SEARCH
            OBJECT-SEARCH
            QUESTION-WHERE
            QUESTION-WHAT
            QUESTION-WHICH
            WHICH-ADJ
            WHICH-NPO
            FORM-QUESTION-NAME
            CONVERT)
VALUE)

NIL
*
```

```
(DEFPROP SUBJECT-SEARCH
 (LAMBDA(L)
  (COND ((EQ (CADAAR L) (QUOTE FOUND))
         (EVAL (FORM-QUESTION-NAME (QUOTE SUBJECT))))
        (T (FILL F-INSTANCE REPLY 21 RPLACD)
           (SETQ PASS-TO-REPLY (ZOOM SUBJECT)))))
FEXPR)

NIL
*
```

FEXPR

L - A list containing a sub-list of two slots--RESULT and POINTER.

If RESULT of the SUBJECT-SEARCH slot is FOUND, then SUBJECT-SEARCH calls FORM-QUESTION-NAME.

```
(DEFPROP OBJECT-SEARCH
 (LAMBDA(L)
  (COND ((OR (EQ (CADAAR L) (QUOTE FOUND)) (NULL OBJECT))
         (EVAL (FORM-QUESTION-NAME (QUOTE OBJECT))))
        (T (FILL F-INSTANCE REPLY 21 RPLACD)
           (SETQ PASS-TO-REPLY (ZOOM OBJECT)))))
FEXPR)

NIL
*
```

FEXPR

L - A list containing a sub-list of two slots--RESULT and POINTER.

If RESULT of the OBJECT-SEARCH slot is FOUND or there is no object in a question, then OBJECT-SEARCH calls FORM-QUESTION-NAME.

```
(DEFPROP QUESTION-WHERE
 (LAMBDA(OBJ)
  ((LAMBDA(LOC-PT LOC-REL)
    (SETQ PASS-TO-REPLY
          (APPEND (LIST LOC-REL (QUOTE THE)) (ZOOM LOC-PT)))
    (FILL F-INSTANCE REPLY 6 RPLACD)
    (FILL F-INSTANCE SUBJECT SEMANTICS OBJ RPLACD))
   (FILLER OBJ LOCATION POINTER)
   (FILLER OBJ LOCATION RELATION)))
EXPR)

NIL
*
```

OBJ - Physical object frame

Answers a where type question.

```
(DEFPROP QUESTION-WHAT
 (LAMBDA(OBJ)
  ((LAMBDA(FUNCTIONAL)
    (COND ((NULL FUNCTIONAL) (FILL F-INSTANCE REPLY 7 RPLACD))
          (T (SETQ PASS-TO-REPLY (ZOOM FUNCTIONAL))
             (FILL F-INSTANCE REPLY 2 RPLACD)))
    (FILL F-INSTANCE OBJECT SEMANTICS OBJ RPLACD))
   (FILLER OBJ ((CONVERT (FILLER F-INSTANCE PREDICATE RELATION))))))
EXPR)

NIL
*
```

OBJ - Physical object frame

Answers a what type question.

```
(DEFPROP QUESTION-WHICH
 (LAMBDA(OBJ)
  (COND ((NULL OBJ) (WHICH-ADJ))
        (T
         (WHICH-NPO
          (CONVERT (FILLER F-INSTANCE PREDICATE RELATION)))))
  (FILL F-INSTANCE SUBJECT SEMANTICS PHYSOB RPLACD))
EXPR)

NIL
*
```

OBJ - Physical object frame

Initializes processes for a which type question.

```
(DEFPROP FORM-QUESTION-NAME
 (LAMBDA(OBJ)
  (LIST (READLIST
         (NCONC (EXPLODE (QUOTE QUESTION-))
                (EXPLODE (CAR F-INSTANCE))))
        OBJ))
EXPR)

NIL
*
```

OBJ - SUBJECT or OBJECT

Forms a question type procedure name.

```
(DEFPROP CONVERT
 (LAMBDA(REL-PREP)
  (COND ((EQ REL-PREP (QUOTE IN)) (QUOTE CONTAINS))
        ((EQ REL-PREP (QUOTE ON)) (QUOTE SUPPORTS))))
EXPR)

NIL
*
```

REL-PREP - Preposition

Converts IN into CONTAINS and ON into SUPPORTS.

```
(DEFPROP WHICH-ADJ
 (LAMBDA NIL
  (SEARCH-WORLD SUBJECT (FILLER WORLD CONTAINS))
  (COND ((EQ S-RESULT (QUOTE SIMILAR))
         (SETQ PASS-TO-REPLY
               (APPEND (LIST (FILLER PHYSOB LOCATION RELATION)
                             (QUOTE THE))
                       (ZOOM (FILLER PHYSOB LOCATION POINTER))))
         (FILL F-INSTANCE
               SUBJECT-SEARCH
               RESULT
               (QUOTE SIMILAR)
               RPLACD)
         (FILL F-INSTANCE SUBJECT-SEARCH POINTER SUBJECT RPLACD)
         (FILL F-INSTANCE REPLY 12 RPLACD))
        (T (FILL F-INSTANCE
                 SUBJECT-SEARCH
                 RESULT
                 (QUOTE NONE)
                 RPLACD)
           (FILL F-INSTANCE REPLY 26 RPLACD))))
EXPR)

NIL
*
```

Answers WHICH type questions of the pattern WHICH ?NP-N IS ?ADJ

```
(DEFPROP WHICH-NPO
 (LAMBDA(FUNCTIONAL)
  (SEARCH-WORLD SUBJECT (FILLEP OBJ (FUNCTIONAL)))
  (COND ((EQ S-RESULT (QUOTE SIMILAR))
         (FILL F-INSTANCE
               SUBJECT-SEARCH
               RESULT
               (QUOTE SIMILAR)
               RPLACD)
         (FILL F-INSTANCE SUBJECT-SEARCH POINTER SUBJECT RPLACD)
         (FILL F-INSTANCE REPLY 13 RPLACD))
        (T (FILL F-INSTANCE
                 SUBJECT-SEARCH
                 RESULT
                 (QUOTE NONE)
                 RPLACD)
           (FILL F-INSTANCE REPLY 27 RPLACD)
           (SETQ PASS-TO-REPLY (LIST FUNCTIONAL (QUOTE A))))))
EXPR)

NIL
*
```

FUNCTIONAL - CONTAINS or SUPPORTS

Answers WHICH type questions of the pattern

WHICH ?NP-N IS ?P ?NPO

## References

[1] Bobrow, D.G. "Natural Language Input for a Computer Problem Solving System." In Minsky, M. (Ed.), Semantic Information Processing. Cambridge, Mass.: The M.I.T. Press, 1968.

[2] Bobrow, D.G., and the PARC understanding group. "GUS-1, A Frame Driven Dialog System." Artificial Intelligence 8:2, (Spring 1977).

[3] Bobrow, D.G., and Winograd, T. "An Overview of KRL, a Knowledge Representation Language." Cognitive Science 1:1 (January 1977).

[4] Davis, R., and King, J. "An Overview of Production Systems." (Stanford AI MEMO 271). Stanford, Ca.: Stanford University, October 1975.

[5] Hewitt, C. "A Language for Proving Theorems in Robots." Proceedings of the International Joint Conference on Artificial Intelligence, p. 295-301, Bedford, Mass., Mitre Corp., (1969).

[6] Kaplan, R. "On Process Models for Sentence Analysis." In Norman, D.A., Rumelhart, D.E., and the LNR Research Group, Explorations in Cognition. San Francisco, Ca.: Freeman, 1975.

[7] Marinov, V. "Memory by Description in the ROOM Program." 1977 ACM Computer Conference in Atlanta, Jan. 31-Feb. 2, abstract, p.4.

[8] Marinov, V., Bregar, W., Poon, C., and Shih, T. "It's a Small World - An Approach To Representation." Department of Computer Science, Oregon State University, 1977, unpublished.

[9] McCarthy, J., and Hayes, P. "Some Philosophical Problems from the Standpoint of Artificial Intelligence." In Meltzer B., and Michie, D. (Eds.), Machine Intelligence 4, p. 463-502. New York: American Elsevier Publishing Company, Inc., 1969.

[10] Minsky, M. "A Framework For Representing Knowledge." In Winston, P. (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill, 1975.

[11] Nash-Webber, B. "The Role of Semantics in Automatic Speech Understanding." In Bobrow, D.G. and Collins, A. (Eds.), Representation and Understanding. New York: Academic Press, 1975.

[12] Newell, A., and Simon, H. Human Problem Solving. Prentice Hall, 1972.

[13] Nilsson, N. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, 1971.

[14] Raphael, B. "SIR: A Computer Program for Semantic Information Retrieval." In Minsky, M. (Ed.), Semantic Information Processing. Cambridge, Mass.: The M.I.T. Press, 1968.

[15] Simmons, R.F. "Semantic Networks: Their Computation and Use for Understanding English Sentences." In Schank, R., and Colby, K. (Eds.), Computer Models of Thought and Language. San Francisco, Ca.: Freeman, 1973.

[16] Winograd, T. Understanding Natural Language. New York: Academic Press, 1972.

[17] Winograd, T. "Five Lectures on Artificial Intelligence." (Stanford AI MEMO 246). Stanford, Ca.: Stanford University, September 1974.

[18] Winograd, T. "Frames and the Declarative-Procedural Controversy." In Bobrow, D.G., and Collins, A. (Eds.), Representation and Understanding. New York: Academic Press, 1975.

[19] Woods, W.A. "Transition Network Grammars for Natural Language Analysis." Communications of the Association for Computing Machinery, 1970, 13(10), p. 591-606.

[20] Woods, W.A., Kaplan, R.M., and Nash-Webber, B. "The Lunar Sciences Natural Language Information System: Final Report." (BBN Report No. 2378). Cambridge, Mass.: Bolt Beranek and Newman, June 1972.

[21] Woods, W.A. "What's In A Link: Foundations for Semantic Networks." In Bobrow, D.G., and Collins, A. (Eds.), Representation and Understanding. New York: Academic Press, 1975.