AN N-SPACE MODEL FOR VISUAL AND VERBAL CONCEPTS

by

JEFFREY FORD EASTMAN

A thesis submitted to the Graduate Faculty of
North Carolina State University at Raleigh
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

DEPARTMENT OF ELECTRICAL ENGINEERING

RALEIGH

1 9 7 5

APPROVED BY:

James W. Gault	[illegible]
[illegible]	John Staudhammer Chairman of Advisory Committee

BIOGRAPHY

The author was born in Cambridge, Massachusetts on March 25, 1945 and received his early education in Stanford, California and Marburg, Germany. He spent seven years in the United States Navy as a Polaris Fire Control Technician and NESEP Candidate. After receiving his B.S. degree in Electrical Engineering from North Carolina State University in 1970, he worked for a year as an engineer and then accepted a Research Assistantship on a National Science Foundation Grant in the area of computer graphics.

The author married Edith-Margaret Lloyd Naylor in 1971.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Dr. John Staudhammer for providing him with the opportunity and environment in which to pursue the work reported herein, and to Dr. Alan Tharp for his helpful criticisms of it. Thanks are also due to Marillyn Mulholland for her first dissertation typing and to Edith Eastman for her sense of humor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. PREVIEW

1.1 Introduction

The rapid evolution in computer technology over the last twenty-five years has fostered an equally rapid evolution in computing languages. The goal of this evolution has been to free the programmer from the hardware-dependent details of program execution and allow him to address the more important and interesting problem-oriented features of his algorithm. Higher level languages such as FORTRAN, ALGOL, and PL-I go a long way towards this goal, but still require the programmer to learn to structure his thoughts into the rigid syntax of the machine. Ideally, a person with a problem to solve should be able to "discuss" it with a computer using the language he knows best--natural language--and then the computer would tell him the answer. While the current state-of-the-art in computational linguistics is still a long way away from this ideal, some programs have been written [1,2,3,4,5,6] which can "understand" questions and give answers pertaining to a suitably small area of expertise, or "world."

While the smallness of these "worlds" distresses the computational linguist interested in the long range ideal, small worlds do exist in which even simple discourse can be very beneficial. One of these small worlds is the world of computer graphics modeling. The world of computer graphics is especially suited to natural language input because the depth of understanding exhibited by the computer is immediately apparent in the visual feedback generated by it. Also since language probably evolved as a medium for describing things people saw--the location of food; descriptions of dangerous animals; hunt strategies--the development of a model which relates visual and verbal stimuli would be an ideal kernel from which to evolve models relating the other basic senses and finally more abstract senses such as emotion. This dissertation documents a conceptual model of this type which models visual and verbal stimuli in a common, multidimensional Concept Space. In this model, visual structures as displayed on a color computer graphic system exist as hierarchical structures of points in the Concept Space. A transformation also is shown which allows these structures to be directly specified using imperative sentences in an expandable subset of natural English. An implementation of some of the features of the model has been written in SPITBOL [7], a compiler version of SNOBOL [8]. This program, ENGRAF, allows English language specification of objects to be modeled on the color computer graphics facility at North Carolina State University [9,10,11]. In operation, ENGRAF translates simple English commands into the conceptual space model and then issues appropriate subroutine calls to the graphics system to display the structures thus constructed.

1.2 Dialog

The following interaction documents the level of sentence understanding exhibited by ENGRAF. The program generates three basic commands to the connected graphics system: *LOAD(), *MODIFY(), and *DELETE(). In this system, all output beginning with an asterisk (*) would be transparent to the operator so that in most situations his feedback would consist of OK... plus the visual feedback of seeing the screen image change. The arguments which the program generates for these commands are numeric values representing the log of the feature's actual or the actual value itself, whichever is appropriate. The order in which these arguments appear is shown below:

<u>1.2.1</u> *LOAD (object #, shape, log width, log height, log depth, log size, log transparency index, log texture index, redness, greenness, blueness, brightness, x rotation, y rotation, z rotation, x location, y location, z location)

A *LOAD command is issued by ENGRAF whenever an imperative sentence is recognized which contains the verb "draw" or one of its synonyms.

<u>1.2.2</u>
```
DRAW A ROUGH RED CIRCLE AT -1, 2.2, 5.55
*LOAD (1, CIRCLE, -, -, -, , , 1, 1, 0, 0, , , , , -1,2.2, 5.55)
```

In its current evolution, ENGRAF recognizes only two types of nouns (BOXES and CIRCLES); however other nouns may be added easily to the lexicon. These primitive shapes are also stored in the graphics system and may be used to define more complicated structures as soon as this capability is added to ENGRAF. The program already performs extensive error checking to insure that the correct meaning is determined.

<u>1.2.3</u>
```
MOVE A CIRCLES TO 3, 4, 5
THE NUMBER FEATURE OF "A CIRCLES" DOES NOT COMPUTE
```

In 1.2.3, the determiner "a" implies that the form of the following noun "circles" should be singular. Since the number features of these two words do not agree, the system will not process the sentence.

<u>1.2.4</u>
```
MOVE 3 CIRCLES TO 3, 4, 5
THERE IS ONLY ONE FIGURE OF THAT DESCRIPTION
```

In this sentence, the determiner "3" and noun "circles" are in agreement; however in inspecting the data base to locate the specified objects the system was only able to locate one of them. Thus the operator was advised of this fact and no further action was taken.

<u>1.2.5</u> MOVE THE BLUE CIRCLE TO 3, 4, 5

```
MOVE THE BLUE CIRCLE TO 3, 4, 5
THERE ARE 0 FIGURES OF THAT DESCRIPTION
```

Again, in searching the data base for a blue circle, the system only finds the red one created in 1.2.2. The operator is advised accordingly.

<u>1.2.6</u>

```
MOVE THE ROUGH CIRCLE TO 3, 4, 5
*MODIFY (1, CIRCLE, -, -, -, , , 1, 1, 0, 0, , , , , 3, 4, 5)
OK
```

Here, the specified object was located in the data base. The command was understood to require modification of an existing object, so the appropriate *MODIFY command was issued.

<u>1.2.7</u>

```
MOVE IT TO 3, 3, 3
*MODIFY (1, CIRCLE, -, -, -, , , 1, 1, 0, 0, , , , , 3, 3, 3)
OK
```

As an alternative to the tedious specification of objects by their names, ENGRAF allows anaphoric references to be made. This feature requires the system to remember which objects were referenced in previous frames. ENGRAF keeps a complete time line of events so that errors in interpretation or operator intent may be easily corrected. This feature is discussed later.

<u>1.2.8</u>

```
MAKE THEM LARGE
"THEM" IS UNDEFINED
```

Incorrect use of anaphoric referents , as above when no plural objects have been discussed yet, will not be acceptable to the system.

<u>1.2.9</u>

```
MAKE IT VERY TALL
THE HEIGHT FEATURE OF "MAKE IT VERY TALL" DOES NOT COMPUTE
```

The program also allows nouns to disallow certain modifiers or adjectives. Since a tall circle becomes an ellipse, the dictionary entry for circle specifies that it may not be tall or wide, and since a circle is a two-dimensional object, it may also not have depth (see "-" features in 1.2.2). In 1.2.9, "IT" references the circle last modified in 1.2.5 so that making "IT" very tall is anomalous and is appropriately rejected.

```
1.2.10  MAKE IT MUCH MORE TRANSPARENT AND ROUGHER
        *MODIFY (L, CIRCLE, -, -, -, , 1. , 2. , 1, 0, 0, , , , ,
        3, 3, 3)
```

Modifications which are allowed by the noun-modifier restrictions may be compounded as in 1.2.10. In this sentence, "much" is assumed by the system to modify both the words "more transparent" and "rougher." If this interpretation of the sentence does not please the operator, he has the option to cause the system to back up over the offending command and then replace it with another command which the system interprets correctly as in 1.2.11 below.

```
1.2.11  NO, MAKE IT MUCH MORE TRANSPARENT AND A TEENSY BIT ROUGHER
        *MODIFY (1, CIRCLE, -, -, -, , , 1, 1, 0, 0, , , , , 3, 3, 3)
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1, 0, 0, , , , ,
        3, 3, 3)
        OK
```

In this sentence, the preceding "NO," causes the system to move back in time to prior to the preceding instruction before performing the indicated modification. This "simple-minded" treatment of ambiguity is necessitated by the single processor nature of the SPITBOL implementation and could be made more natural by the addition of multiprocessing features inherent in other newer Artificial Intelligence languages [12].

Of special interest in this program is the ability to define multiple word idioms such as "more transparent" and "a teensy bit" as single syntactic units. Use of this feature models the behavior of human speakers, we believe, in situations where the sum of the meanings of the constituent words of the idioms does not equal the total meaning of the sum of the words.

```
1.2.12  COLOR IT A LITTLE BIT REDDER
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1. 25, 0, 0,
        , , , , 3, 3, 3)
        OK
```

```
1.2.13  NO, COLOR IT PURPLE
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1, 0, 0, , , , ,
        3, 3, 3)
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1, 0, 1, , , , ,
        3, 3, 3)
        OK
```

```
1.2.14  XROTATE IT 45 DEGREES AND YROTATE IT 27 DEGREES
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1, 0, 0, , 45,
        , , 3, 3, 3)
        *MODIFY (1, CIRCLE, -, -, -, , 1. , 1. 049999, 1, 0, 0, , 45,
        27, , 3, 3, 3)
        OK
```

1.2.12 through 1.2.14 illustrate additional use of the idiom feature, the back up feature, and the ability to specify object rotations respectively.

```
1.2.15  DRAW 3 VERY SMOOTH SHORT AND VERY OPAQUE BOXES
        *LOAD (2, BOX, , -1, , , -2, -2, , , , , , , , , ,)
        *LOAD (3, BOX, , -1, , , -2, -2, , , , , , , , , ,)
        *LOAD (4, BOX, , -1, , , -2, -2, , , , , , , , , ,)
        OK
```

ENGRAF also allows multiple objects to be created and modified within the same command (1.2.15). Subsequent use of complicated anaphoric references (1.2.16) and modifications relative to other existing objects are also possible (1.2.17) with continued checking to make sure that only sentences which make sense are executed (1.2.18, 1.2.19).

<u>1.2.16</u>
```
MOVE 2 OF THEM TO 1, 2, 3
*MODIFY (2, BOX, , -1, , , -2, -2, , , , , , , , , 1, 2, 3)
*MODIFY (3, BOX, , -1, , , -2, -2, , , , , , , , , 1, 2, 3)
OK
```

<u>1.2.17</u>
```
MAKE THEM MORE TRANSPARENT THAN THE PURPLE CIRCLE AT 3, 3, 3
*MODIFY (2, BOX, , -1, , , 1. 5, -2, , , , , , , , , 1, 2, 3)
*MODIFY (3, BOX, , -1, , , 1. 5, -2, , , , , , , , , 1, 2, 3)
OK
```

<u>1.2.18</u>
```
MAKE THEM AS ROUGH AS THE BLUE CIRCLE
THERE ARE 0 FIGURES OF THAT DESCRIPTION
```

<u>1.2.19</u>
```
COLOR 4 OF THEM BLUE GREEN
THERE AREN'T THAT MANY OF THEM
```

<u>1.2.20</u>
```
MAKE 1 OF THEM AS ROUGH AS THE PURPLE CIRCLE AND MAKE THE REST
OF THEM SMOOTH
*MODIFY (2, BOX, , -1, , , 1. 5, 1. 049999, , , , , , , ,
1, 2, 3)
*MODIFY (3, BOX, , -1, , , 1. 5, -1, , , , , , , , , 1, 2, 3)
OK
```

Compound sentences as in 1.2.20 are allowed so that the operator may make full use of the features discussed above.

<u>1.2.21</u>
```
REMOVE A BOX AT 1, 2, 3
*DELETE (3)
OK
```

Objects which are no longer of interest to the operator may be deleted as in 1.2.21, assuming that they exist. If not, the operator will be informed of his error (1.2.22).

```
1.2.22  REMOVE THE TALL BOX AT 1, 2, 3
        THERE ARE 0 FIGURES OF THAT DESCRIPTION
```

```
1.2.23  COLOR A BOX TALL
        THE HEIGHT FEATURE OF "COLOR A BOX TALL" DOES NOT COMPUTE
```

Extensive checking for semantic agreement between the various portions of an input sentence insures that anomalous commands which are grammatically correct as above (1.2.23) will nevertheless be rejected.

```
1.2.24  DRAW A TALL RED BOX AND A SMALL BRIGHT BLUE CIRCLE
        *LOAD (5, BOX, , 1, , , , , 1, 0, 0, , , , , , ,)
        *LOAD (6, CIRCLE, -, -, -, -1, , , 0, 0, 1, 4, , , , , ,)
        OK
```

```
1.2.25  COLOR A TALL BOX GREEN AND MOVE IT TO 7, 7, 7
        *MODIFY (5, BOX, , 1, , , , , 0, 1, 0, , , , , , ,)
        *MODIFY (5, BOX, , 1, , , , , 0, 1, 0, , , , , 7, 7, 7)
        OK
```

```
1.2.26  DRAW 3 CIRCLES AT 3, 6, 9 AND COLOR 2 OF THEM RED
        *LOAD (7, CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
        *LOAD (8 , CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
        *LOAD (9 , CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
        *MODIFY (7, CIRCLE, -, -, -, , , , 1, 0, 0, , , , , 3, 6, 9)
        *MODIFY (8 , CIRCLE, -, -, -, , , , 1, 0, 0, , , , , 3, 6, 9)
        OK
```

```
1.2.27  GO BACK IN TIME
        *MODIFY (7, , , , , , , , , , , , , , , ,)
        *MODIFY (8, , , , , , , , , , , , , , , ,)
        *DELETE (7)
        *DELETE (8)
        *DELETE (9)
        OK
```

```
1.2.28  GO BACK IN TIME
        *MODIFY (5, BOX, , 1, , , , , 1, 0, 0, , , , , , ,)
        *MODIFY (5, BOX, , 1, , , , , 1, 0, 0, , , , , , ,)
        OK
```

<u>1.2.29</u>
```
GO FORWARD IN TIME
*MODIFY (5, BOX, , 1, , , , , 0, 1, 0, , , , , , ,)
*MODIFY (5, BOX, , 1, , , , , 0, 1, 0, , , , , 7, 7, 7)
OK
```

<u>1.2.30</u>
```
GO FORWARD IN TIME
*LOAD (7, CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
*LOAD (8, CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
*LOAD (9, CIRCLE, -, -, -, , , , , , , , , , , 3, 6, 9)
*MODIFY (7, CIRCLE, -, -, -, , , , 1, 0, 0, , , , , 3, 6, 9)
*MODIFY (8, CIRCLE, -, -, -, , , , 1, 0, 0, , , , , 3, 6, 9)
OK
```

<u>1.2.31</u>
```
GO FORWARD IN TIME
ATTEMPT TO GO FORWARD IN TIME FAILED
```

The above sequence (1.2.24 to 1.2.31) illustrates the completeness of the session history which is maintained by ENGRAF. This feature may be used by the operator to review the development of his image or to set up a sequence of frames to be used in making an animated film. A program which performed an interpolation between the successive time frames (Parke [13]) would generate movies. Notice that an attempt to go into the future is not allowed by ENGRAF.

<u>1.2.32</u>
```
"HUGE" IS VERY LARGE
I UNDERSTAND
```

<u>1.2.33</u>
```
"SKY BLUE" IS BLUE AND GREEN
I UNDERSTAND
```

<u>1.2.34</u>
```
DRAW A HUGE SKY BLUE BOX
*LOAD (10, BOX, , , , 2, , , , 1, 1, 0, , , , , ,)
OK
```

<u>1.2.35</u>
```
GOODBY
BYE, SEE YOU LATER
```

Since the initial vocabulary defined for the system will always be inadequate for some users, ENGRAF allows new words to be defined using combinations of other words which the system already knows. This is possible due to the multidimensional representation of meaning used which allows word meanings to be "added" together to produce a new meaning representation which is in the same vector format as that of the original words. Thus in 1.2.32, the word "huge" has been defined to be equivalent to the words "very large." This definition creates an initial n-space meaning for the word "huge" which may later be modified independently of the words "very" and "large." In a similar manner, the two-word substring "sky blue" is defined initially to be equal amounts of "blue" and "green" in 1.2.33. It would not be difficult to implement a word-modification ability in ENGRAF so that

1.2.35 MAKE THE WORD "SKY BLUE" BLUER

would be correctly interpreted. In 1.2.34 we demonstrate the ability of using the words which were created earlier, and in 1.2.35 the session is terminated.

2. BACKGROUND

2.1 Parsing

In order for a computer program, such as ENGRAF, to "understand" an English sentence, the program must contain a model of the language which allows the meanings of individual words to be determined and somehow agglomerated into the meaning of the sentence as a whole. Some linguists have broken this process down into three levels of understanding: Syntactic, Semantic, and Conceptual. The process of assigning a syntactic structure to a sentence is called parsing.

Depending on the class of English sentences which a given program must "understand," the parsing algorithm used may vary in complexity. Probably the simplest parsing strategy is "keyword parsing." Keyword parsers such as Bobrow's STUDENT [6] and Shapiro's ARTIST [14] contain sentence patterns with "blanks" in them. In operation, each "blank" in each sentence pattern is allowed to match a small subset of words, and the meaning of the sentence is determined by the sentence pattern used plus features associated with the words found in the "blanks." In 2.1.1 we show an example of such a sentence pattern taken from ARTIST (p 5).

```
                                                   | CENTER     |
                                                   | LEFT SIDE  |
2.1.1  DRAW A CIRCLE OF RADIUS <dec#> WITH         | RIGHT SIDE |
                                                   | TOP        |
       AT THE  <point>                             | BOTTOM     |
```

where

<dec#> Fortran type real number punched with a decimal point (no exponents allowed)

```
<point> →  | POINT (<dec#>, <dec#>)                    |
           | <contact point> OF THE <previous part>    |
```

$$\text{<previous part>} \rightarrow \left|\begin{array}{l}\text{FIRST}\\ \\ \text{TENTH}\\ \text{LAST}\\ \Omega\end{array}\right| \left|\begin{array}{l}\text{<previously drawn component>}\\ \\ \text{<previously drawn object>}\end{array}\right|$$

As the number of sentence patterns grows with the sophistication of the system, the problem of matching all existing patterns against a single input sentence results in a combinatorial explosion. Thus "bottom-up" parsing techniques such as this one have given way to "top-down" predictive techniques which utilize list structured grammars and need examine only a small subset of possible interpretations at each point in the parsing.

These modern parsing techniques are based upon the early work of Chomsky [15], who inaugurated a currently popular model of language structure called Transformational Grammar. In this grammar, sentences are seen as "surface structures" which represent, at a conversational level, kernels of information contained in a "deep structure." The transformation from surface structure to deep structure is many-to-one so that a single deep structure can generate or be generated by a number of different surface structures. Verbal communication is seen in this model to be composed of: (1) a deep structure is conceived by the speaker and transformed (2) into a surface structure which is then articulated. Upon receiving the surface structure (3) the listener performs an inverse transformation (4) to determine the deep structure transmitted. Effective communication requires that the initial transformation of the speaker have an inverse transformation which closely approximates that of the listener. Otherwise the sentence may be ambiguous (multiple valued inverse), anomalous (null valued inverse), or just incorrect (different inverse). The nature of this deep structure has not been mathematically formalized, although linguists have developed many symbolic models which use it. In this dissertation, we propose that deep structures may be modeled as structures in a multi-dimensional space and that the transformations may be defined mathematically.

In Chomsky's theory, the medium of these transformations is known as a grammar, and a grammar may be formally defined by a series of rewrite rules of the form

<u>2.1.2</u> $\Phi \rightarrow \Psi$

which indicate that any symbol Φ appearing in the deep structure may be replaced by Ψ and the process repeated until there are no more Φ in the string. Chomsky defined four classes of grammars using this technique by their power to generate correct surface structures in a language. A given grammar is said to model a language if it generates only correct strings of the language. It is not constrained to generate all correct strings, however. Such grammars are defined in terms of rewrite rules (2.1.2) utilizing terminal (V_T) and non-terminal (V_N) symbols. For English grammars, the set of terminal symbols $\{V_T\}$ is usually chosen to be the set of syntactic parts-of-speech. Non-terminal symbols are usually chosen to represent larger syntactic units such as Noun Phrase, Verb Phrase, Prepositional Phrase and Sentence.

The following are formal definitions of the four classes of grammars rated in descending order of their generative power and in ascending order of their simplicity. All transformations are of the form 2.1.2 where Φ and Ψ are defined separately to be:

2.1.3 I. Type 0 or Unrestricted

$$\Phi \ \varepsilon \ \{V_N + V_T\}^*; \quad \Psi \ \varepsilon \ \{V_N + V_T\}^*$$

2.1.4 II. Type 1 or Context-Sensitive

$$\Phi = \alpha \ A \ \beta; \quad \Psi = \alpha \ \omega \ \beta$$

where

$$\alpha, \omega, \beta \ \varepsilon\{V_N + V_T\}^*; \quad A \ \varepsilon\{V_N\}$$

also

$$A, \omega \neq \{\Omega\}$$

2.1.5 III. Type 2 or Context-Free

$$\Phi \ \varepsilon\{V_N\}; \quad \Psi\{V_N + V_T\}^*$$

2.1.6 Type 3 or Finite State

$$\Phi \ \varepsilon\{V_N\}; \quad \Psi = V_T \ V_N \text{ or } V_T$$

Finite State grammars as defined above are the easiest to implement, and they may be implemented using the simple finite state sequential machines of automata theory [16]. Unfortunately, it may be shown that there are a large class of English sentences (those with imbedded sentences and relative clauses, for example) which cannot be described with Finite State grammars. The addition of recursion (the ability of a non-terminal symbol to replace itself), however, allows recursive transition networks to possess the generative power of Context-Free grammars. These grammars are able to describe a wider class of English sentences but a more powerful formalism, attributed to Woods [5], is the Augmented Transition Network (ATN). ATNs use the contents of

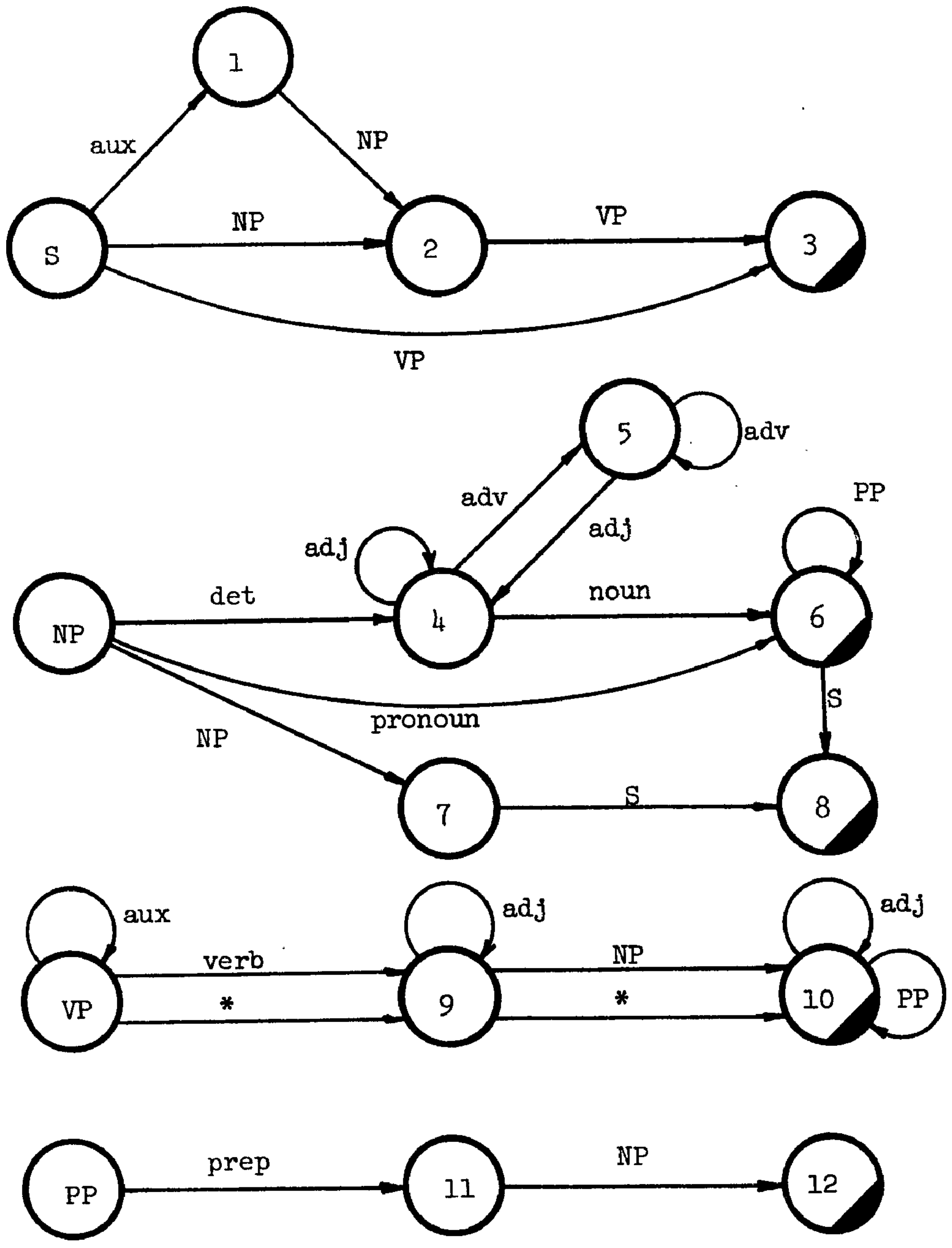
*any string except the null string $\{\Omega\}$

global registers to context-modify the topology of recursive transition networks. Thus ATNs have the generative power of Unrestrictive grammars. In Figure 2.1 we illustrate a simplified ATN grammar which is able to describe a wide class of English sentences. Since language understanding programs are mostly concerned with the inverse transformation from surface structure to deep structure, this grammar has been so structured. Also the register tests have been omitted for simplicity's sake. In this grammar, the branches between nodes of the network are conditioned both by the syntactic part-of-speech shown and the context register contents which are not shown. Parsing a sentence involves finding a complete path through the sentence network (S). Whenever a non-terminal symbol (e.g. NP) is found, a call to the appropriate sub-network is made. If this call succeeds then the branch is conditioned and a state transition is made. Whenever a terminal symbol is found, the part-of-speech of the next word in the input sentence is found and if it conditions the branch, a state transition is made. A network may succeed when a ◉ is reached.

Using the ATN Grammar of figure 2.1, the sentence

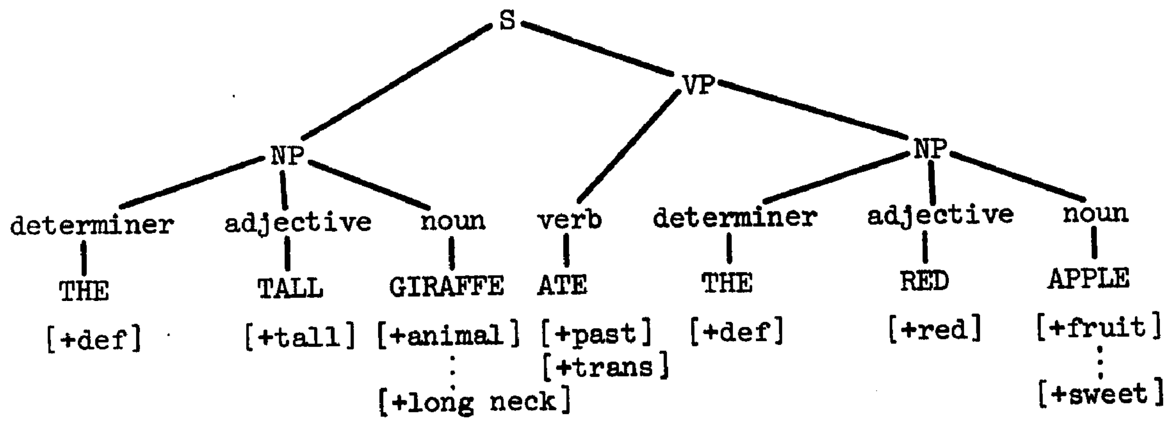
<u>2.1.7</u> The tall giraffe ate the red apple
would be reduced to a hierarchical structure representation similar to figure 2.2.

If we assign to each word a set of features as the [ ] words in figure 2.2, we may then merge these features to form a kind of deep structure for the sentence in Figure 2.3.



Augmented Transition Networks

Figure 2.1



<u>Sample Parsing</u>

Figure 2.2

SUBJECT	VERB	OBJECT
giraffe	eat	apple
[+def]	[+past]	[+def]
[+tall]	[+trans]	[+red]
[+animal]		[+fruit]
⋮		⋮
[+long neck]		[+sweet]

<u>Sample Deep Structure</u>

Figure 2.3

2.2 Semantics

The bracketed features which we assigned to the words in Figure 2.2 and used to form the deep structure of Figure 2.3 represent the meanings which the system associates with these words. Semantic knowledge of this kind will allow the program to reject or accept sentences based upon a knowledge of its meaning even though it may be syntactically correct to the parser. Thus even though the sentence

2.2.1 *The red apple ate the tall giraffe

is syntactically correct, a semantic analysis will quickly notice that "apples" cannot "eat" anything and so will reject the sentence. Programs of this nature, as exemplified by Winograd's SHRDLU [4], typically combine the syntactic and semantic analysis into one process. They do this by allowing previously parsed words in the sentence to define extra tests (other than syntactic) on the branches of the network which have yet to be traversed. This is closely related to the augmentation in ATNs since the effective topology of the network may be altered by these semantic restrictions.

As examples of semantic restrictions in action, consider the sentence

2.2.2 A tall giraffe is eating a red apple.

Parsing of the singular determiner "a" in 2.2.2 requires that the noun ultimately parsed have singular form. Once this agreement is established, the noun phrase "a tall giraffe" is similarly established to represent a singular subject. This condition is required by the singular form of "is eating" since

2.2.3 *A tall giraffe are eating a red apple

would be incorrect. Also associated with the verb "eat" is the restriction that its subject have the feature [+ animal] and that its object have the feature [+ edible]. Note that while "apple" may not explicitly have this feature, it may be implied from the feature [+ fruit] by an implication mechanism within the program. Such implication mechanisms reduce the amount of explicit semantic information which must be stored about a word although they require more execution time to operate.

Critical to the semantic analysis of a sentence is the formalism used to represent knowledge. Winograd used features such as those above associated with individual words, and stored knowledge about his "blocks world" as assertions and theorems in a LISP-based subset of Hewitt's PLANNER [17]. In his system, questions were expressed as theorems to be proven, and declarative sentences added new assertions to the data base. Winograd's parser, called PROGRAMMER, embodied the syntactic and semantic methods of the previous section although in a different form than ATN's.

Quillian [18,19] introduced the idea of a semantic network for storage of information. In a semantic network, information is stored in much the same way as in a normal dictionary--words are defined by references to other words in the dictionary which are similarly defined. The major shortcoming of semantic networks lies in the lack of any absolute reference for meaning. For example, a person with no understanding of the English language could wander through an English dictionary forever looking up definitions of definitions of definitions ... in a reductio ad absurdum, and never find an accurate definition of a single word--unless he had a dictionary which could provide absolute meaning (translation from his language) for at least the most common words used in the English dictionary. A semantic network suffers from the same difficulty since the meaning of a word is defined solely by its relationships with other words, etc. The intuitive beauty of semantic nets, however, lies in the fact that they bear a strong resemblance to the neural networks which make up the human brain. We will show later how our multidimensional concept space model has this same intuitive flavor.

An extension of the idea of a deep structure is the "deep case structure" of verbs proposed by Simmons [3]. Simmons recognized that the allowed structure of a sentence is primarily influenced by its verb. For example, transitive verbs require an object to act upon, whereas intransitive verbs do not. Simmons' case structures are more involved than this, however, and he defines a number of "cases" (relationships of noun phrases to the verb) besides the simple subject and object discussed previously. In addition to the cases of a given verb, a case structure contains information about its "modality." Modality in this model is further broken down to represent Tense, Aspect, Form, Mood, Essence, Modal, Manner, and Time features of the verb. These are fully discussed in [3]. The important features of case structures from our point of view is the fact that each verb contains restrictions about the types of noun phrases which can act in each case relation to the verb. Since each verb may have several cases (CAUSAL ACTANT, THEME, LOCUS, SOURCE, GOAL), existing within a given sentence, these restrictions may be used to determine which noun phrase belongs in which case, and whether sufficient information exists within the sentence for comprehension, or must be implied from previous sentences. These restrictions may be easily embodied in our conceptual space model as we will show in Chapter 3.

2.3 Concepts

At a level of understanding higher than the syntactic/semantic understanding of individual sentences is the conceptual relating of sentences within a dialog. This level of understanding is necessary since normal English sentences rarely provide all of the information necessary for their total comprehension. To do so would be ponderous and redundant. Rather, a great deal of information about a sentence may only be found by inspecting its context within the other sentences of the dialog, or by making intuitive inferences based upon previous experiences in the same subject area. Schank [2, 20] has proposed a model of conceptual understanding at this level which can account for these inferential abilities and also offers a model for high level conceptual understanding of complex dialogs. In this model, Schank proposes that sentences and dialogs can be mapped into Conceptual Structures consisting of primitive action verbs (ACTS) and physical objects called Picture Producers (PPs). PPs can be said to elicit a picture in the "mind's eye" of an individual, and may be modified by Picture Aiders (PAs) for variation. Similarly, ACTS may be modified by Action Aiders (AAs) for more subtle specification of verb behavior. Schank connects these building blocks of "conceptual quanta" into complex structures by connecting different types of links between these elements to show their interrelations and dependencies. Intuitive feeling indicated that these structures could be modeled in multidimensional spaces which led to the development of the model described here.

Another interesting development at the conceptual level is Bruce and Schmidt's [21] Motivational Rules for deciding when a given action should be performed. These rules are based upon the assumption that actions are motivated: "...that is if person A performs act F then we infer that he can do F and that he has chosen to do F." [21]

According to this theory, people are motivated into action by the potential gain the action will produce for themselves (Hedonism) or their associates (Extended Hedonism); as a "response in kind" to an action intentionally performed on them (Reciprocity); or because an action is an expected part of normal behavior (Normative). An action may also be initiated out of generosity or good will (Disposition). Since ENGRAF is motivated to perform actions for one reason only (Normative), application of these rules to this system is not germane.

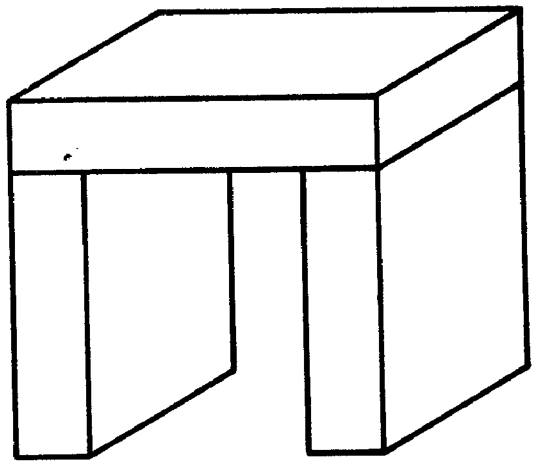
Of more direct importance to the development of ENGRAF is the use of "time frames" by Winograd [4] and Weissman [22] to solve problems of intent and anaphoric reference (Winograd) and to solve temporal strategies (Weissman). Both programs use extensive histories of past actions to solve problems peculiar to their "worlds." The application of time frames in ENGRAF will be discussed in a later section.

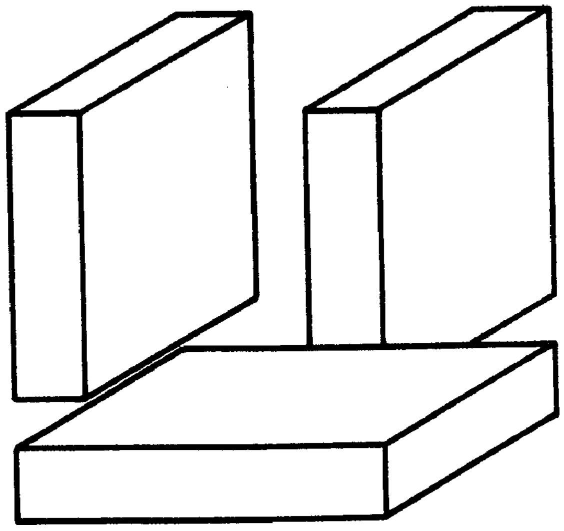
A powerful concept-like structure for storing structural information about objects has been implemented by Winston [23]. Winston's structure is primarily intended as a data base for scene analysis work although it retains a high degree of generality. In this model, for example, the system would be presented with an instance of an arch (Figure 2.4) along with several non-examples of an arch--a non-example being a structure which differed from being an arch "by one or a few crucial deficiencies." [23] The system would then construct a conceptual model of an arch which would exclude all of the non-examples shown (Figure 2.5).

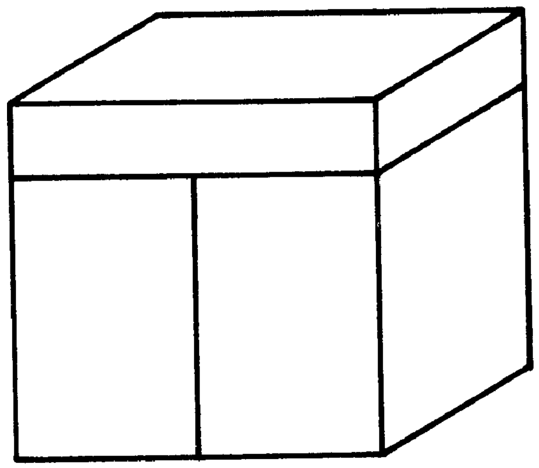
Hierarchical representations of physical objects of this nature are often used as data structures in computer graphics modeling, and a similar structure is used in ENGRAF. Also the use of non-examples to speed the learning process fits nicely within the concept space model upon which ENGRAF is based.

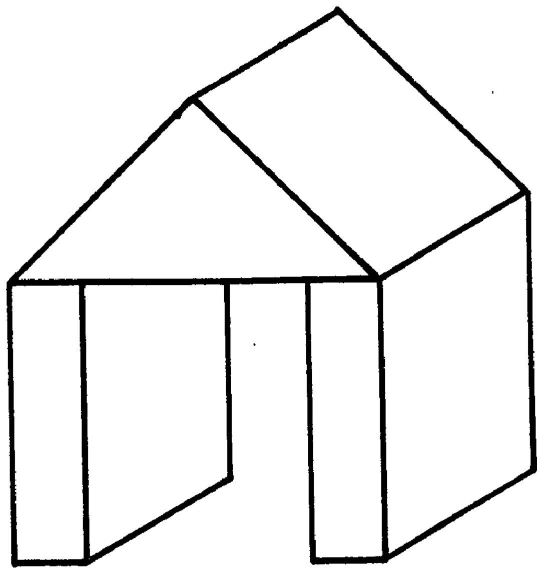
2.4 Ambiguity

The assignment of meaning to English sentences by a program is greatly complicated by the fact that words often have multiple meanings and that the determination of the correct meaning may require information explicitly stated in previous sentences of the discourse or implied by the general subject area. There are two types of


Arch

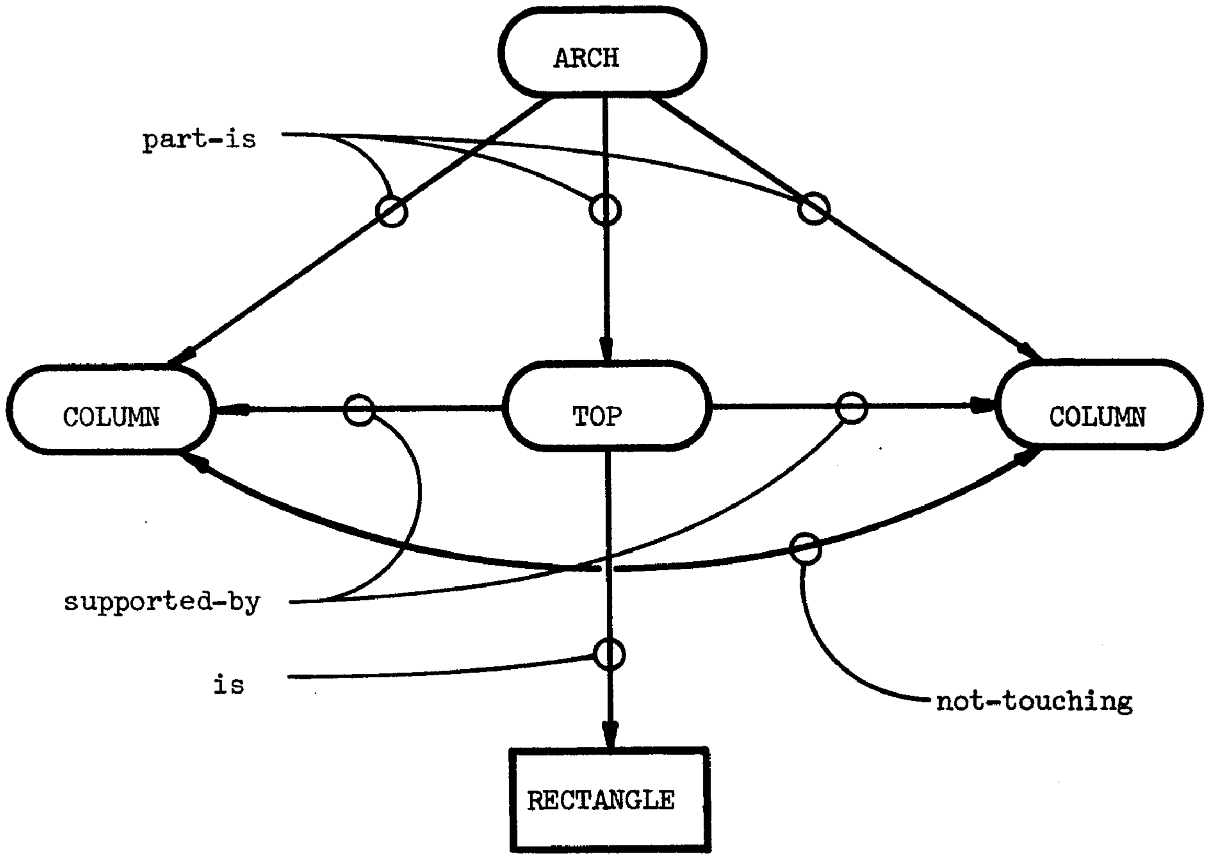

Near Miss


Near Miss


Near Miss

Examples and Non-examples

Figure 2.4



<u>An Arch Description</u>

Figure 2.5

ambiguity which are commonly observed in these programs.

The simplest kind of word ambiguity is called syntactic ambiguity. It results when a given word may have two or more grammatical parts-of-speech such as in

2.4.1 Duck ahead.

The syntactic ambiguity inherent in this sentence results from the fact that "Duck" may be either a verb as in "Duck down ahead" or a noun as in "A duck is ahead." In the graphics world of ENGRAF, syntactic ambiguity would be involved in the interpretation of 2.4.2 and 2.4.3:

2.4.2 Draw a deep rough boy.

2.4.3 Draw a deep blue box.

This ambiguity is caused by the fact that "deep" may either be an adjective modifying "box" or an adverb modifying "rough" or "blue." The ambiguity in 2.4.2 is easily resolved by noting the fact that as an adverb, "deep" may only modify a color (semantic restrictions). Since "rough" in this sentence is not a color, "deep" must be an adjective. The ambiguity in 2.4.3 is much more difficult to resolve, however, since blue is indeed a color. The technique used by ENGRAF in situations of this nature is to pick consistently one interpretation and then let the operator use the program's back-up ability to correct in the next sentence if the choice was incorrect. By maintaining consistency, the operator will quickly learn the interpretation which the program will choose and he can thus learn to get the desired response.

A more difficult type of ambiguity to deal with is semantic ambiguity. Semantic ambiguity arises when a word has two or more meanings all of which are the same part of speech:

2.4.4 The record was broken.

Here one has no way of guessing whether the "record" in question was a track record, for example, or a phonograph record. Semantic ambiguity may often be treated by examining the context within which the ambiguity occurred. The meaning of "record" is clearer in the two following sentences 2.4.5 and 2.4.6:

2.4.5 As John crossed the finish line, the record was broken.

2.4.6 As it slid from the table, the record was broken.

Semantic ambiguity is rarely a problem in small worlds such as the realm of ENGRAF, since the context of all words is known *a priori* by the nature of the world itself.

3. AN N-DIMENSIONAL CONCEPT SPACE

3.1 Dimensionality

All of the language models in the last chapter have one feature in common: they are qualitative, symbolic models, visualized primarily as two-dimensional networks. We feel that a more quantitative depth of understanding may be achieved by expanding the dimensionality of these models to encompass higher order spaces than the two-dimensional space of pen and paper. In order to understand the nature of this expansion process, however, it will be helpful to review some concepts from the theory of matrices.

A point in a space S_n of dimension n may be uniquely defined by a n-tuple position vector, $\bar{x}$, where $\bar{x}$ may be written:

3.1.1 $\bar{x} = (x_1, x_2, x_3, \ldots., x_{n-1}, x_n)$

In this notation, the x_i may be visualized as being the coordinates of the point in S_n. A space S_n is defined to be a linear space if, for any position vectors $\bar{x}$, $\bar{y}$, $\bar{z}$ and any real numbers α, β, the following properties are true on S_n.

3.1.2 $\bar{x} + \bar{y}$ is a unique vector in the set to which $\bar{x}$ and $\bar{y}$ belong.

3.1.3 $\bar{x} + \bar{y} = \bar{y} + \bar{x}$

3.1.4 $(\bar{x} + \bar{y}) + \bar{z} = \bar{x} + (\bar{y} + \bar{z})$

3.1.5 There exists a vector $\bar{\emptyset}$ such that $\bar{x} + \bar{\emptyset} = \bar{x}$

<u>3.1.6</u> For every vector $\bar{x}$, there exists a vector $-\bar{x}$ such that $\bar{x} + (-\bar{x}) = \bar{\emptyset}$

<u>3.1.7</u> $\alpha\ \bar{x}$ is a vector of the set to which $\bar{x}$ belongs

<u>3.1.8</u> $\alpha(\beta\bar{x}) = (\alpha\beta)\bar{x}$

<u>3.1.9</u> $\alpha(\bar{x} + \bar{y}) = \alpha\bar{x} + \alpha\bar{y}$

<u>3.1.10</u> $1 \cdot \bar{x} = \bar{x}$

Arithmetic operations on vectors differ from the corresponding operations on scalars. We define addition and scalar multiplication by the following:

<u>3.1.11</u> $\bar{x} + \bar{y} = (x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots, x_{n-1} + y_{n-1}, x_n + y_n)$

<u>3.1.12</u> $\bar{x} \cdot \bar{y} = <\bar{x}, \bar{y}> = x_1y_1 + x_2y_2 + x_3y_3 + \ldots + x_{n-1}y_{n-1} + x_ny_n$

The scalar multiplication of 3.1.12 is also called the "inner product" of two vectors. The inner product has some physical significance in that

<u>3.1.13</u> $\bar{x} \cdot \bar{x} = <\bar{x},\bar{x}> = |\bar{x}|^2$ $|\bar{x}|$ = the magnitude of $\bar{x}$ (scalar)

<u>3.1.14</u> $\bar{x} \cdot \bar{y} = <\bar{x},\bar{y}> = |\bar{x}| \cdot |\bar{y}|$ cos Θ where Θ is the interior angle between $\bar{x}$ and $\bar{y}$

We see from 3.1.14 that if two vectors $\bar{x}$ and $\bar{y}$ are at 90° angles from each other, then their inner product is zero. We define this relationship as "orthogonality" and state formally:

<u>3.1.15</u> $\bar{x}$ and $\bar{y}$ are orthogonal iff $\langle \bar{x},\bar{y} \rangle \equiv 0$

A slightly weaker vector relationship than orthogonality is that of "linear independence." If $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n$ are all vectors in a linear space S_n and there exist numbers $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$, not all zero, such that

<u>3.1.16</u> $$\alpha_1\bar{x}_1 + \alpha_2\bar{x}_2 + \alpha_3\bar{x}_3 + \ldots + \alpha_n\bar{x}_n = \bar{\emptyset}$$

then the vectors $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n$ are "linearly dependent." if 3.1.16 implies all $\alpha_i = 0$, then the set of $\bar{x}_i$ are said to be "linearly independent." Clearly, if a set of vectors are orthogonal, they are also linearly independent. The notion of linear independence may be illustrated by noting that the vectors

<u>3.1.17</u> $$\bar{x}_1 = \begin{vmatrix} 1 \\ 2 \\ 0 \\ 4 \end{vmatrix} ; \quad \bar{x}_2 = \begin{vmatrix} -1 \\ 0 \\ 5 \\ 1 \end{vmatrix} ; \quad \bar{x}_3 = \begin{vmatrix} 1 \\ 6 \\ 10 \\ 14 \end{vmatrix}$$

are linearly dependent since they satisfy the equation

<u>3.1.18</u> $$3\bar{x}_1 + 2\bar{x}_2 - \bar{x}_3 = \bar{\emptyset}$$

while no such equation may be found for

<u>3.1.19</u> $$\bar{x}_1 = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} ; \quad \bar{x}_2 = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} ; \quad \bar{x}_3 \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \end{vmatrix}$$

The vectors of 3.1.19 are thus linearly independent (they are also orthogonal).

A finite set of vectors $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ is a "basis" for a linear space S iff they are linearly independent and every $\bar{x} \in S$ is a linear combination of the $\bar{x}_i$. (A basis for S generates S.)

Let S be a linear space. If there exist n linearly independent vectors in S while every set of n + 1 vectors from S is linearly dependent, then S has "dimension" n [24].

From the above, we may note that there are many (in fact, infinite) basis vector sets defined for any n-dimensional space S_n. Indeed any n linearly independent vectors in S_n will act as a basis for S_n. It may also be shown (p. 29 [24]) that given a space S_n of dimension n and any set of r linearly independent vectors $\{\bar{x}_i | i = 1,2,\ldots r<n\}$ there exist vectors $\bar{x}_{r+1}, \ldots, \bar{x}_n$ such that the vectors $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ form a basis for S_n. Thus if we postulate the existence of an n-dimensional space in which concepts may be modeled, the above indicates that we may go about finding a basis for this space in an iterative manner: the only requirement for adding a new vector $\bar{x}_{r+1}$ to the $\bar{x}_1, \ldots, \bar{x}_r$ already found is that $\bar{x}_{r+1}$ be linearly independent to all existing members of the basis set.

3.2 A Concept Space

The idea of using a multidimensional formalism, while new from a computational linguistics point of view, has proven to be of some value in the modeling of other psychological behavior. Frijda [25] has found that the recognition of emotional states in posed subjects behaves in a strongly multidimensional manner. In his experiments, Frijda used a number of photographs taken of a model as she acted out various emotional states. Each of these photographs was then rated by a panel of judges on a set of 27 bipolar adjective pairs. A factor analysis was then performed, and after Varimax rotation 6, orthogonal dimensions were discovered. These dimensions may be roughly defined by their most significant adjective pairs as determined from the factor loadings of Table 3.1 to be: (1) pleasant - unpleasant (2) derisive - mild (3) controlled - uncontrolled (4) directed - undirected (5) amazed - understanding (6) complicated - simple.

From a linguistic point of view, the importance of this experiment lies in the fact that the meanings of a collection of 27 adjective features describing emotional content present in human faces may be represented as linear combinations of the six orthogonal basis vectors found. Using this representation, the meanings of each adjective word are defined as vectors having six degrees of freedom corresponding to the basis dimensions of the derived emotion space. While this experiment in particular used bipolar adjective pairs and so constrained the derived meanings of each pair to be negatives of each other, it does not seem unreasonable to allow each word to have its own unipolar meaning vector representation. If this were allowed, then each of the 54 adjectives used in the study would have been represented as vectors in the emotion space and probably not all of them would have had exact opposites.

Scales	I	II	III	IV	V	VI
1. Controlled--uncontrolled	+.44		-.81			
2. Indifferent--involved		-.56	-.74			
3. Friendly--angry	-.78		-.40			
4. Sleep--tension	+.37			-.88		
5. Abandonment--reserve	+.41	+.40	+.67			
6. Authoritarian--submissive		-.42	-.62	+.59		
7. Startled--relieved	+.79				-.36	
8. Artificial--natural		-.91				
9. Unpleasant--pleasant	+.98					
10. Closed--open			-.79	-.36		-.33
11. Active--passive				+.89		
12. Derisive--mild		-.80				
13. Attension--rejection	+.51	+.43		+.57	+.38	
14. Cool--warm	-.50		-.76			
15. Admiring--despising	+.38	+.66	+.49			
16. Quiet--anxious	-.76		+.45	+.38		
17. Hesitant--determined			+.69		+.45	
18. Sad--happy	+.90					
19. Directed--undirected				+.93		
20. Eagerness--distaste	-.95					
21. Deep--shallow	-.71	+.53				-.36
22. Amazed--understanding				+.43	+.77	
23. Dull--clear	-.55			-.69		
24. Moved--unmoved	-.51	+.41		+.67		
25. Oppressed--free	+.81			+.36		+.31
26. Complicated--simple	+.31			+.35		+.83
27. Withdrawing--approaching	+.85					

<u>Rotated Factor Loadings</u>

Table 3.1

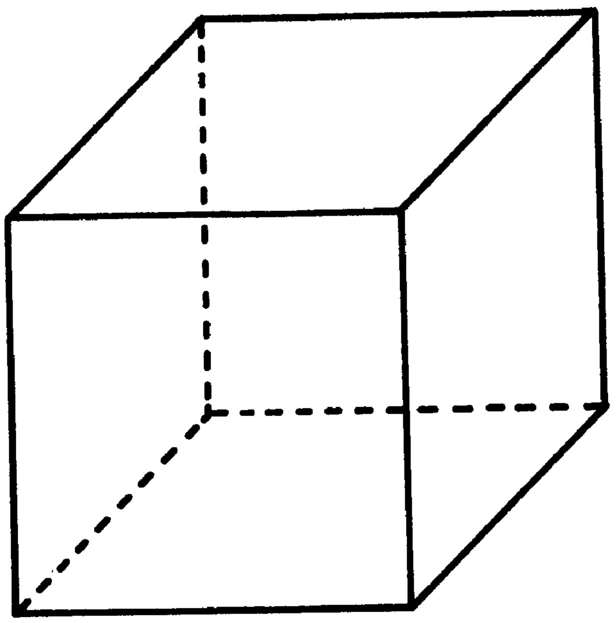
Of interest also in Frijda's experiment was the observed correlation between certain sets of physical facial features and the six factors upon which the emotion space was based. This resulted in the validation of hypothesis that recognition of emotion is based partly upon perceived features as well as a cognitive prediction based upon world knowledge. Since perception of smell and taste information has been shown to behave multidimensionally [26, 27], and since touch and hearing may be argued to represent temperature and pressure information as functions of four dimensions (x, y, z, time), we may explore a possible basis set for visual information also.

Following Frijda's lead in the use of adjectives as basis set members, we consider first the three dimensions HEIGHT, WIDTH, DEPTH which allow scaling information in the orthogonal Cartesian coordinate directions y, x, and z respectively, to be modeled. A fourth dimension, SIZE, may also be included even though it fails the linear independence test of 3.1.16 for reasons seen later. These four "dimensions" allow the meanings of such words as tall, short, large, deep, narrow, and tiny to be defined vectorially. In a computer graphics world in particular, we are interested in features such as these as well as features which describe object rotations and locations. Thus the additional dimensions X ROTATION, Y ROTATION, Z ROTATION, X LOCATION, Y LOCATION, and Z LOCATION seem germane. While the preceding dimensions allow the basic shape of an object to be modified and specified, we also need dimensions in which the surface appearance may be defined. Accordingly we add the three dimensions RED, GREEN, and BLUE which allow color to be defined in terms of the three phosphor colors used in commercial television. Finally, we may also consider some measure of BRIGHTNESS, TRANSPARENCY and TEXTURE to provide additional dimensionality, even though BRIGHTNESS fails the linear independence test also.

All 16 of the above concept-space "dimensions" provide various measures of the visual features belonging to a given real world object. They are, however, useless by themselves unless the shape of the object they are to modify is modeled in a compatible manner.

3.3 Objects

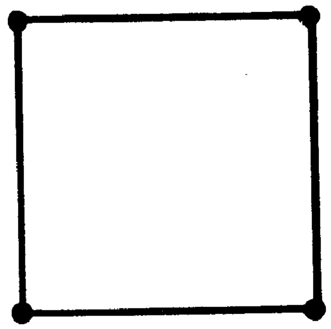
Since the shape of the things which we see plays such an important part in their conceptualization, we need an accurate and quantitative medium to model this information. While we could undoubtedly model a generalized object as a three-dimensional matrix, of suitably fine resolution, into which we would then store the features applicable to each cell, this method would be computationally infeasible due to the large numbers of such cells required. We therefore borrow the "polygon approximation" model of structure from computer graphics and extend it to suit our purposes. In this model, real-world objects are modeled by collections of flat, usually opaque polygons in a Cartesian coordinate space (Figure 3.1). Three-dimensional objects are modeled by sets of polygons which describe their exterior surface contour to an acceptable degree of precision. Obviously the larger the number of polygons used to model curved objects, the more accurately the model will portray reality. Using this formalism, we define a polygon to be an ordered matrix of the



Polygonal Approximation

Figure 3.1

$SQUARE =



$$= \begin{bmatrix} X_1 & X_2 & X_3 & X_4 \\ Y_1 & Y_2 & Y_3 & Y_4 \\ Z_1 & Z_2 & Z_3 & Z_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Primitive Polygon

Figure 3.2

$$\begin{bmatrix} X_1 & X_2 & X_3 \ldots & X_m \\ Y_1 & Y_2 & Y_3 \ldots & Y_m \\ Z_1 & Z_2 & Z_3 \ldots & Z_m \\ H_1 & H_2 & H_3 \ldots & H_m \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ T_{41} & T_{42} & T_{43} & T_{44} \end{bmatrix} \cdot \begin{bmatrix} X_1 & X_2 & X_3 \ldots & X_m \\ Y_1 & Y_2 & Y_3 \ldots & Y_m \\ Z_1 & Z_2 & Z_3 \ldots & Z_m \\ 1 & 1 & 1 \ldots & 1 \end{bmatrix}$$

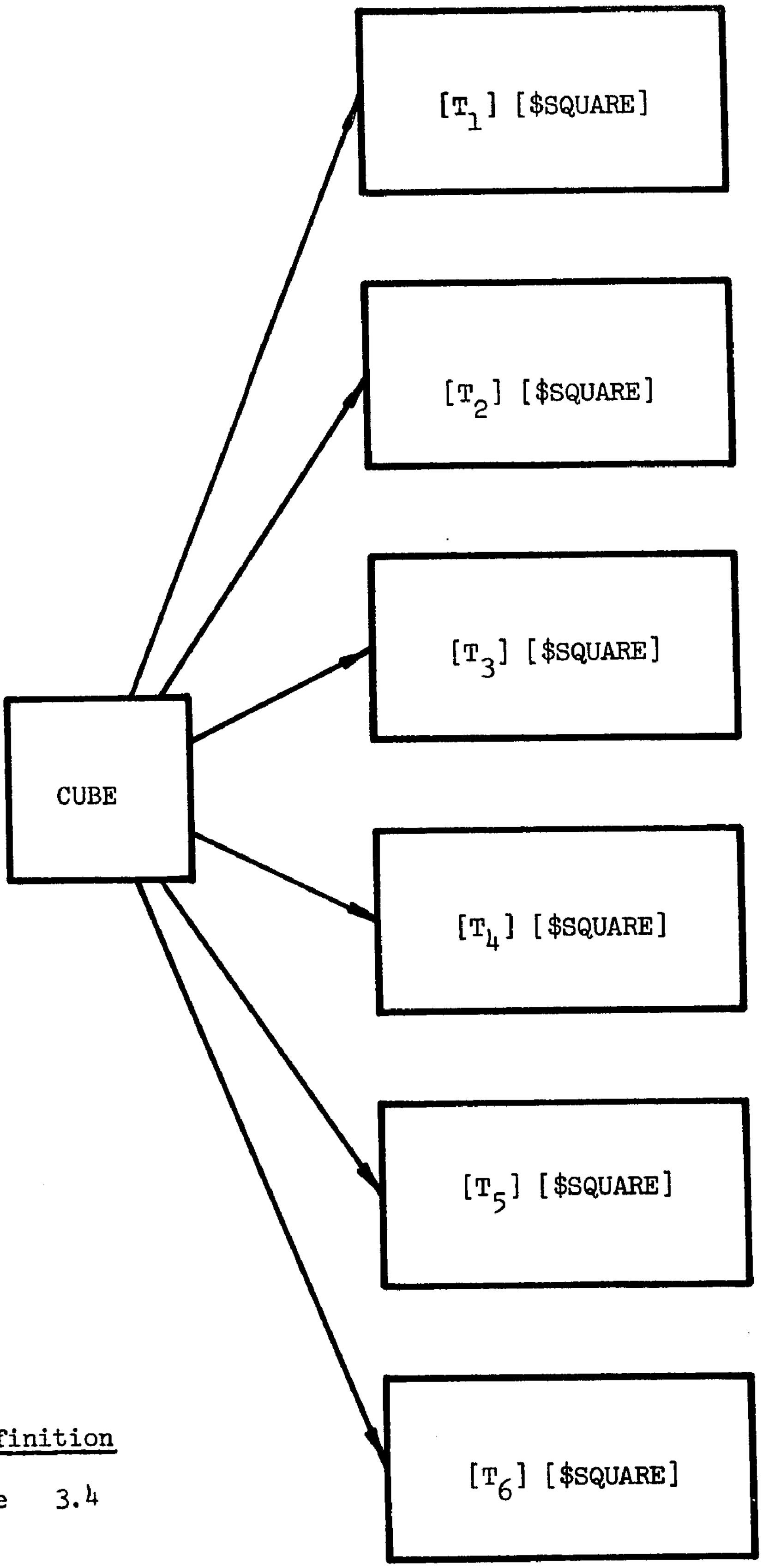
Polygon Matrix Transformation

Figure 3.3

coordinates of its vertices in a homogeneous [28] 4-space (x, y, z, h) (Figure 3.2). The use of homogeneous coordinate representation allows translational and perspective information about the polygon to be expressed in a concise manner but is equivalent to the more standard (x, y, z) notation. If the polygons used to model the structure of an object are defined in a normalized manner, then models may be constructed hierarchically by premultiplying the primitive polygon matrices by transform matrices embodying the correct scaling, translational, and rotational information to move each primitive polygon into its correct position in the model. Figure 3.3 illustrates this transformation process on a single primitive polygon ($SQUARE), and Figure 3.4 shows how a hierarchical structure of transformation matrices and polygons may be constructed to model a cube.

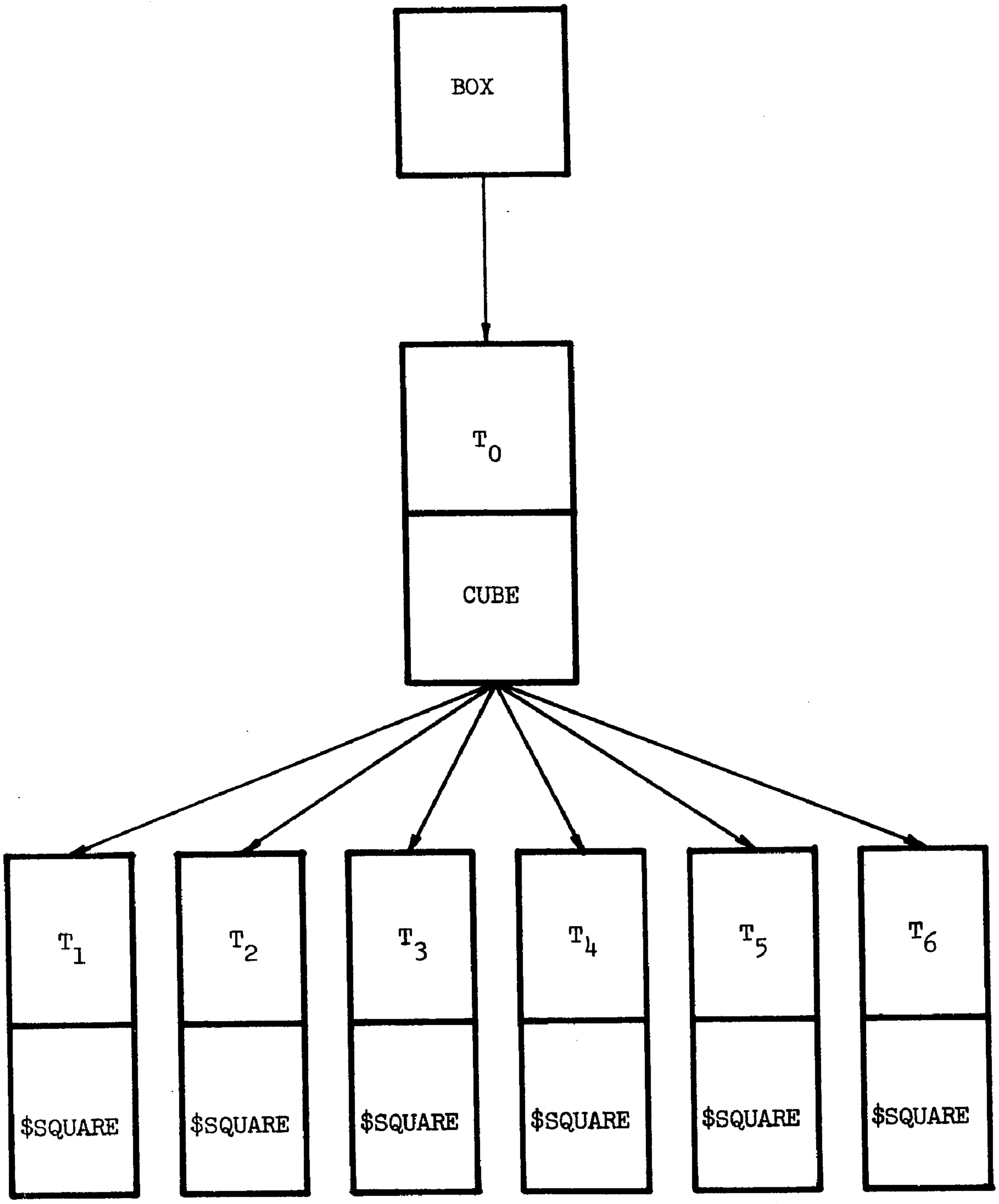
Since the resulting data structure called CUBE is really just a collection of transformed polygon matrices, we may use it as in Figure 3.5 to generate the entire equivalence class of rectangular polyhedra called BOX, by incorporating additional rotational, translational and scaling information into the transform matrix T_{ϕ}. We may thus model the arch of Figure 2.3.1 as a collection of three transformed BOXES using the data structure of Figure 3.6. In this data structure, each node contains a name and a transform matrix plus pointers to its leftmost descendent node (LSON) and the node in the structure to its immediate right (RSIB).

We may increase the power of this model to represent other features by adding dimensionality to the point representation used and increasing the size of the transform matrices used. In Figure 3.7



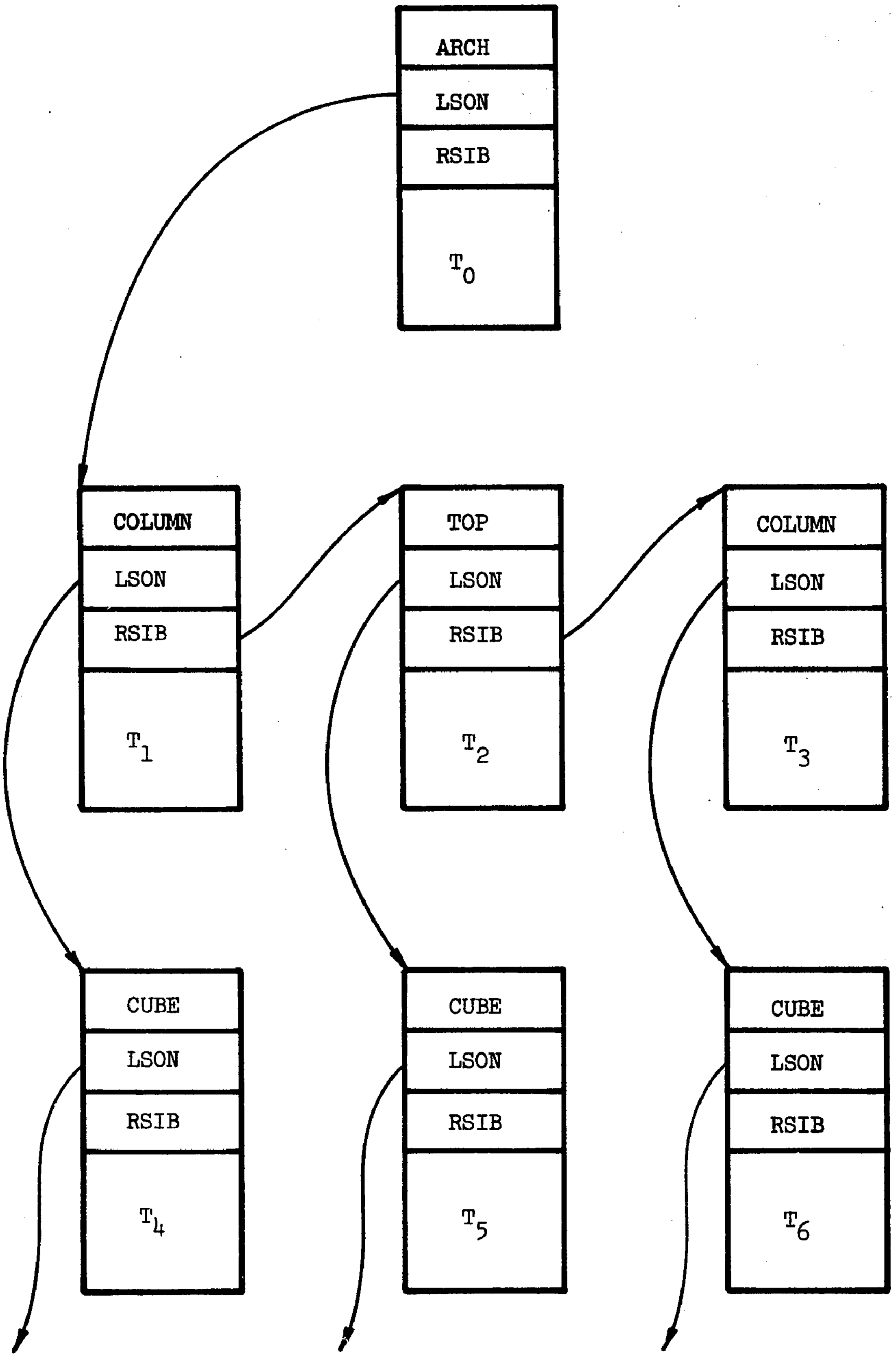
Cube Definition

Figure 3.4



Box Definition

Figure 3.5



<u>Arch Definition</u>

Figure 3.6

we have added three additional dimensions to describe the color of each point in terms of the three phosphors used to generate colored television pictures (red, green, blue). These "color coordinates" are assumed to be unity in the primitive polygons so that the color of a polygon is specified entirely by the transform matrix used. We may use the same expansion procedure to incorporate information about other features of the polygons or objects themselves, although the storage capacity required for these matrices will grow as the square of their dimensionality. It is not necessary to store each transform matrix explicitly, however, since it may be generated when necessary directly from the values of the features which it controls. This is more in keeping with the manner in which we describe objects verbally and forms the basis for a more formal definition of our concept space.

We thus define an OBJECT to be a collection of points in a multi-dimensional feature space where these points are connected into a hierarchical SHAPE structure. The dimensions of this space are the magnitudes of the features incorporated into the transform matrices of the previous model, and consist of at least the following:

<u>3.3.1</u> [SIZE, HEIGHT, WIDTH, DEPTH, XROT, YROT, ZROT, XLOC, YLOC, ZLOC, RED, GREEN, BLUE, TRANSPARENCY, TEXTURE, BRIGHTNESS, ...]

The first ten dimensions of this space (SIZE $\rightarrow$ ZLOC) affect only the physical appearance of the structural relationships implied by the SHAPE hierarchy into which they are imbedded. As may be demonstrated, they affect only the upper left four rows and columns of the transform matrix of Figure 3.7. This matrix has been partitioned in

$$\begin{bmatrix} X_1 & X_2 & X_3 \cdots & X_m \\ Y_1 & Y_2 & Y_3 \cdots & Y_m \\ Z_1 & Z_2 & Z_3 \cdots & Z_m \\ H_1 & H_2 & H_3 \cdots & H_m \\ R_1 & R_2 & R_3 \cdots & R_m \\ G_1 & G_2 & G_3 \cdots & G_m \\ B_1 & B_2 & B_3 \cdots & B_m \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \cdots & T_{17} \\ T_{21} & T_{22} \cdots & T_{27} \\ T_{31} & T_{32} \cdots & T_{37} \\ T_{41} & T_{42} \cdots & T_{37} \\ T_{51} & T_{52} \cdots & T_{57} \\ T_{61} & T_{62} \cdots & T_{67} \\ T_{71} & T_{72} \cdots & T_{77} \end{bmatrix} \cdot \begin{bmatrix} X_1 & X_2 & X_3 \cdots & X_m \\ Y_1 & Y_2 & Y_3 \cdots & Y_m \\ Z_1 & Z_2 & Z_3 \cdots & Z_m \\ 1 & 1 & 1 \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 \ldots & 1 \end{bmatrix}$$

Transformed Polygon Matrix	Transform Matrix	Original Polygon Matrix
7 x m	7 x 7	7 x m

<u>The 7-space Transformation</u>

Figure 3.7

$$\text{Transform Matrix } 7 \times 7 \quad = \left[\begin{array}{c|c|c|c} A & 0 & 0 & 0 \\ \hline 0 & B & 0 & 0 \\ \hline 0 & 0 & C & 0 \\ \hline 0 & 0 & 0 & D \end{array}\right]$$

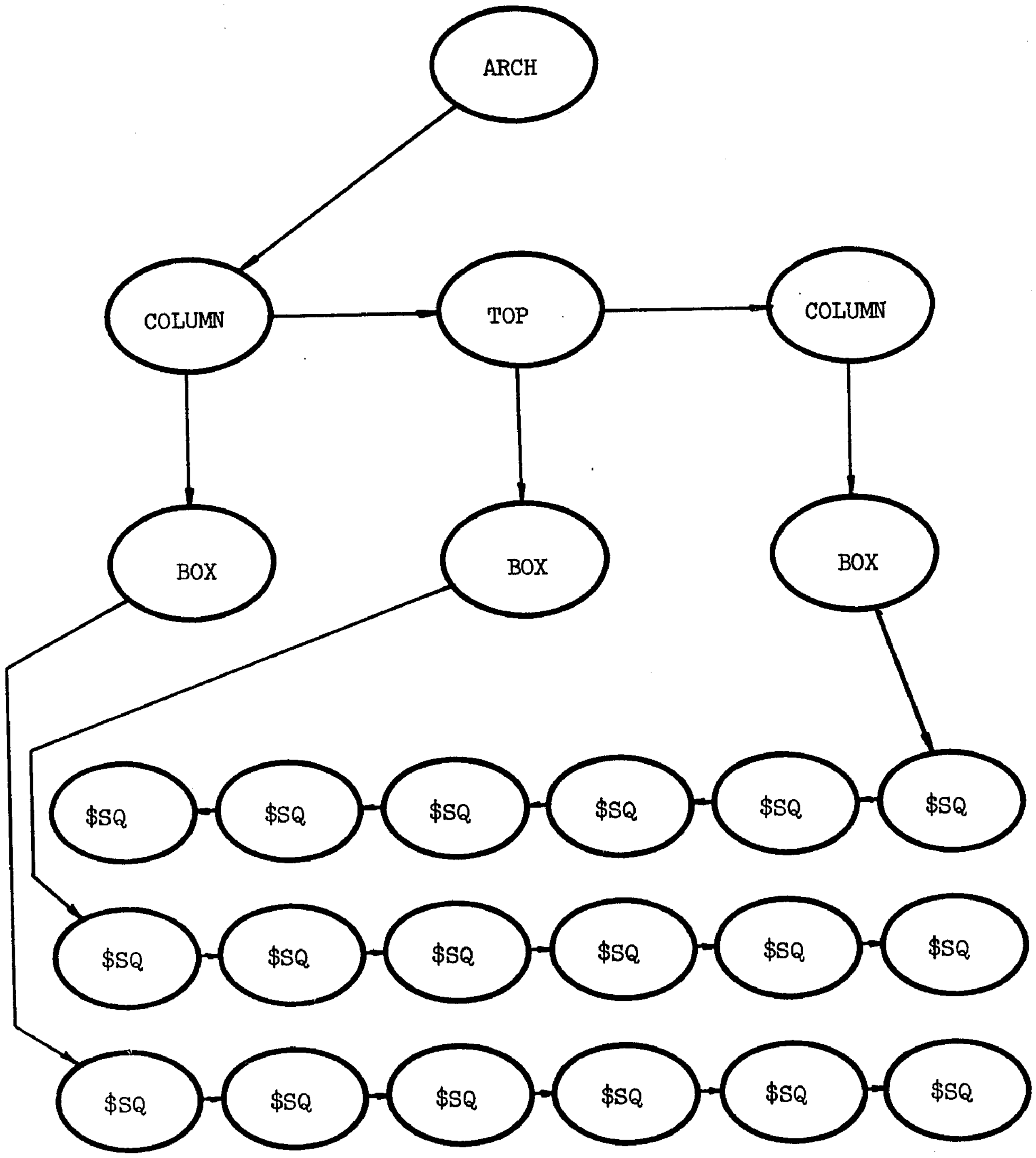
<u>Partitioned Transform Matrix</u>

Figure 3.8

Figure 3.8 to illustrate the manner in which the other dimensions of the space are incorporated into it. Submatrix A is the 4 x 4 matrix embodying the shape-affecting dimensions discussed above, matrix B incorporates BRIGHTNESS and the three color coordinates of the concept-space, and C and D represent TRANSPARENCY and TEXTURE dimensions respectively. The resulting transform matrix is a diagonal partitioned matrix, and this fact allows a considerable reduction in computational complexity. The diagonal nature of the transform matrix results from the linear independence of the concept-space dimensions chosen, and means that the transform process may be treated as four low order matrix multiplications rather than one high order multiplication.

We may now define each SHAPE node as being a point in the concept space, and the SHAPE hierarchy as being a structure of these points which allows complex OBJECTS to be conceptualized. A SHAPE hierarchy for the OBJECT named "arch" is illustrated in Figure 3.9. In this figure, each node has a name associated with it and an n-space position vector ([SIZE, ..., BRIGHTNESS]) representing its location in the concept space. The names of the terminal nodes ($SQUARE) refer to the primitive polygonal shapes defined earlier.

We postulate that a human's conceptual model of the world consists of a large number of OBJECTS connected into relational structures, as yet undefined, in a higher order relational space. Some insight into the nature of this space will be presented in Section 3.6. In order to demonstrate the utility of our model as thus far



Concept-Space Arch Definition

Figure 3.9

presented, we need to demonstrate a transformation from verbal surface structures into the model, and a transformation from the model to a visual surface structure or image.

3.4 The Verbal Transformation

We have already defined OBJECTS to be hierarchical structures of points in an n-dimensional concept space. We now note that the displacement of any of the points in a SHAPE hierarchy will have a pronounced effect on the OBJECT defined. Specifically, motion of the point named ARCH in Figure 3.9 within the space will cause a corresponding change in the conceptualization of the ARCH in its totality. Thus a TALL ARCH differs from a SHORT ARCH only by its location along the "height" axis of the space. We may use this feature of the model to our advantage by introducing MODIFIERS to move the definition points of OBJECTS about in the space. To modify an OBJECT, all that is then necessary is to add the displacement vectors of the correct series of MODIFIERS to the OBJECT position vector in question.

In English sentences, we visualize each noun phrase as defining an OBJECT or OBJECTS and the verbs and prepositions of the sentences as relating OBJECTS to other OBJECTS, or directing their appropriate modification. We see groups of adverbs and adjectives within a noun phrase as MODIFIERS which move the concept-space definition point of the noun OBJECT to create new instances of the OBJECTS. This is accomplished by adding the displacement vectors of all MODIFIERS in a noun phrase to the position vector of the noun.

Some nouns may not allow some types of modification. A "circle" when made "tall" is no longer a "circle"--it becomes an "ellipse." Other nouns may have other types of feature restrictions so that in the general case we see each noun as a disjunction of permissible regions--hypersolids--in the concept space, each with its own nominal position vector and SHAPE hierarchy. In a similar manner, MODIFIER words may be allowed to have multiple displacement vectors associated with them so that the process of parsing a sentence becomes one of choosing the appropriate meaning vector to apply. This process may be guided by the intersection of the noun's feature restrictions with the modifiers displacement vector in a classical manner.

In determining the magnitudes of the displacement vectors associated with MODIFIERS, the absolute magnitude chosen is of no importance to the verbal transformation. As long as the relative magnitudes of MODIFIERS expressing the same dimensionality are consistent with their degree of expression of that dimensionality, then any scaling induced by the choice of absolute magnitude used will be cancelled by the inverse scaling associated with the inverse verbal transformation. Thus the magnitudes of the words "tiny," "small," "large," and "huge" would be in ascending order relative to the amount of SIZE associated with each word. The absolute magnitudes of these vectors are not of importance to the visual transformation for the same reason.

In the context of this model, we see language as being a multidimensional phenomenon. Since it is reasonable to suppose that language evolved out of a need for man to communicate information about his surroundings which he perceived in a multidimensional manner, we see words and sentences as tokens which express this multidimensional meaning. It is necessary to use tokens of meaning because we do not possess any channels capable of directly transmitting smells, tastes, and images. We are thus required to convert this kind of multidimensional information into a form for which we do have an appropriate channel. Since our larynx is well suited for producing a wide variety of sounds, and since our ears are also well developed, the use of language as a channel evolved early in our past.

With this in mind, we will restate the definition of verbal communication presented in Section 2.1: Verbal communication requires (1) that the speaker conceptualize a multidimensional concept which he wishes to transmit. The speaker then performs a verbal transformation (2) on this concept to reduce it to a sequence of tokens which he can transmit over the low-dimensioned voice channel. The listener (3) on receiving this sequence of tokens performs an inverse verbal transformation (4) to reconstruct the transmitted concept. Accurate communication requires that the two transformations be nearly exact inverses of each other.

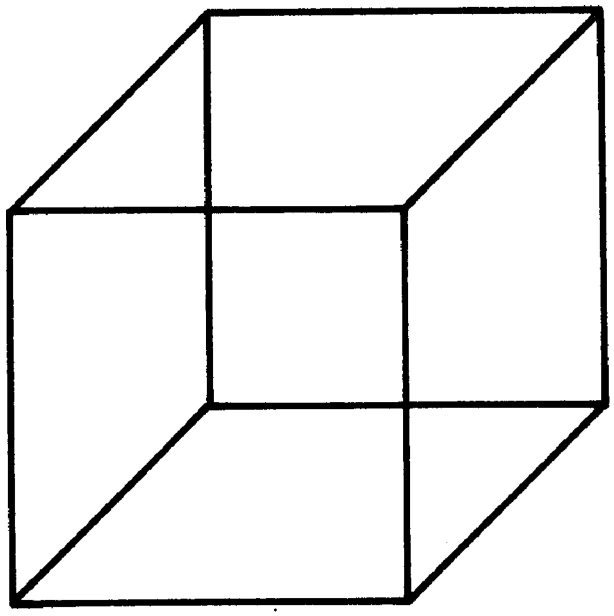
3.5 The Visual Transformation

The primary reason for basing the concept space initially on visual features is the fact that the transformation from a visual concept model to a picture requires a minimal reduction in dimensionality. This is due to the high bandwidth associated with a picture channel and allows an appraisal of the accuracy of the concepts thus modeled in a straightforward manner using computer graphic techniques. We suspect that the two-dimensional topologies associated with the linguistic and conceptual models presented earlier is due in part to the inability of researchers to conceptualize multidimensional spaces, and in part to the lack of a suitable channel for their direct inspection.

Since OBJECTS are structures of points in a 16-dimensional concept space, the transformation of these structures into a form suitable for transmission on a picture channel must involve a reduction of dimensionality. This may be accomplished trivially by neglecting all but the 10 structural dimensions of the space, and decomposing the OBJECT's SHAPE hierarchy into a linear list of polygon matrices in (x, y, z, h) using the matrix transformations presented in Section 3.3. A further dimensionality reduction could be performed by normalization of the x and y coordinates of each polygon vertex by their h coordinate

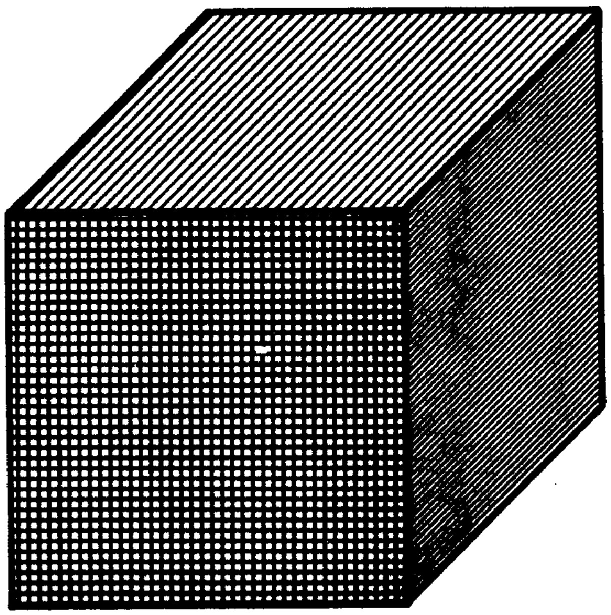
$$x' = \frac{x}{h} \qquad y' = \frac{y}{h} \tag{3.5.1}$$

and their display as a sequence of two-dimensional lines outlining each polygon as the cube in Figure 3.10. This is not a very satisfactory projection, however, because of the inclusion of normally invisible surfaces, and because of the lack of color, texture, and transparency information contained in the original model. In fact, this figure is perceived as ambiguous and the intent observer will notice his own perception of the object switching as his mind attempts



Ambiguous Cube

Figure 3.10



Non-Ambiguous Cube

Figure 3.11

to settle on one interpretation. For accurate visual communication it is necessary to perform a hidden surface removal process [9] to remove the unnecessary tokens (polygons) displayed, and to incorporate the BRIGHTNESS, RED, GREEN, BLUE, TRANSPARENCY, AND TEXTURE information inherent in the concept into the image displayed. This has been accomplished in Figure 3.11, to yield a more accurate representation of the concept. In this example, the tokens are three-dimensional (TEXTURE as a function of x, y) but are sufficient to cue the recognition of the higher dimensional visual concept. Note that it is not necessary for the exact concept model to have been transmitted--only that a concept called CUBE in the mind of the viewer have been conceptualized.

The inverse visual transformation from a flat picture (as above) to a multidimensional concept model is complicated by the lack of complete structural information in the tokens transmitted. There are, in actuality, an infinite number of real, three-dimensional objects which the picture of Figure 3.11 could be representing, and it is only our own individual "world knowledge" which associates this picture with a cube. The discipline of Scene Analysis addresses this problem, and recent efforts [23, 29] have had success using techniques borrowed from the computational linguist. We suspect that humans' use of "world knowledge" in performing this process amounts to having a number of OBJECTS conceptualized and a mechanism for comparing the visual projections of these structures with the images seen by each eye. The process of scene analysis then becomes that of moving OBJECTS about in the concept space until the image produced under visual transformation agrees with that seen by each eye. Due to the finite number of dimensions present in the concept space, adjustment of the transformed image could be carried out one dimension at a time. This remains a promising area for further research.

3.6 Conceptual Structures

A direct application of this model would be in the extension of Schank's Conceptual Structures. The nearly one-for-one correspondence between the Picture Producers and Picture Aiders of Schank's model to the OBJECTS and MODIFIERS of ours should allow their direct combination. The heart of Schank's model is the structural relationship which he posits between the PPs and the actions (ACTs) which they perform. These ACTs primarily involve time-dependent changes in the relationships between various PPs, and would correspond to time varying structures of OBJECTS within our concept space. While our model at present only allows time-independent relationships between OBJECTS to be modeled, a sequence of these relationships could represent the time continuum in the same manner as the individual frames of a movie.

We see Conceptual Structures as representing understanding at a slightly higher qualitative level than the quantitative representation used by us, but note that in order for such qualitative information to be used effectively it must be quantitatively tied to the real world. Our model provides this medium. Because all of the semantic knowledge about each OBJECT is contained in our concept-space representation, we feel that the PPs and PAs of Schank's Theory may be merged into the OBJECT of ours. Expressed in this manner, we see

Conceptual Structures as time varying structures of OBJECTS in our n-dimensional concept space. While the exact nature of these structures represents an avenue for further work, we are heartened by the fact that Schank decomposes the verbs which define these structures into a small number of biologically relevant primitive ACTs which could form a basis set for structures in a yet higher order relational space.

3.7 Semantic Networks

We may take advantage of the intuitive beauty of semantic networks as models of word meaning by noting that the concept space model of OBJECTS may be viewed as a semantic network. In a semantic network, all of the features possessed by a given noun are indicated by symbolic links to the definitions of words representing those features. The binary nature of these links precludes a quantitative representation of "shades of meaning", however, and unless all such feature words expressing the same dimensionality are ranked by their varying degrees of expression of this dimensionality, a semantic network cannot represent absolute meaning.

Our concept-space representation for OBJECTS could be construed to be a type of semantic network in which all OBJECTS are connected to tokens expressing the individual features of the space by weighted links. The weighting of these links would thus allow a continuous specification of the degree to which each OBJECT embodied each feature.

The concept space representation of OBJECTS may also be seen as a mechanism for resolving Case Relational dependencies in Case Structure analysis. Since each verb in this model may allow only certain types of "things" to stand in each Case Relation to it, this comparison may be made on the basis of an OBJECT's concept-space location. To each Case Relation of each verb, we would associate a disjunction of hyper solids in the concept space which represented the allowed types of OBJECTS for that Case Relation. Inclusion of the definition point of a given OBJECT within one of these hyper solids would be sufficient to allow the given Case Relation, and the arithmetic-nature of this test would make it easily implementable on a digital computer.

3.8 Summary

A multidimensional concept space has been developed in which conceptualizations of real world objects exist as hierarchical structures of points in the space. A transformation has been shown whereby the concept-space meanings of adjective modifiers are added to the concept-space meaning of the noun which they modify using vector addition. The resulting hierarchical structure is compatible with current Conceptual Theory in that a transformation is shown whereby Picture Producers really do produce pictures. While the class of objects which may be modeled using these techniques currently must be concrete, real, physical entities, the conceptualization of more abstract entities will hopefully follow.

4. ENGRAF

4.1 Introduction

The multidimensional concept space formalism in Chapter 3 was presented in primarily a verbal manner with some visual illustrations. Since conceptualization of these tokens required some sort of inverse transformations to occur in the mind of the reader, it is likely that the true nature of the formalism was not transmitted exactly. Workers in Artificial Intelligence have long realized the inadequacy of this information transmission medium to accurately convey complex theoretical models so that it has become standard practice to present these models in terms of programs which may run on general purpose computers. Since a computer program of a theoretical model will execute in an unambiguous manner, it is presumed that the use of a computer language to convey a model will not suffer from the ambiguity inherent in natural language descriptions. Also, the deterministic nature of computer languages provides a rigorous mathematical framework in which to prove the completeness and accuracy of a given model. To achieve a similar test of our concept space formalism, a computer program, ENGRAF, was written which implements the basic mechanisms outlined. This program consists of about 350 SPITBOL statements and runs in a 140 K byte region on an IBM 370/165. The ability of this program to "understand" English language imperative sentences was illustrated in Chapter 1. As in most natural language systems, the major mechanism in this understanding process is the parser which is responsible for performing the inverse verbal transformation from the surface structure input into the particular conceptual formalism used (e.g. deep structure, deep case structure, conceptual structure, concept-space structure).

4.2 The Parser

Existing parsers generally view the words of a surface string as operands to be manipulated as data by the parser. This view of the relationship between the parser and the strings it processes requires that a large amount of language dependent control information be written into the parser body *a priori*. In these programs, information pertaining to the meaning of individual words is usually stored in a lexicon in terms of binary symbols representing various set-membership criteria which are then used by the parser to control the parsing. Placing the majority of control responsibility within the parser itself greatly complicates the design, and may cast the parser into an inflexible mold. Further extensions of the program's understanding ability may require drastic revisions to its operation, and a program's size and understandability tend to grow inversely.

In contrast, young children seem to learn natural languages with little effort--indeed naturally-- and many people know several different languages simultaneously. It may be questionable then whether the parsing mechanisms used for human language understanding are as complicated as these existing parsers, for if they are, then it would seem that language acquisition in humans might be more difficult than it is. A more plausible mechanism would appear to be one devoid of any language-dependent control information. A parser of this nature would be a very simple mechanism, and would be forced to treat words in a surface string as *operators* which direct the parsing operation. Such a parser would derive all of its language-dependent control information from the definitions of the words themselves as stored in the lexicon.

One way to incorporate this type of control information into a word's definition is to add structure to the lexicon. In ENGRAF, this is accomplished by storing words and their meanings in a list-structured data base called a "Structured Lexicon" (SL). The SL differs from the standard lexicon in that only a small subset of the words stored in it are accessible at any given point in the parsing. This feature gives the SL the top-down parsing ability of modern language recognizers. Information is stored in the SL in an *ad hoc* aggregation of list structures defined by Relational Nodes (RELNODES), each containing a KEY, a VALUE, and an ACTION. Parsing an input sentence involves entering the sentence list structure $\boxed{S}$ and attempting to find a path to a terminal node such that the KEYS of the intermediate RELNODES define the sentence. Whenever a RELNODE is reached whose KEY can be satisfied by the leftmost words remaining in the input string, these words are removed and the subroutine calls indicated by the node's ACTION are evoked using the node's VALUE as arguments. Assuming the ACTION routines are successful in operation, parsing will continue at the leftmost descendent RELNODE to the one just qualified. If not, then the parsing fails and an appropriate error message is generated.

In order that the SL possess a greater generative power than that of the finite-state machine implied by the use of word-matching KEYS only, two additional types of KEYS are allowed. The first additional KEY type is a recursive type which causes the current state of the parsing to be saved on a push down stack, and resumed at another RELNODE in the SL. Whenever the parser reaches a terminal node in the SL, the current state of the parsing is reformatted into a concept-space PASS vector and the last parsing context is restored from the top of the stack. Parsing is successful when a terminal node in the top level list structure is reached and the input string has been reduced to null. The addition of this recursive ability to the otherwise finite-state control mechanism increases its generative power to the context-free level, and the context-sensitive nature of the list structure itself should make the SL a context sensitive formalism.

The third KEY type is introduced solely for the sake of computational efficiency. It is an associative KEY which causes the first word in the input string to be tested for a particular feature as contained in an associative data base. Thus it is not necessary for the parser to perform a linear search of all nouns, for example, at every point in the parsing when the sublist NOUN may qualify a RELNODE KEY. Instead, the associative data base may be queried to determine whether the first word in the remaining input string has the feature <noun>. If it does, then the VALUE returned by the associative search is used as the argument to the ACTION specified by the RELNODE which evoked the associative search. Entries in this associative data base are not limited to single words. Groups of words exhibiting distinct meaning may also be included in the data base. A heuristic applied to the associative search process attempts to find the longest substring beginning with the first word in the remaining input string which has the desired feature. It does this by examining multi-word links contained in the data-base entry for the first word in the substring. For example, the data base entry for the word "to" contains a multiple-word link (@ 4 PREP)

<u>4.2.1</u>   TO = @ 4 PREP @ 1 PREP (meaning of single word "to") $

which indicates to the search mechanism that a four-word "idiom" beginning with the word "to" exists within the data base which has the feature <prep>. If the search mechanism is currently processing the input string

<u>4.2.2</u>   ... TO THE LEFT OF A RED CIRCLE

for an instance of a substring with the feature <prep>, it will first perform an associative search on the four-word substring "to the left of." Since in this instance the substring has a distinct meaning, the entire substring will be removed from the input string and the meaning from its definition

<u>4.2.3</u>   TO THE LEFT OF = @ 1 PREP (meaning of four-word group) $

used as the VALUE of the RELNODE which evoked the search.

We believe that this multiple-word associative ability models human language processing, since English is full of multi-word "idioms" for which the meaning applied to the idiom itself is not linearly related to the meanings of the individual words which comprise it. Word idioms like

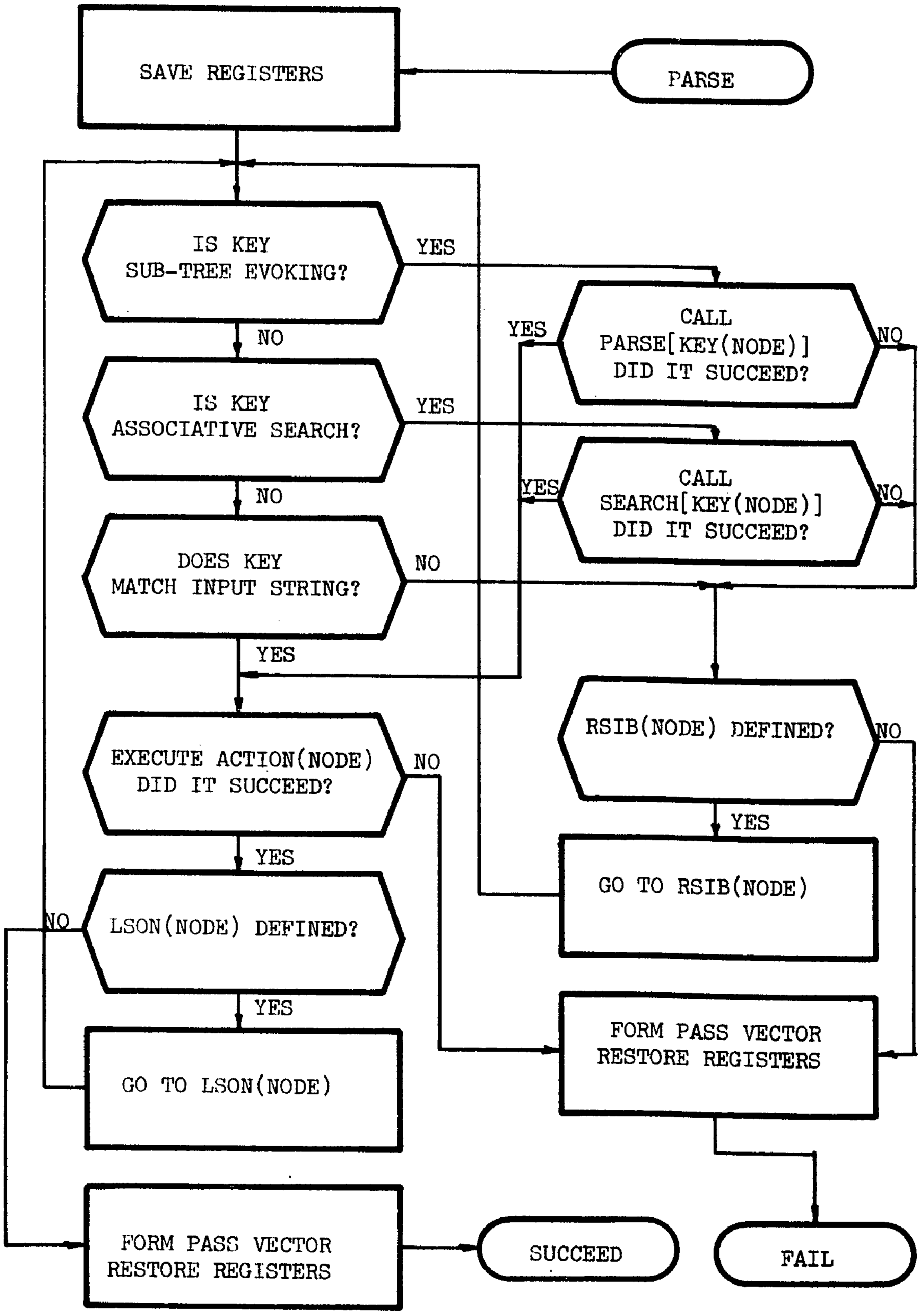
<u>4.2.3</u> Red China

<u>4.2.4</u> a teensy bit

<u>4.2.5</u> to the left of

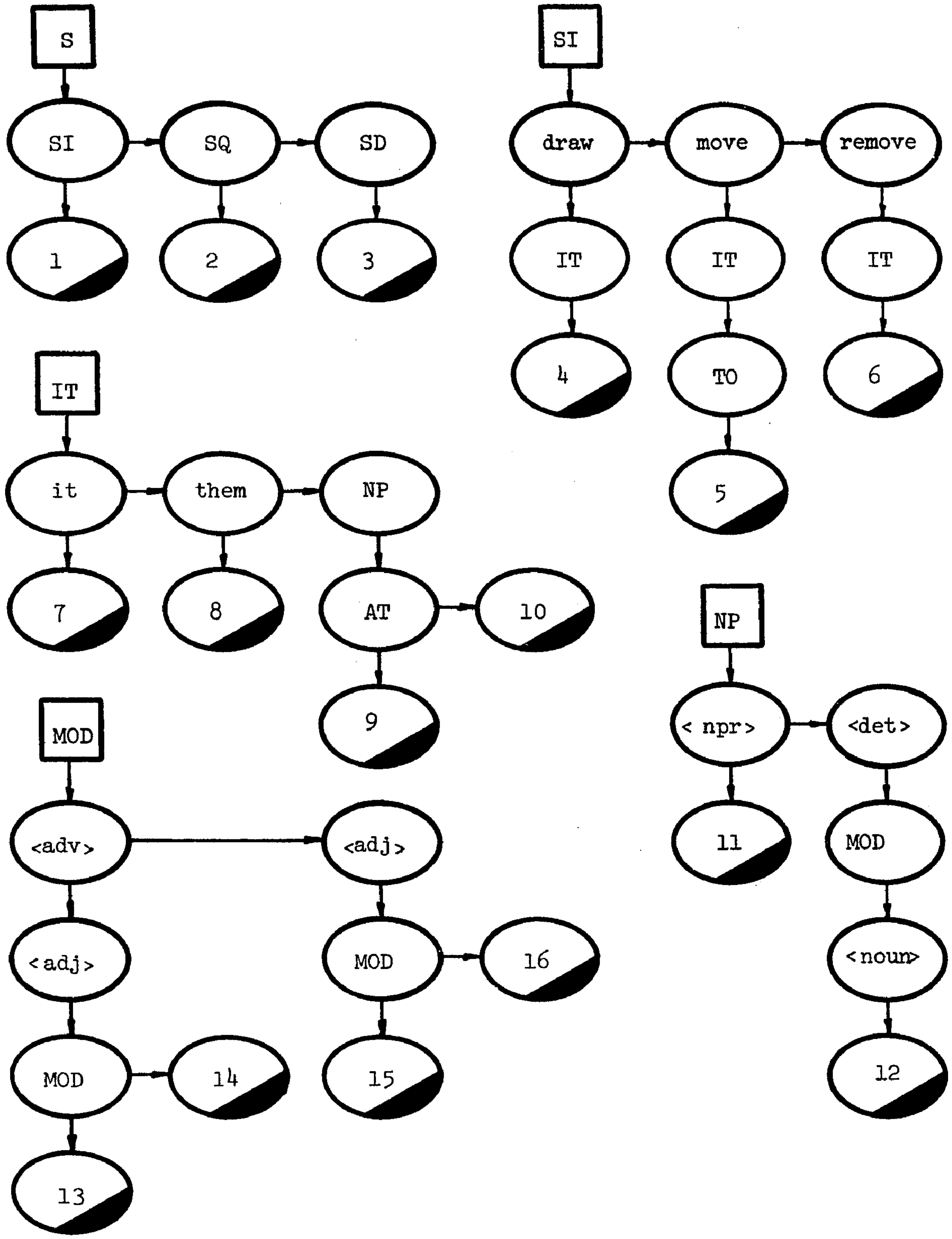
<u>4.2.6</u> far out

are better treated as morphological units rather than attempting to formalize rules which allow their aggregate meaning to be "computed" from the meanings of their constituent words. A plausible mechanism for the evolution of these idioms in natural languages would be that word groups used often in communication tend, through usage, to develop their own unique meanings. While this meaning may initially reflect the meanings of the constituent words, passage of time and changes in usage may cause the idiom meaning to diverge from its initial "computed" value. Once this occurs, it follows that often-used multiple-word idioms become contracted through normal speech slurring into single word entities which may be used again in this evolutionary cycle. From a pragmatic viewpoint, however, it is much easier to parse at this idiomatic level than at the single-word level because the associated grammar is greatly simplified. To illustrate this last point, a flow chart of the parsing algorithm is given in Figure 4.1, along with a sample structured lexicon which is able to handle a wide variety of simple imperative sentences (Figure 4.2). In the SL example, the circular elements are RELNODES


SAVE REGISTERS
PARSE
IS KEY SUB-TREE EVOKING?
YES
NO
CALL PARSE[KEY(NODE)] DID IT SUCCEED?
YES
NO
IS KEY ASSOCIATIVE SEARCH?
YES
NO
CALL SEARCH[KEY(NODE)] DID IT SUCCEED?
YES
NO
DOES KEY MATCH INPUT STRING?
NO
YES
RSIB(NODE) DEFINED?
NO
YES
EXECUTE ACTION(NODE) DID IT SUCCEED?
NO
YES
GO TO RSIB(NODE)
LSON(NODE) DEFINED?
NO
YES
FORM PASS VECTOR RESTORE REGISTERS
GO TO LSON(NODE)
FORM PASS VECTOR RESTORE REGISTERS
SUCCEED
FAIL

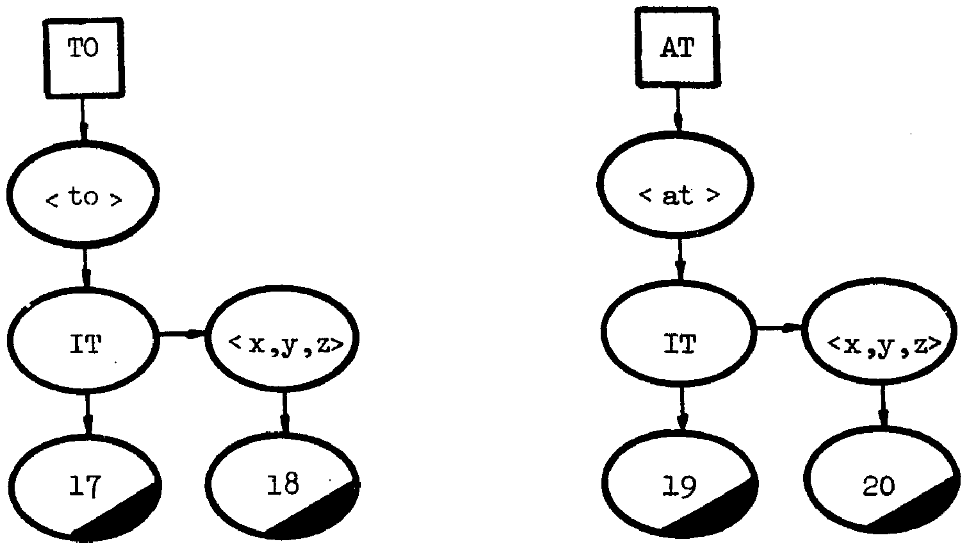
<u>Parsing Algorithm</u>

Figure 4.1



Structured Lexicon

Figure 4.2a



Structured Lexicon

Figure 4.2b

and the square elements are list-heads for the various tree-connected RELNODES. The three KEY types are indicated by:

lower case = string matching

UPPER CASE = recursive parsing at indicated list-head

<feature> = associative search for "feature"

In this example, the associative data base is not shown, nor are the VALUES and ACTIONS of the indicated RELNODES. Terminal nodes are indicated by a ◒, and are numbered to facilitate the example parsing presented in Section 4.4. While this formalism is clearly not as concise as the ATN presented in Section 2.1 due to the redundancy introduced by its tree structure, it is certainly equal in descriptive power and exhibits the positive attributes of structured programming in terms of understandability and ease of debugging.

4.3 Representation of Meaning

The VALUES stored explicitly within the RELNODES of the SL and implicitly within the associative data base are used to represent the portion of word meaning which is representable as vectors in the concept space presented in Chapter 3. The remainder of the meaning attributable to a word or class of words is represented in terms of the ACTIONS contained in the various RELNODES themselves. We see word meanings to be defined by their effect upon the conceptualization of the sentence in which they reside. Schank [2] sees a similar classification of word meaning and has defined four classes of word meanings:

a) words which produce a mental picture

b) words which modify a mental picture

c) words which perform an action

d) words which modify an action

Roughly speaking, words in types (a) and (b), above, contain most of their meaning in their VALUES within the SL, and words in types (c) and (d) embody their meanings in terms of ACTIONS within the SL. This is not a hard and fast division, however, since part of the meaning of every word is its effect on the language-independent parsing algorithm of Figure 4.1. This algorithm contains a set of 26 registers in which concept-space meanings are processed, and this register context is automatically saved on and restored from the push down stack which is used for recursive control within the SL. The context is saved in a sparse vector form for efficiency, and upon popping, a PASS vector is available to the calling RELNODE which contains the terminal state of the lower level structure. The format of this PASS vector is as follows:

<u>4.3.1</u> PASS = A*v/B*v/.../Z*v

...where the capital letters are the names of the (up to 26) registers. The * symbol may be one of a number of operators such as addition$^{(+)}$, multiplication$^{(\cdot)}$ and replacement $^{(:)}$, and each "v" represents the value associated with that vector coordinate. Registers with null values are not included in the PASS vector. The concept-space VALUES of words in the SL are in an identical format as the PASS vectors, so that not only is the value of each vector coordinate specified, but the effect which that value is to have on the register is also indicated. Meaning vectors are merged with the register context by a general purpose DECODE function which performs anomaly checking using special "must not" (-) and "may be" (+) characters which may be placed into the registers to assure number and feature agreement. Interaction of a given vector component with its associated register is governed by the truth-table presented in Table 4.1 and the concept-space meaning of each register is shown in Table 4.2. Execution of the DECODE function is one of the ACTIONS which may be evoked by a RELNODE, and its failure will result in the failure of the sub-tree under evaluation by the parser.

4.4 Example

Parsing an input sentence is accomplished by a call to the function PARSE (S, TEXT), where S is a pointer to the sentence tree, and TEXT is the input string to be processed. Operation of this parser is dependent upon the use of the multidimensional representation of meaning as presented in Chapter 3, and may be illustrated by the following example:

4.4.1 DRAW A VERY TALL RED BOX AT THE BLUE CIRCLE

Operation is initiated by a call to PARSE at the tree $\boxed{\text{S}}$. Since the first RELNODE encountered has its KEY = SI, a recursive call is made to the tree $\boxed{\text{SI}}$. Parsing in this tree removes the word "draw" from the input sentence and evokes a recursive call to the tree $\boxed{\text{IT}}$, which evokes another recursive call to the tree $\boxed{\text{NP}}$ after observing that the KEYS "it" and "them" do not match the front of the remaining string.

		MEANING VECTOR COMPONENT (M)			
		null	+	-	other
REGISTER (R)	null	R ← M	R ← M	R ← M	R ← M
	+	R ← R	R ← R	FAIL	R ← M
	-	R ← R	FAIL	R ← R	FAIL
	other	R ← R	R ← R	FAIL	R ← R * M

Merger Truth Table

Table 4.1

REGISTER	DIMENSIONALITY	REGISTER	DIMENSIONALITY
A	DEFINATENESS	K	GREENNESS
B	NUMBER	L	BLUENESS
C	WIDTH	M	INTENSITY
D	HEIGHT	P	X ROTATION
E	DEPTH	Q	Y ROTATION
F	SIZE	R	Z ROTATION
G	SHAPE	X	X LOCATION
H	TRANSPARENCY	Y	Y LOCATION
I	TEXTURE	Z	Z LOCATION
J	REDNESS		

Register Dimensionality

Table 4.2

Parsing in the NP tree checks the word "a" for the feature <npr> which fails, and then for the feature <det> which succeeds. The meaning vector stored for the string "a" in the associative data base is:

<u>4.4.2</u> @1DET.A:-/B:-/$ (":" implies replacement)

which indicates that a one-word string starting with "a" has the feature <det>, and that its concept-space "meaning" is indefinite singular (A:-/B:-/). This meaning vector is used as the argument to the function DECODE (the specified ACTION for that node), which inserts (-) in the "definiteness" register (A) and the "number" register (B). Thus the state of the registers at this point is:

<u>4.4.3</u>	A = -	"definiteness"
	B = -	"number"
	all others = null	

Parsing continues with a recursive call to parse sub-tree $\boxed{\text{MOD}}$. In this sub-tree, the word "very" is noted to be an <adv> and the ACTION specified by that node is to save its meaning vector in a temporary storage location for further use. Subsequently the <adj> "tall" is removed from the input string and its ACTION specifies that its meaning vector be DECODED onto the MOD registers to yield:

<u>4.4.4</u> D = 1 "height"

Another operation specified by the ACTION causes the saved meaning vector of the <adv> to be DECODED. This vector specifies a multiplication to yield:

<u>4.4.5</u> D = 2 "height"

Another recursive call to parse sub-tree $\boxed{\text{MOD}}$ causes the word "red" to be parsed yielding:

4.4.6 J = 1 "redness"

The subsequent call to parse sub-tree $\boxed{\text{MOD}}$ fails since the next word in the sentence (box) does not have the feature <adj> or <adv>. This causes an exit from the previous call to parse $\boxed{\text{MOD}}$ at terminal node 16 which reformats the register J into a PASS vector.

4.4.7 PASS = J + 1/ ("+" implies addition)

A successful return from this sub-tree results in the PASS vector being decoded onto the $\boxed{\text{MOD}}$ registers at the calling level to yield:

4.4.8	D = 2	"height"
	J = 1	"redness"

A successful return from this sub-tree is initiated at terminal node 15 with the PASS vector reflecting both meanings.

4.4.9 PASS = D+2/J+1/

This return causes the $\textcircled{\text{MOD}}$ node in the $\boxed{\text{NP}}$ sub-tree to succeed and so the PASS vector is decoded onto its registers to yield:

4.4.10	A = -	"definiteness"
	B = -	"number"
	D = 2	"height"
	J = 1	"redness"

At this point, the <n> "box" is parsed and its meaning vector decoded onto the $\boxed{\text{NP}}$ registers to yield:

4.4.11	A = -	"definiteness"
	B = -	"number"
	D = 2	"height"
	G = $BOX	"shape"
	J = 1	"redness"

Had the noun "boxes" been used incorrectly in the sentence,

4.4.12 *DRAW A VERY TALL RED BOXES ...

the DECODE function would have failed since "boxes" has a plural "number" feature (A:+), and (+) and (-) flags may not be merged (Table 4.3.1). In a similar manner, the sentence

4.4.13 *DRAW A VERY TALL RED CIRCLE ...

would fail, since the noun "circle" contains "must not" flags in the "height," "width," and "depth" dimensions.

Returning to our example, the successful traversal of the $\boxed{\text{NP}}$ sub-tree causes a successful termination at terminal node 12 with the PASS vector returned as follows:

4.4.14 PASS = A+-/B+-/D+2/G:$BOX/J+1/

Parsing then resumes at the (NP) node in the $\boxed{\text{IT}}$ sub-tree with the ACTION of that node causing the PASS vector to be DECODED onto the $\boxed{\text{IT}}$ registers. Movement to the (AT) node subsequently evokes a recursion to the $\boxed{\text{AT}}$ sub-tree which causes the word "at" to be removed from the input string and a recursive call made to the $\boxed{\text{NP}}$ sub-tree. Successful traversal of this sub-tree returns a PASS vector conceptualizing "the blue circle." The ACTION specified by the (NP) node in the $\boxed{\text{AT}}$ sub-tree causes this PASS vector

4.4.15 PASS = A++/B+-/G:$CIRCLE/L+1/

to be DECODED, and the OBJECTS existing in the current time frame searched to find the location of "the blue circle." In this search process, the non-null dimensions contained in the $\boxed{\text{AT}}$ registers are compared with each OBJECT in the current time frame, and a list is compiled of all OBJECTS having the "shape" $CIRCLE, and "blueness" greater than or equal to 1. Thus a "very blue circle," having "blueness" equal to 2, would also be located by this search if it existed. After a list of candidate OBJECTS has been compiled from the current time frame, a check is made to determine if the number found is in agreement with the "number" and "definiteness" values contained in the $\boxed{\text{AT}}$ registers. If the "definiteness" register contains an indefinite flag (-), then agreement is confirmed as long as the number of located objects is non-zero. If a definite flag is found (+), however, the number of OBJECTS found must agree exactly with the contents of the "B" register.

In our example, let us assume that only one "blue circle" was located in the current time frame. This OBJECT is then inspected to determine its x, y, and z concept-space coordinates which are returned in the PASS vector upon popping from the $\boxed{\text{AT}}$ subtree. Assuming that the "blue circle" was located at (4, 5, 6), the PASS vector would be:

4.4.16 PASS = X+4/Y+5/Z+6

The ACTION specified by the $\textcircled{\text{AT}}$ node in the $\boxed{\text{IT}}$ sub-tree causes this PASS vector to be DECODED onto its registers, whereupon the $\boxed{\text{IT}}$ sub-tree terminates successfully. The resulting PASS vector returned to the $\textcircled{\text{IT}}$ node in the $\boxed{\text{SI}}$ sub-tree represents the concept-space conceptualization of

4.4.17 A VERY TALL RED BOX AT 4, 5, 6

in vector form.

4.4.18 PASS = A+-/B+-/D+2/G:$BOX/J+1/X+4/Y+5/Z+6

The ACTION taken by the $\textcircled{\text{IT}}$ node is to create a SPITBOL program which, when executed, will cause the current time frame to be updated to include the new addition of this OBJECT. This program is ultimately executed by the $\textcircled{\text{SI}}$ node in the $\boxed{\text{S}}$ sub-tree and becomes a permanent part of the SL database. A similar program is also created by the $\textcircled{\text{IT}}$ node which will cause the current time frame to be restored to its previous state when executed. The program thus maintains a series of programs to move the current time frame forward in time, and a series of programs to move it back in time. Use of this time line capability allows incorrect program responses to be corrected, and animation sequences to be constructed. There is currently no facility provided for the insertion of additional time frames between those already created, although this extension would be straightforward.

In the above example, it should be noted that the entire meaning of the word "draw" was contained in the ACTIONS of the nodes which succeeded its node in the $\boxed{\text{SI}}$ sub-tree. Thus the only function of the word was to select the correct branch of the $\boxed{\text{SI}}$ sub-tree to process. Had a network topology been adopted for the SL instead of its hierarchical topology, this would not have been possible.

4.5 Display

The function of the parser in ENGRAF is to use English language commands input by the operator to maintain a current time frame containing the concept-space conceptualizations of the OBJECTS which the operator has created. This portion of the system is written in SPITBOL and executes under TSO on an IBM 370/165. The function of the display subsystem is to use the contents of this current time frame to produce shaded color television pictures of the OBJECTS contained in it. While this subsystem is not operational in the current version of ENGRAF, we do possess an operating color graphics facility based upon an Adage AGT-30 and Varian 620. This system [9, 10, 11] performs all of the basic operations required of ENGRAF's display subsystem, but for reasons of telecommunication complexity has not been interfaced to the IBM-based SL parser.

The existing color display system uses a set of function switches to evoke the various ACTIONS of which it is capable, and a set of six, analog input dials for interactive specification of the VALUES of the dimensions.

4.5.1 SIZE, XLOC, YLOC, ZLOC, XROT, YROT

in the limited concept-space of this system. The system also uses keyboard input to specify the VALUES of the dimensions:

4.5.2 RED, GREEN, BLUE

Extension of the current graphics system to encompass all of the concept-space dimensions presented in Chapter 3 would be straightforward.

5. EXTENSIONS

5.1 Questions

Most natural language understanding systems demonstrate their degree of language competence by answering questions about facts stored in their data bases. In these systems, declarative sentences are often input first, and a data base of facts constructed using the information contained in these sentences. Systems may be characterized by the nature of their data bases as being "text based" or "encoded" systems. Text based systems store all information in surface structure format and attempt to answer questions about this information by retrieving the appropriate responses directly from the data base. Systems which use some formalism--different from text--for information storage are called "encoded" systems because of the transformation required to convert surface strings into this formalism. Encoded systems also require an inverse transformation to return encoded information to surface string format, although most systems do not fully implement this feature resulting in somewhat stilted conversations. In this regard, ENGRAF is no exception. The inverse transformation from our conceptual model in which information is stored is complicated by the fact that all surface string words are removed by the encoding process and thus cannot be used to guide the decoding process. This decoding process has not been implemented, although we will discuss a possible implementation later.

ENGRAF differs from most existing natural language systems in that its degree of language competence is demonstrated visually instead of verbally. Thus question answering ability is not required for effective communication with the system, although we will discuss its implementation anyway. Questions may be roughly divided into two types depending upon the degree of response which is required. The simplest type of questions to answer are those which require only a yes/no reply such as

<u>5.1.1</u> IS THE BLUE BOX AT 1,2,3 LARGER THAN THE RED CIRCLE?

and

<u>5.1.2</u> CAN YOU DRAW A TALL GREEN CIRCLE?

More difficult questions to answer may require answering the questions "why?", "what?", "where?", "when?", "why?", and "how?". These are typically called "wh-" questions because of their common letters and may be typified by the following:

<u>5.1.3</u> WHERE IS THE BLUE BOX?

<u>5.1.4</u> HOW TALL IS THE BOX AT 1,2,3?

<u>5.1.5</u> WHAT IS ABOVE THE GREEN CIRCLE?

<u>5.1.6</u> WHY DID YOU DRAW THAT?

The existence of distinct syntactic structures for interrogative sentences allows their easy discrimination from imperative and declarative constructions. Using ENGRAF's current implementation, the addition of an SQ sub-tree in the SL would suffice, and all of the lower level sub-trees from the SI sub-tree could be used without modification since they return their results as concept-space PASS vectors. Question 5.1.1 would be answered by comparing the SIZE dimensions of the two specified OBJECTS and returning either a "yes" or "no" response. Questions of this type have already been implemented on earlier versions of ENGRAF which did not utilize the SL, and would be straightforward with it. Questions of the type of 5.1.2 which require the program to evaluate its own processing abilities may be easily answered without storing this information explicitly. All that would be required would be to evoke the $\boxed{\text{SI}}$ sub-tree after "can you" had been parsed by the $\boxed{\text{SQ}}$ sub-tree. If this operation succeeded then the answer would be "yes" and if not, "no." In this example, the answer would be "no," since the NP sub-tree would find the "tall...circle" anomalous.

The wh- questions 5.1.3 and 5.1.4 both require that the appropriate concept-space coordinate of the indicated OBJECT be returned along with its units and dimensionality.

<u>5.1.7</u> IT IS AT (4,5,6)

<u>5.1.8</u> 2.35 METERS TALL

Question 5.1.5 may also be answered by outputting a list of the OBJECTS in the current time frame which are located above the location defined for the "green circle"--assuming that it exists. For an understandable response, however, the system must be able

to do better than to enumerate by rote the concept-space coordinates of all such OBJECTS. This requires that an inverse verbal transformation be implemented. This could be accomplished by linking all word meaning vectors stored in the SL into sorted lists by their dimensionality. Thus faced with an OBJECT which contained HEIGHT, $BOX, REDNESS, and BLUENESS dimensionality we might determine by examining these lists that the words "short purple box," if parsed, would result in a similar conceptualization.

Response to questions like 5.1.6 would require examination of the motivation behind each action which the program undertook. In the current ENGRAF, the only motivation is "normative" so that the response to all such sentences would be:

5.1.9 BECAUSE YOU TOLD ME TO

In a more realistic system in which operator commands initiated sequences of actions such as in animation sequences, more involved responses could be envisioned.

5.2 Declaratives

Declarative sentences provide a means for increasing the size of the "world" in which a program is capable of interacting. A good natural language system should be able to use information phrased in declarative sentences to either expand its descriptive capabilities or allow more subtlety in expression. The current ENGRAF accepts a limited subset of declarative constructions which are used to define new word meanings to the system. In the sense that these new word meanings must be formalized within the dimensionality limits of the existing system (16), this ability affects the subtlety of expression to which ENGRAF is capable of responding. Were it possible for declarative sentences to cause the addition of other dimensions to the concept space, then this ability would expand the descriptive capabilities of the system.

In order to extend the program to understand generalized declarative constructions like

5.2.1 JOHN DREW A TALL RED BOX

the program would have to have a means of conceptualizing "John," and somehow attaching to that conceptualization that it "drew" an OBJECT called "a tall red box." While we could undoubtedly conceptualize "John" as an OBJECT, it remains unclear exactly how to conceptualize quantitatively the act of "drawing." As a first approximation, it might be useful to maintain separate "time lines" for every active OBJECT in the system's memory. This would allow conceptualizations involving time to be modeled using Conceptual Structures functionally identical to Schank's. It appears reasonable to assume that verb meaning may be modeled in much the same manner as our OBJECTS, perhaps using dimensionality borrowed from Case Structures (TENSE, ASPECT, FORM, MOOD, ESSENCE, MODAL, MANNER, TIME); and that the analog of our primitive SHAPES would be the primitive ACTS of Schank's theory. Such a data structure would be very large, to be sure, but considering that our brains contain roughly 10^{10} cells, a 10^{10} bit data structure would not be unreasonable.

5.3 Non-Visual Dimensions

An easy extension to the current program would involve the addition of other, non-visual dimensions to the existing concept space. By adding taste dimensions [SWEET, SOUR, BITTER, SALTY] and smell dimensions [FLOWERY, FRUITY, PUTRID, SPICY, BURNED, RESINOUS], it would be possible to conceptualize

5.3.1 A HAM SANDWICH

and

5.3.2 A CHOCOLATE ICE CREAM CONE

more realistically. We would be faced, however, with the problem of verifying the accuracy of these conceptualizations without an effective channel for communicating their new dimensionality. Thus we would be forced to rely upon verbal comparisons between OBJECT's flavors and upon the structural consistency imposed by the visual portions of the conceptual space model.

One dimension for which an I/O channel exists is TEMPERATURE. Addition of this dimension would add new meaning to

5.3.3 A HOT PASTRAMI AND CHEESE SANDWICH ON RYE

and

5.3.4 HOTTER THAN HELL

and it could easily be incorporated into the concept space. This dimension is generally associated with our sense of touch which is also sensitive to PRESSURE, in varying degrees. This too could be added.

All of these new dimensions would be added to ENGRAF's concept space by the creation of new registers in the parser. Since SPITBOL allows creation of new variables at execution time, this would be accomplished automatically by adding new words expressing the added dimensionality to the SL's associative data base. The only other changes would affect the patterns used to generate output statements, and these could be modified at execution time also by using the SPITBOL CODE( ) function.

5.4 Speech Recognition

Recent efforts in the area of speech recognition have had considerable success using the predictive parsing techniques of the Augmented Transition Network. Walker [30] made use of Winograd's PROGRAMMER [4] in a recent speech understanding system which was subsequently modified to allow "best first" parsing instead of the original "depth first" operation. In this technique, a multiprocessing approach is used in which several possible parsings of an utterance may be processed simultaneously, with only the "best" parsing processed at any given point. This "best" parsing is determined by uncertainty coefficients compiled for each possible parsing.

Another technique, employed by Miller [31], uses an ATN grammar which is configured so that parsing may start in the middle of a string at an "island of reliability" and proceed both rightward and leftward. Several such "islands" may be processed simultaneously as a threshold is lowered (tide going out) until a complete parsing is achieved. This technique requires that subnetworks in the grammar contain references to the contexts which may evoke them.

Application of the Structured Lexicon to speech understanding could result in the elimination of the associative KEY type. This KEY type was introduced to eliminate the tedious linear searching required for matching word length substrings exhibiting common syntactic features. By reducing the pattern matching process to the phoneme level, the number of searches at each point in the parsing would be reduced to about 10. Thus, rather than performing an associative search for "nounness," we would evoke parsing in a $\boxed{\text{NOUN}}$ sub-tree in which all nouns had been stored phoneme by phoneme. The meaning attached to each word would be stored in its terminal node, and parsing could proceed in a "best first" manner based upon the phoneme probability distribution derived from the phoneme recognizer. Again, the system would be large, but probably less than the 10^{10} bit "reasonableness" bound introduced earlier. The addition of context information to $\boxed{}$ nodes would also allow use of Miller's technique.

5.5 Conclusion

A quantitative, mathematical formalism has been introduced which utilizes the linear independence of features normally used to describe real world objects to model or conceptualize these OBJECTS in a multi-dimensional space. In this formalism, OBJECTS--resembling PP's of Conceptual Theory--are defined as structures of points in the concept space which are based upon primitive SHAPES. These structures ultimately define the exterior surfaces of real world objects and the significant features which they possess in terms of our own physical sensors. The formalism has been partially implemented as a computer graphics command language, ENGRAF, which uses a "Structured Lexicon" parser to form n-space conceptualizations from English language input. In this parser, the basic control mechanism is completely language-independent, and all language-dependent information is stored in an _ad hoc_ collection of list structures which control the basic mechanism.

The concept-space formalism has been shown to be compatible with currently accepted linguistic theories, as has the parsing technique employed. Applications of these techniques have been shown in the areas of computer graphics, scene analysis, concept modeling, and speech recognition. It is hoped that the introduction of these quantitative techniques into an otherwise qualitative discipline will provide added insight into our own psychological processes.

6. LIST OF REFERENCES

1. Kaplan, R. M. 1972. Augmented transition networks as psychological models of sentence comprehension. Artificial Intelligence, 3:77-100.

2. Schank, R. 1973. Identification of conceptualizations underlying natural language. <u>In</u> R. Schank and K. M. Colby (Eds.), Computer Models of Thought and Language. San Francisco: Freeman, 187-248.

3. Simmons, R. F. 1973. Semantic networks: their computation and use for understanding English sentences. <u>In</u> R. Schank and K. M. Colby (Eds.), Computer Models of Thought and Language. San Francisco: Freeman, 63-113.

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

5. Woods, W. A. October, 1970. Transition network grammars for natural language analysis, CACM, 13,10:591-606.

6. Bobrow, D. G. 1964. Natural language input for a computer problem solving system. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

7. Dewar, R. B. K. 1971. SPITBOL, Illinois Institute of Technology, Chicago, Illinois, S4D23.

8. Griswold, R. E., J. F. Poage, and R. P. Polonsky. 1971. The SNOBOL 4 Programming Language (2nd ed.). New Jersey: Prentice-Hall.

9. Eastman, J. F. 1974. An efficient scan-conversion and hidden surface algorithm. To be published in Computers and Graphics.

10. Eastman, J. F. and J. Staudhammer. January, 1974. Computer display of colored, three-dimensional objects. Proc. IEEE Second Annual Symposium on Computer Architecture, Houston, Texas.

11. Staudhammer, J. and D. J. Ogden. 1974. Computer graphics for half-tone three-dimensional object images. To be published in Computers and Graphics.

12. Bobrow, D. G. and B. Raphael. September, 1974. New programming languages for artificial intelligence research. ACM Computing Surveys, 6,3:155-174.

13. Parke, F. I. June, 1972. Computer generated animation of faces. UTEC-CSc-72-120, Computer Science Department, University of Utah, Salt Lake City, Utah.

14. Shapiro, L. G. March, 1973. Artist: an experiment in picture creation using natural language input. Technical Report 73-02, Department of Computer Science, University of Iowa, Iowa City, Iowa.

15. Chomsky, A. N. 1957. Syntactic Structures. The Hague: Mouton and Co.

16. Booth, T. L. 1968. Sequential Machines and Automata Theory. New York: John Wiley and Sons, Inc.

17. Hewitt, C. 1969. PLANNER: A language for proving theorems in robots. Proc. of the International Joint Conference on Artificial Intelligence, Bedford, Massachusetts, 295-301.

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

19. Quillian, M. R. 1969. The teachable language comprehender: a simulation program and theory of language. Communications of the ACM, 12,8:459-476.

20. Schank, R. C. 1974. Understanding paragraphs. Technical Report No. 5, Institute for Semantic and Cognitive Studies, Castagnola, Switzerland.

21. Bruce, B. and C. F. Schmidt. July, 1974. Episode understanding and belief guided parsing. Unpublished paper.

22. Weissman, S. J. December, 1973. On the implementation of an information system for semantic modeling and computer understanding. Report R-635, Coordinated Science Laboratory, University of Illinois, Urbana, Illinois.

23. Winston, P. H. 1972. The MIT robot. Machine Intelligence 7. New York: John Wiley and Sons, Inc., 431-463.

24. Lancaster, P. 1969. Theory of Matrices. New York: Academic Press.

25. Frijda, N. H. 1972. Recognition of emotion. In L. Berkowitz (Ed.), Advances in Experimental Social Psychology, 167-223.

26. Mozell, M. M. 1971. The chemical senses. In J. W. Kling and L. A. Riggs (Eds.), Experimental Psychology (3rd ed.). New York: Holt, Rinehart and Winston, Inc., 193-222.

27. Psychology Today: An Introduction. 1972. CRM Books, Del Mar, California.

28. Newman, W. M. and R. F. Sproull. 1973. Principles of Interactive Computer Graphics. New York: McGraw-Hill, 467-480.

29. Clowes, M. B. 1971. On seeing things. Artificial Intelligence, 2:79-116.

30. Walker, D. E. 1973. Speech understanding, computational linguistics, and artificial intelligence. Technical Note 85, Stanford Research Institute, Menlo Park, California.

31. Miller, P. L. November, 1974. A locally organized parser for spoken input. CACM, 17,11:621-630.