AD/A-005 040

COMPUTATIONAL UNDERSTANDING

ANALYSIS OF SENTENCES AND CONTEXT

STANFORD UNIVERSITY

PREPARED FOR

ADVANCED RESEARCH PROJECTS AGENCY

NATIONAL INSTITUTE OF MENTAL HEALTH

MAY 1974

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER STAN-CS-74-437	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD/A-005040
4. TITLE (and Subtitle) COMPUTATIONAL UNDERSTANDING: ANALYSIS OF SENTENCES AND CONTEXT.		5. TYPE OF REPORT & PERIOD COVERED technical, May, 1974
		6. PERFORMING ORG. REPORT NUMBER STAN-CS-74-437
7. AUTHOR(s) Christopher Kevin Riesbeck		8. CONTRACT OR GRANT NUMBER(s) DAHC 15-73-C-0435 MH 06645-13
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University Computer Science Department Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS ARPA/IPT, Attn: S. D. Crocker 1400 Wilson Blvd., Arlington, Va. 22209		12. REPORT DATE May, 1974
		13. NUMBER OF PAGES 250
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ONR Representative: Philip Surra Durand Aeronautics Bldg., Rm. 165 Stanford University Stanford, California 94305		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Releasable without limitations on dissemination.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

PRICES SUBJECT TO CHANGE

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The goal of this thesis was to develop a system for the computer analysis of written natural language texts that could also serve as a theory of human comprehension of natural language. Therefore the construction of this system was guided by four basic assumptions about natural language comprehension. First, the primary goal of comprehension is always to find meanings as soon as possible. Other tasks, such as discovering the syntactic relationships, are performed only when essential to decisions about meaning. Second, an attempt is made to understand each word as soon as it is read, (continued)

DD FORM 1 JAN 73 1473

Reproduced by NATIONAL TECHNICAL INFORMATION SERVICE U S Department of Commerce Springfield VA 22151

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO-AIM238

MAY 1974

COMPUTER SCIENCE DEPARTMENT
REPORT NO. STAN-CS-74-437

COMPUTATIONAL UNDERSTANDING: ANALYSIS OF SENTENCES AND CONTEXT

Christopher Kevin Riesbeck

Abstract:

The goal of this thesis was to develop a system for the computer analysis of written natural language texts that could also serve as a theory of human comprehension of natural language. Therefore the construction of this system was guided by four basic assumptions about natural language comprehension. First, the primary goal of comprehension is always to find meanings as soon as possible. Other tasks, such as discovering syntactic relationships, are performed only when essential to decisions about meaning. Second, an attempt is made to understand each word as soon as it is read, to decide what it means and how it relates to the rest of the text. Third, comprehension means not only understanding what has been seen but also predicting what is likely to be seen next. Fourth, the words of a text provide the cues for finding the information necessary for comprehending that text.

A dissertation submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract DAHC15-73-C-0435 and by the National Institute of Mental Health Contract MH06645-13.

ACKNOWLEDGMENTS

While taking full responsibility for the failings of this thesis, I would like to thank those who valiantly tried to save me from my erring ways.

Most of all I would like to thank Professor Roger C. Schank under and with whom I have worked for four years. Both his approach to computational linguistics in general and his own work in particular form the basis for the work described here. And any patches of clarity in the prose that follows is the result of his persistence.

Gratitude is also owed to the other two readers of this thesis, Professors Jerome Feldman and Cordell Green. Their comments on the first part of this thesis led to the work described in the second part. Professors Kenneth Colby and Eve Clark also provided many comments and criticisms of the first part of this thesis.

I'm grateful to Neil Goldman, Paul Martin, Charles J. Rieger III, and Dr. Manfred Wettler for many discussions about memory processes and language comprehension. I am also grateful to Dr. Wettler and the Istituto per gli studi semantici e cognitivi for the opportunity they gave me to work on computational understanding. And I'd like to thank Kathleen Zeisler who transformed an ungodly mass of scribblings into legible copy.

Unfortunately I must report that my work in the last stages was severely hampered by the distracting presence of one Suzanne Barrett. It is fortunate for her that I am not one to hold grudges

TABLE OF CONTENTS

ACKNOWLEDGMENTS		iii
TABLE OF CONTENTS		iv
CHAPTER 1	INTRODUCTION	
1.1	What It's About	1
1.2	Computational Understanding	3
1.3	Aspects of this System	5
1.4	What This System Does	10
	PART I	
	GUIDE TO PART I	17
CHAPTER 2	SENTENTIAL ANALYSIS	
2.1	Introduction to Sentential Analysis	19
2.2	History of the Analyzer	22
2.3	The Program Itself	27
2.4	Overview of the Analysis Process	29
2.5	Sense Versus Meaning	33
CHAPTER 3	PREVIOUS WORK	35
CHAPTER 4	BASIC IDEAS IN THE ANALYZER	
4.1	Goals and Assumptions	47
4.2	Pattern Matching	52
4.3	Ambiguity	54
CHAPTER 5	CONCEPTUAL DEPENDENCY	
5.1	General Aspects	56
5.2	CD Elements	59
5.3	CD Structures	63

CHAPTER 6 SENTENTIAL ANALYSIS MECHANISMS

6.1 Overview of Expectations and Actions 79
6.2 Analyzing with Expectations 82
6.3 Features and Expectations 83
6.4 Functions in the Analyzer 84
6.5 Conceptual Semantics 92
6.6 The Monitor 94

CHAPTER 7 EXAMPLES 98

CHAPTER 8 MULTI-SENTENCE ANALYSIS

8.1 Introduction 130
8.2 Modifications to the Analyzer 132
8.3 Examples of Text Analysis 135

CHAPTER 9 REVIEW OF THE ANALYZER 146

PART II

GUIDE TO PART II 151

CHAPTER 10 INTRODUCTION TO EXTENDED ANALYSIS 153

CHAPTER 11 CONTEXT

11.1 The Word "Context" 156
11.2 Contextual Effects 158
11.3 The Context Cluster 161

CHAPTER 12 EXTENDED ANALYSIS MECHANISMS

12.1 Overview of Extended Analysis 166
12.2 A Flow Table for Extended Analysis 171
12.3 Needs and Expectations 176
12.4 Internal Versus External Needs 179
12.5 Traps Versus Needs 182
12.6 The Focus of a Test 184
12.7 Forms for Needs and Requests 186
12.8 Example Definition Using Needs 189

CHAPTER 13 THE EFFECT OF KNOWLEDGE ON ANALYSIS

13.1 Texts and Needs 193
13.2 Texts and Stories 195
13.3 Patterns and Predictions 202
13.4 Conceptual Requests to Language Requests 203
13.5 Preference: Sense Choice 207
13.6 Preference: Example Analysis 212
13.7 Preference: Following a Chain 216

CHAPTER 14 REVIEW

14.1	Observations	221
14.2	Future Work	227
14.3	Summary	236
14.4	Conclusions	239

REFERENCES 241

CHAPTER 1

INTRODUCTION

1.1 WHAT IT'S ABOUT

This thesis presents a system for understanding natural language. By understanding, I mean the assignment of meaning structures to pieces of natural language texts. The particular meaning structures that we use will be explained later, but for the moment assume that meaning structures are symbolic representations of the concepts underlying language. Computational understanding, as defined here, is the extraction of conceptual structures from input texts.

The thesis consists of two parts. Part I is about a program for the meaning analysis of sentences. The program assigns meaning structures to a wide range of sentences. Typical of the class of sentences handled are the following:

> John gave Mary a beating.
>
> Bill prevented Rita from loaning the book by taking the book.
>
> Rita advised Mary to drink the wine.
>
> Did Bill know that John was hunting?

The surface syntactic structures of these sentences are fairly simple, but it is the complication in their meaning structures that makes these sentences interesting.

Part II is not about a program but about a broader theory, based on the analysis program presented in Part I. The topic of Part II is called extended analysis. Extended analysis is a system of organization by which the meaning structures and the structures representing the analysis processes are tied together. Extended analysis processes can directly affect not only static meaning structures but the flow of analysis itself. They handle multi-sentence texts, treating them as coherent entities, just as the sentential analyzer treated sentences as coherent entities. The discussion of extended analysis treats in detail two sentence texts such as:

> John hated Mary. John gave Mary a sock.
>
> John was feeding the deer at the zoo. John gave a buck some sugar.
>
> John saw a beggar on the street. John gave a buck to him.

Part II also treats, in broader terms texts in general, how they are understood as coherent units, and how the analysis of one part of a text affects the analysis of other parts of that text. Some of the work that was the impetus to Part II is embedded in Part I.

Contexts have been used in the analysis program to guide the analysis process, and how this guiding occurs is one of the main topics of Part II. So, in a larger sense, what is presented here is not only a program for the analysis of sentences, but a theory of contextual effects on this analysis.

1.2 COMPUTATIONAL UNDERSTANDING

There is not much need to explain why we might want a program that can understand natural language input. We just have to start listing phrases like machine translation, question-answering, man-machine interaction, computer-aided instruction, and so on. Common to all of these is the need for programs that can respond in useful ways to information expressed in a natural language.

However a computational understanding (CU) program can be more than just a program that responds usefully to natural language input. Tasks like the above determine what results a program must achieve, but they do not necessarily determine the methods a program can use to achieve these results. We have had as a goal in this work a CU system that could also be a model of human comprehension.

Given the obvious facts that computers aren't built from neurons and people don't have magnetic memories, what does it mean to say that a computational process is a model of a human one? The definition here will be that one process models another if the same decisions occur in the same order and for the same reasons in both processes. A decision is an action of the form "I choose to believe X because I believe A, B, C, etc." X, A, B, C, etc. are descriptions of situations. Naturally the representation of these descriptions will not be the same in a program as they are in a brain. But we can equate statements in a representation with beliefs represented in whatever way they are in the brain.

When a CU system, that is supposed to be a model of human language comprehension, processes a text, it produces a flow of decisions. This flow does not have to be identical to the flow of decisions someone might follow when comprehending the same text. It is unlikely that any two people process a text in exactly the same way. But we can, though not rigorously, look at the flow of decisions and decide if such a flow is consistent with what we know about human comprehension. If a CU system consistently produces such flows of decisions, then it constitutes a good model of human comprehension.

The focus then is on the decision making that occurs. That means that we are interested in what decisions are made, when they are made, and with what information they are made. <u>What</u> kinds of structures are built--syntactic? semantic? conceptual? <u>When</u> are they built--syntactic before semantic? while the text is being read? after the text has been reread several times? <u>On what information</u> are they built--syntactic structures? knowledge about politics?

A working CU system, one that produces acceptable paths of decisions, must contain answers to all these questions. It must have mechanisms that make these decisions. These mechanisms, particularly if they seem consistent with a computational model of memory processes in general, form an hypothesis about the nature of the mechanisms that underly human decision making.

In the next section the CU system of this thesis is described in terms of the general statements the system makes about language comprehension and the mechanisms underlying it.

1.3 ASPECTS OF THIS SYSTEM

Because our system is intended to be a model of human language comprehension, its existence makes a number of claims about the nature of comprehension. These claims fall into two groups. There are claims about general aspects of comprehension and there are claims about the specific mechanisms of comprehension.

General Aspects of Language Comprehension

This system has the following characteristics, which I believe are proper for any model of comprehension:

1) Comprehension is treated as a motivated process. It is driven by a constantly changing set of goals. These goals treat comprehension as a process of filling out a larger structure. This structure is a picture of where the discourse is going. If such a picture does not exist, there is a drive to find one. These drives motivate comprehension at all levels, from the choice of a meaning for a word to assumptions about the intention of a text.

2) The process of comprehension is not divided into separate modes of operation. There is not, for example, a syntactic phase which precedes a semantic interpretation phase. Nor does comprehension switch back and forth between phases like these. Instead, anything can happen at any time. At any point in the flow of analysis, there are operations manipulating concepts as well as operations interpreting individual words.

3) The knowledge gained at one level in the analysis system diffuses at once throughout the rest of the system. Comprehension involves many different levels of activity. New words are read in. New assumptions about the future of the discourse are made. But whenever something is added or deleted due to this activity, the news of this change is made available to all the other processes.

4) Comprehension is concerned with two important tasks. One is finding information relevant to the process of interpretation. The other is removing information that no longer applies.

5) There is a clear distinction between static knowledge and process knowledge. For example, the dictionary entry for a word in the analyzer may contain process knowledge. This entry makes no sense when considered apart from a flow of analysis.

6) The comprehension process is specified in terms of mechanisms consistent with a more general model of memory processes. The forms for representing static knowledge is likewise consistent with such a model.

Specific Mechanisms

We can characterize the process of comprehension as a process of applying dynamically selected pieces of knowledge to a continuing stream of language input. Dynamic selection means that there must be mechanisms for moving information in and out of active participation in the comprehension process.

Further, comprehension has two dimensions. It has a dimension of time. We say that an analysis begins at one point in time and ends at some later point. It also has a dimension of depth. We say that an analysis begins at the surface (i.e. the word) and ends at some deeper level of conceptual abstraction. There must be mechanisms for moving the dynamically selected information along both of these dimensions.

These tasks are done in the system by several basic kinds of data and control structures:

1) For accessing information--Associations are used for accessing information. Knowledge about the comprehension of a language is associated with word senses. These senses contain two types of structures. There are structures of static forms and structures of active instructions. Both of these structures assume that, when the word is read and these structures are activated, there will be other structures either already present or to come. These other structures will affect and be affected by the structures that make up the sense of a word. We cannot talk about the meaning of a word outside of the flow of analysis because the meaning <u>is</u> the flow of analysis.

    There is more knowledge about comprehension associated with certain words and concepts. This knowledge is about the larger picture that should guide the flow of analysis. The information is represented as sequences of patterns, and the positions in a sequence provide slots in which to place the results of analysis.

2) For passing information through time--The mechanism for passing information from one point in the analysis to a later point is the expectation. An expectation consists of a specification of a situation and a specification of what to do if that sitiation is encountered. The expectation is a prediction that the situation specified will occur. At the same time, the action specified is information that is being carried along in time. It is information about what the predicted situation means to the process of comprehension.

3) For passing information from level to level--Two mechanisms are used to pass information from one level of thought to another. One is the expectation. The expectation can pass the information that one type of structure exists by constructing a structure of another type.

   The other mechanism for passing information is called the need. When a structure exists but is incomplete, a need is generated to fill this gap. That is, a signal is generated that something should be done to fill that gap. This signal is passed from whatever level of abstraction that the structure is on towards the language level. The presence of a need alters the flow of analysis such that if the need can be filled, it will be.

4) For removing information--Information is removed in two ways. First, when a large structure specifying an overall picture is contradicted

by the results of an analysis, a new structure must be found. The removal of the old structure removes with it all the static forms and active instructions it may have included. It also removes the needs that the incomplete static forms generated.

The filling of gaps is the second mechanism for removing information. A need disappears when the form it came from disappears A need also disappears when the gap generating it has been filled. When needs disappear, the alterations those needs caused on the flow of analysis cease.

The system for comprehension thus consists of structures of static forms and of expectations. The expectations, originating primarily in the senses of words, specify the active instructions for analysis. The mechanism of the need forms the link by which the static forms communicate with these expectations. Associations between words and structures and associations within structures themselves, provide the paths by which the information is found and later collected together and removed. The organization of knowledge into these structures, the drive to complete partially filled forms, and the making of expectations, all belong to a set of reasonable general memory processes.

1.4 WHAT THE SYSTEM DOES

Since only Part I of this thesis is programmed as yet, we can only talk about output from the Part I form of the analyzer.

The analysis done by the program of Part I is one of comprehension rather than of parsing. The output of a simple sentence like "John gave Mary a book," for example, is not a description of the syntactic relationships between the words appearing in that sentence. In fact, none of the words in the sentence appear in the output. Instead the output is a structure of concepts, involving a basic action of transferral of ownership. The ownership of an object that is a book changes from a person named John to a person named Mary. The person responsible for this change is the same John who previously owned the book. Because the output of analysis is meaning and not syntactic structure, the analyzer produces this same output for the sentence "Mary received the book from John."

The meaning structures will be described in detail later, but an example of what they look like can be given here. We show here output produced by the analysis of a rather long sentence. We chose a long sentence for two reasons. First, the output from the analysis of this example shows many of the different meaning elements that can be used. Second, the example demonstrates that long sentences are not a problem for the analyzer. This is because the analyzer doesn't try to do the whole sentence at once, but rather takes it piece by piece, as it reads the sentence from left to right (although the output does not have to appear in the same order--see the "prevent" example that appears next).

The sentence is : JOHN HURT MARY BECAUSE MARY INFORMED BILL THAT JOHN ADVISED RITA TO PREVENT BILL FROM BUYING THE BOOK BY GIVING THE BOOK TO JOHN.

The output, obtained in 25 seconds of processing time, is (comments in lower case):

```
TIMØØ :   ((VAL T))                    This is a list of time
TIMØ1 :   ((BEFORE . NIL))             relationships.  Every
                                       event has a time which
TIMØ2 :   ((BEFORE TIMØ1 X))           specifies when that
                                       event occurred with
TIMØ3 :   ((BEFORE TIMØØ X))           respect to other events.
TIMØ4 :   ((AFTER TIMØ3 X))
TIMØ5 :   ((AFTER TIMØ3 X))
TIMØ6 :   ((BEFORE TIMØ5 X))
((CON                                  This part of the output
                                       structure says that Mary
  ((ACTOR (MARY)                       told Bill that...
          <=>
          (MTRANS)
          TO
          (CP PART (BILL))
          FROM
          (CP PART (MARY))
          MOBJECT                      ...John had told Rita
          ((ACTOR (JOHN)               that...
                  <=>
                  (MTRANS)
                  TO
                  (CP PART (RITA))
                  FROM
                  (CP PART (JOHN))
```

```
MOBJECT                         ...if Rita gave the
((CON                           book to John...
  ((CON
    ((ACTOR (RITA)
             ⟺
            (ATRANS)
            TO
            (JOHN)
            FROM
            (RITA)
            OBJECT
            (BOOK REF (THE)))
      MODE
      (NIL)
      TIME
       (TIM∅6))
    ⇐
    ((CON                      ...then Bill would be
       ((ACTOR (BILL)          unable to buy the book
                ⟺                   and...
               (ATRANS)
               OBJECT
               (MONEY REF (A))
               TO
               (NIL)
               FROM
               (BILL))
        TIME
        (TIM∅5))
```

```
                                ⟸
                                ((ACTOR (NIL)
                                        ⟺
                                        (ATRANS)
                                        OBJECT
                                       (BOOK REF (THE))
                                       TO
                                       (BILL)
                                       FROM
                                       (NIL))
                                 TIME
                                 (TIMØ5)))
                               MODE
                               ((NEG)))))
                            ⟸C
                            ((ACTOR (RITA)⟺ T (JOY)⟸⟹ F (JOY))
                              INC               ...this would
                              (2)               make Rita
                              TIME              happier.
                              (TIMØ4)
                              MODE
                              (NIL)))))
          MODE
          (NIL)
          TIME
          (TIMØ3)))
TIME
(TIMØ2))
```

```
 ⇐
((CON ((ACTOR (JOHN)⇔(DO)) TIME (TIMØØ) MODE (NIL))
         ⇔
       ((ACTOR (MARY)⇔T (UNSPEC)) INC      Because Mary told
                                           Bill all this, John
                                  (-2)     did something to
                                           hurt her.
                                  TIME

                                  (TIMØØ)

                                  MODE

                                  (NIL)))
TIME
  (TIMØØ)
MODE
  (NIL))))
```

We said that the analyzer, in building a structure like the above, went from left to right, but that the output did not have to reflect the order of things as presented in the sentence. This is because the analyzer can operate freely on structures like the above, redoing old parts as well as adding new ones. For example, the analysis of "John prevented Mary from going to the store by taking the bicycle" involves building one structure, and then modifying and expanding subparts of that structure. "John prevented Mary" is analyzed as "John did something which caused Mary to be unable to do something." When "from" is read, it is assumed that it introduces a clause specifying what Mary can't do. Hence, when "Mary going to the store" is analyzed, the meaning structure is changed to "John did something which caused Mary to be unable to go to the store." When the "by" is read, it is assumed that it introduces a clause specifying what John did. "Taking the bicycle" is analyzed and the final meaning is "John took the bicycle which caused Mary to be unable to go to the store."

The analyzer is not limited to verbs as a source of conceptual frames. For example in "John gave Mary a beating" the conceptual frame is provided by "a beating". The result of analyzing this sentence is the same as the result of analyzing "John beat Mary," namely that John hit Mary repeatedly.

The conceptual manipulations of conceptual frames can be fairly complex. In "John gave Mary a headache," "a headache" provides the basic conceptualization, but several things are added. First, of course, Mary has a headache. Further, John did something to cause Mary to have a headache. Hence a specification of causation is added. Finally, what was caused was not having a headache, but rather coming to have a headache. Hence the state of having a headache becomes instead a change to the state of having a headache.

During the analysis of a sentence, the analyzer makes many predictions. For example, when "prevented" is read it predicts that "from" and "by" will introduce clauses specifying what was prevented and how, respectively. When "gave" is read it predicts that noun phrases for both a human and a physical object will follow, that the human is the recipient of "gave" and that the physical object is the object. Thus the analyzer assumes that "Mary" is the <u>recipient</u> of "gave" in "John gave Mary...", but that "a rock" is the <u>object</u> of "gave" in "John gave a rock...."

To some extent, the analyzer as programmed already includes contextual effects, although a systematic treatment of context is not given until Part II. For example, the sentence "John gave Mary a sock" is normally interpreted by the analyzer as meaning that John transferred ownership of an item of footwear to Mary. If, however, the analyzer had previously handled a sentence like "John hated Mary," then it assumes that "John gave Mary a sock" means that John hit Mary. It does this using a prediction that John will do something to hurt Mary because he hates her.

These then are some of the kinds of analyses that are produced. The stress in all of them is on the construction and modification of meaning structures. What is also important are the ways in which these analyses are achieved. But to explain this, we must proceed to a fuller description of the analysis process.

PART I

GUIDE TO PART I

In Part I, we step back from the general claims of the introduction and focus on the process of analysis more specifically. In particular Part I is oriented around the description of a working analysis program. This program is described in detail, giving example analyses as well as the definitions of words that led to the analyses. This is done in Chapters 7 and 8.

The most compact and accurate way to describe the analyses and definitions is to use the forms that the analysis program does. However, this requires that the reader learn two "languages" first.

One is the language of the representational system. With this language we express the X, A, B, C, etc. in the decision "I choose to believe X because I believe A, B, C, etc." This language is called Conceptual Dependency and is described in Chapter 5. In Conceptual Dependency, beliefs are expressed using a small set of language-free objects and relationships. The goal of the analyzer is to assign to a text a Conceptual Dependency structure. This structure should express a belief that is a reasonable interpretation of the text.

The other language used expresses the active part of the decisions, i.e. the "I choose to believe" and the "because I believe". The basic element of this language is the expectation. An expectation is a specification of a situation ("I believe A, B, C, etc.") paired with a specification of an action ("I choose to believe X"). When the situation specified is encountered, the action associated is performed. Making a decision is therefore the triggering of an expectation. All this is described in Chapter 6.

Chapters 2, 3 and 4 introduce and motivate the directions that the developments in Part 1 take. Chapter 4 is concerned with the reasons why things are done the way they are. Chapter 3 discusses briefly some of the alternative approaches that could have been taken. And because there is a long stretch of theory (from Chapter 3 through Chapter 6), Chapter 2 offers some pictures of what it's all about for the reader to keep in mind. Of particular importance, I think, is the flow table in section 2.4. The theoretical discussion looks at the various functions one at a time, but the flow table presents them in their natural habitat, working together in a flow of analysis.

CHAPTER 2

SENTENTIAL ANALYSIS

2.1 INTRODUCTION TO SENTENTIAL ANALYSIS

Describing the analyzer in detail means specifying two aspects of it: the theory and the program. The program can be specified by presenting the functions with which it is built and showing how these functions are used in the task of analysis. The program aspect of the analyzer occupies the latter two-thirds of Part I. However, the construction of the program is not haphazard. It is the purpose of the first third of the description to present the concepts and motivations about which the program is organized. This is the theoretical aspect of the analyzer. To motivate this theoretical description, it is in turn helpful to talk briefly about the program to which the theory leads. In particular much can be said about the program from the outside without worrying about its internal machinery.

The program has the following properties which distinguish it from other analyzers:

1) Its object is not to parse a sentence into a syntactic structure, be it surface or deep. Its goal is to discover the meaning of the sentence, in the context in which it appeared.
2) Even as a tool, syntax plays a small role in the analysis. Of greater importance to the program are the partial meanings that are found during the processing. These provide much of the in-

formation necessary for continuing the analysis.

3) There is not a clear division between linguistic and non-linguistic knowledge. Beliefs and inferences can play an important part in the determination of meaning. The representation used allows the analyzer to interact when necessary with a memory model.

4) The sentences understood are about human behavior. The program makes assumptions about the normal relationships involving people with other people, and people with objects.

The *expectation* is the basic mechanism used by the program. An expectation is a description of a situation that is recognized as likely to become true in the near future. Further, associated with any expectation is a set of actions to perform, actions that are appropriate in the expected situation. In general an expectation organizes information so that one can respond appropriately to a situation as soon as that situation is recognized. The importance of an expectation is not just that it prepares a set of actions for use if needed, but also that it narrows how future situations are perceived. An expectation looks only for certain features and ignores any other features of the real situation that may be present. In language processing, for example, an expectation may predict that a certain preposition has some particular meaning, and thus the many other possible meanings will not be seen. While we can imagine expectations for many situations, of concern here are those expectations that are closely related to language comprehension, as well as what their sources and their effects are.

The demon mechanism found in Charniak (1972) is similar to the expectation. Charniak uses expectations about human behavior in specific context to help solve problems of anaphora. However, language processing is bypassed and the semantic representation for the demons is not generalizable beyond the particular examples considered.

The program here depends upon the use of a general representational system for meaning. Expectations about real world situations can be described in many different ways in English, but the program must be able to recognize these situations no matter how they are described. Further, with a meaning representation, it is possible for an expectation to be generated from a non-linguistic source and still be able to communicate with the analysis. For example, if I hear the sentences "John was hunting. He shot a buck," I will interpret the second sentence as meaning John shot an animal, not that he spent a dollar. I would make this same interpretation for the same reason--i.e. from the same expectation--if I saw John coming out of the woods with a rifle in his hands, and he told me he'd shot a buck.

2.2 HISTORY OF THE ANALYZER

The description of the theory and programming of an English analyzer will be a static one, that is, it will look at the system as something in one fixed and final form. But this kind of description omits important aspects of the topic just as much as a synchronic description of a language omits important aspects of that language. For example, a static image cannot show whether the system is a blueprint for future work or the product of past effort. And it is important to know this when trying to decide if the claims made for the system are just beliefs or if to some extent they have been substantiated or inspired by actual experience. For this and other reasons, a brief history of the growth of the analyzer will be given before the more detailed description of its workings.

The work on this particular system took place over a one year period at Stanford University. Prior to this there had been an analyzer written (see Schank and Tesler (1970)) which also went from natural English sentences to Conceptual Dependency representations, but beyond this common goal there is no real connection between the two efforts.

The stages of development of the program were:

1) A general outline of an approach to analysis- At this time, fairly general assumptions were made about the purpose of writing an analysis program, and what sorts of problems of language analysis should be focussed on. Also at this stage a simple control structure was decided

upon such that most of the programming work work would be in the form of dictionary expansion. Finally, certain specific sentences exemplifying the kinds of problems of concern were chosen as the initial goals of the programming.

2) Writing the program-First a very small control program was written. Then the simplest of the sentences from stage (1) was chosen ("John gave Mary a book"). After deciding what seemed to be occurring during the analysis of the sentence, definitions for the words in the sentence were written that would follow the same path of actions. Sometimes the definitions required new functions and the control structure had to be expanded to be able to handle them. The process of taking an example and expanding the vocabulary to handle it was the basic means of growth in the analyzer. Words were not considered by themselves but in their interaction with other words in specific contexts. The sentences chosen for each iteration of the expansion were intended to be as different from the previous ones as possible, in terms of the kinds of processes invoked. An important step in the expansion loop was to test the program on a file of previously resolved examples. Though changes in the analyzer were normally made only to definitions of specific words and hence did not affect the actions of other parts of the analyzer, still it did happen that an injudicious sharing of

global variables or something similar led to the destruction of previous successes. The corrections required were never important theoretically however.

3) MARGIE-The next stage involved the addition of a motivation for choosing sentences for the expansion process. Being developed concurrently with the analyzer were a deductive memory system using Conceptual Dependency representations as input and output and a generator using Conceptual Dependency representations as input and English sentences as output. It was decided that the three programs should have a common data base so that they could be used as one entity. However, no particular domain of discourse was chosen. Rather, for each of the programs, sentences and conceptualizations that demonstrated important features of it were picked by the worker developing the program. The other programs were then extended where needed to allow these demonstrations.

4) Diplomacy-At approximately the same time as stage (3) a different motivation was applied to the choice of sentences in the expansion process. In contrast to stage (3) where the interest was in demonstrating directly certain aspects of the system, in stage (4) the interest was in handling a specific task domain. The domain was the game _Diplomacy_. Diplomacy although a board game, differs strongly from others like chess and go in that it depends heavily on interpersonal communication. Fur-

ther, success in the game depends primarily on being able to influence, honestly or dishonestly, other people to do what you want, and at the same time judge how and why someone else is trying to use you. Hence there is a great deal of concern with human perception and communication of human behavior. A number of words were added to the analyzer's vocabulary to help it to deal with this domain. All of the work on this particular extension of the analyzer was done by Paul Martin, basing his choices on sentences that had been recorded during actual Diplomacy games between human players.

5) Description-There is a very noticeable effect upon the development of a system when the designer has to stop, organize, and summarize the system in order to describe it and its goals to other people. This work on the analyzer has been described in successively greater detail in the following papers: Goldman and Riesbeck (1973), Riesbeck (1973), and now here. Not only are oversights and ad hoc measures suddenly made embarassingly clear, but more importantly patterns are seen, initial assumptions are clarified and generalized, and a better view of where to go next is obtained.

What follows therefore is the result of these five stages of work. The program itself has been expanded and modified at each point. The functions described are the ones that were found necessary as the stock of examples grew. The theoretical implications of various aspects of the program are also the outcome of these five stages. In particular most of the relationships between standard linguistic concepts and elements of the analyzer were realized during the descriptive stage of development. And finally the form of the description of the analyzer is a product of the way the analyzer grew and also of the previous attempts to describe it. Specific functions appear before their use in structures, so that these structures can be given compactly and exactly. The theory appears before the program because much of the theory was developed first and shaped the program in crucial ways. And an overview appears before the theory to suggest the nature of the concepts with which the theory is concerned.

2.3 THE PROGRAM ITSELF

The program itself is in two parts: a monitor and a dictionary. It runs on the Stanford PDP-10 system at the Stanford Artificial Intelligence Laboratory. The dictionary is written in Lisp 1.6 (see Quam and Diffie (1972)) while the control program is in Mlisp (see Smith (1970)), an Algol-like language that is translated into Lisp 1.6. For normal expansion and debugging an interpreted version of the control program is used, while for demonstrations a compiled copy is made.

Because the program was not designed with a specific task domain in mind, nor to be impressive in demonstrations, nor to study the difficult questions of large data base management, the vocabulary is fairly small. There is a core of about 60 verbs plus their attendant prepositions, and enough nouns and adjectives to construct long sentences without being forced to unnatural actors and objects. Because of the principles guiding expansion in stage (3), however, each of these verbs demonstrates quite different aspects of analysis. Some stress manipulations performed directly upon conceptualizations. Others stress the operations needed to reach these conceptualizations from various word constructions.

The program, with the Lisp interpreter and the Mlisp translator and sufficient working space, filled about 50,000 PDP-10 words. When the control program was compiled and loaded without the Mlisp translator, the total was 35,000 words. The CPU time in the interpreted version for handling sentences including those that will be discussed was between

5 and 10 seconds, while it was less than 2 seconds in the compiled form. Both time and space could have been greatly improved if that had been of interest. For example, predicates are re-evaluated even though it is known that none of their parameters have changed and re-evaluation could be avoided. Also many of the dictionary entries have segments that could be shared with other entries. But the purpose of the program was to be a concrete expression of a theory, such that changes in the theory could be tested and, at the same time, extensions to the program would add to the theory. For this task the analyzer program was always satisfactory.

2.4 OVERVIEW OF THE ANALYSIS PROCESS

The analysis of a sentence is driven by the execution of programs attached to each of the words that occur in this sentence. But the programs associated with a word are not executed immediately upon the discovery of the word. Instead, to each program is attached a condition which the state of the analysis must fulfill before that program is executed. This pair, of condition and program, is a <u>request</u>, and the two parts are called <u>test</u> and <u>action</u> respectively. The whole package of requests which is attached to a word is called the <u>sense</u> of that word. Also part of the sense of a word is a set of features that describes various passive apsects of the word or the concept to which it refers.

The analysis of a sentence consists of two activities:

1) The requests attached to a word are added to the list of requests active, i.e. those requests that have previously been added, but whose conditions have not yet been fulfilled.

2) The list of requests is rechecked whenever new words or concepts occur to see if any of the conditions of the requests have become satisfied. If so, the actions associated with these satisfied conditions are executed, and then the list is rechecked to see if any more conditions have become fulfilled.

These two steps make up the monitoring control program of the analyzer. This program looks for knowledge about the language relevant to the text being analyzed and applies the information that is found.

There will be many examples given later showing exactly how the analyzer works. However, a general idea can be obtained by following, in less detail, the analysis of a simple example. Consider the sentence, "John gave Mary a beating." It is assumed that this sentence has the same meaning as, "John beat Mary." The chart that follows shows an outline of the basic flow of decision and prediction that the analyzer goes through with this example. "NP" is, of course, an abbreviation for "noun phrase". Under the heading "Requests Waiting" appears only the test portion of each request. When the number of a request appears under the heading "Requests Triggered", it means that the test of that request was fulfilled. The action portion of a triggered request follows, under the heading "Actions Taken".

STEP	WORD READ	REQUESTS WAITING	REQUESTS TRIGGERED	ACTIONS TAKEN
∅	none	1-is there an NP?	none	none
1	John	1-is there an NP?	1	assume "John" is subject of the verb to follow
2	gave	2-is the current NP a human? 3-is the current NP an object? 4-is the current NP an action? 5-true	5	assume the word "to", if it appears, introduces the recipient of the "giving"
3	Mary	2-is the current NP a human? 3-is the current NP an object? 4-is the current NP an action?	2	assume Mary is the recipient of the "giving"
4	a	3-is the current NP an object? 4-is the current NP an action? 6-true	6	save the current list of requests and replace it with: 7-does the current word end an NP?
5	beating	7-does the current word end an NP?	none	none
6	period	7-does the current word end an NP?	7	build the NP "a beating" and reset the list of requests
7	none	3-is the current NP an object? 4-is the current NP an action?	4	assume the NP action is the main action of the clause, the subject (John) is the actor and the recipient (Mary) is the object

After Step 7 there are no more words to be read and no more requests are triggered, and so the analysis stops. The final result is that John beat Mary, i.e. that John hit Mary repeatedly, presumably with his hand. Most of the conceptual action except for "John" and "Mary" came from the dictionary entry for the sense of "beating" that was assumed.

Step Ø is, of course, the same in any analysis. In step 2 there is request 5 which has a test that is always true. Such requests are executed immediately. This one changes the sense of the word "to", which might occur later. This is one of the basic ways in which the analyzer tries to avoid problems of ambiguity. The prediction made might be wrong, but it specifies the most reasonable meaning of "to" to try first. Step 4 shows how noun phrases are built. A noun phrase introducer, like an article, temporarily halts other processing until a noun phrase is collected. Step 4 also shows an instance of one request changing the set of requests waiting. Step 7 involves the direct manipulation of conceptualizations. Both Step 4 and Step 7 show how far removed from syntactic manipulations the actions that occur during analysis can be.

A basic feeling for the flow of action during analysis is important for reading what follows. The form of the program used has a number of implications and there will be many comments on these implications. How concepts like meaning and ambiguity relate to requests and senses, what role syntax plays, what aspects of a theory of language comprehension are stressed in this format, and other such topics shall be touched upon.

2.5 SENSE VERSUS MEANING

The term "sense" has a very specialized meaning for the analyzer. It is the label for a package of requests and features that together describe how a word with that sense interacts with other words in a text. A sense, therefore, is not directly equivalent to the meaning of a word, i.e. to a conceptual representation. It differs in two ways from the idea of meaning:

1) The same sense of a word in two different contexts may produce two different meanings. For example, to "break a promise (vow, oath, rule)" while containing a different meaning for "break" than that found in "break a bottle (table, window)" does not require two different senses of "break". One sense, containing a request that tests for a physical object of the "breaking" and another request that tests for an obligation as the object of "breaking", will be able to handle both uses.

2) More importantly, a sense is distinguished from the idea of meaning in that the task of a sense is to contribute to the interpretation of the text as a whole, and not of the word alone. There may be no simple relationship between any subpart of the final conceptualization and some word from the initial sentence. For example, the interpretation the analyzer gives for "John gave Mary a beating" is that John hit Mary re-

peatedly. The contribution of the sense of "give" used is to take the conceptual structure of "beating" and insert John and Mary as the participants, in the correct roles. "Give" itself, however, produces no conceptual piece of its own. The role of "give" in the sentence is only definable in terms of the actions that its sense performs upon other conceptual structures.

Because a sense is a collection of requests, any change to that collection--addition, deletion, or modification--in effect creates a new sense. It is a matter of pragmatics whether the changing of a word's sense should be done by changing the list of requests it has or by replacing that list with a prepared one that embodies these changes. If there are several commonly used packages of requests for a word, then each of these can be saved and used when some request from some other word or concept calls for it.

CHAPTER 3

PREVIOUS WORK

The analyzer just described is based on some very fundamental assumptions about language processing. These choices are not unexpected ones, from an historical view of computational linguistics. They follow a trend towards greater and greater emphasis on semantic and conceptual matters rather than on syntactic ones. But as a result, there is not much relevant preceding research from linguistics and computational linguistics, both of which have stressed the syntactic approach.

There are two previous computational analysis systems though that have been highly influential on the directions that later work has taken, and no new effort can ignore them in discussing alternatives. I refer to the work of Woods (1970) and Winograd (1971). The former produced a very successful syntactically oriented parser. In doing so he verified a set of mechanisms of interest to other designers of syntactic systems. Winograd produced a complete system, an integrated set of programs, which was able to carry on discourse with a human about a changing world. However, each component was limited in domain. As with Woods' work, both the success achieved and the presentation of a tested set of mechanisms for attaining this goal made Winograd's work important.

Before describing where I have gone I must describe where I have not gone and why. What assumptions were basic to the work of Woods and Winograd that kept me from expanding on either of their approaches? And what assumptions were basic to the analyzer that was developed? The first question shall be answered here, and the second in the next section.

The parser written by Woods is a syntactic one. The mechanism for parsing is the augmented transition graph. It enables a programmer to specify a grammar for a language in a form that can be understood by a linguist and at the same time can also be used by a program to assign syntactic structures to sentences in that language. The basic format is a transition graph, that is, a description of a flow of decisions in terms of nodes (or states) and arcs between these nodes. An arc is associated with a condition and traversing that arc is equivalent to deciding that that condition is true. The parsing task is basically one of recognition. At the beginning of the analysis the program is at the initial node of the graph. From this node lead a number of arcs, each of which, in the simplest system, is a condition on a word, for example that it is a noun or a verb. The first word is taken from the sentence. From the set of arcs whose condition is true of the word, one is chosen and traversed. This leads to a new node and hence a new set of exit arcs. The next word is taken from the sentence and so on. Certain nodes are specially treated. If the sentence is finished when the program is at one of these nodes then a successful parse has occurred. Otherwise an error has occurred and backup must be done. This kind of system is much too limited for handling unrestricted English. Two mechanisms are added that extend the power of the graph concept without destroying its clarity.

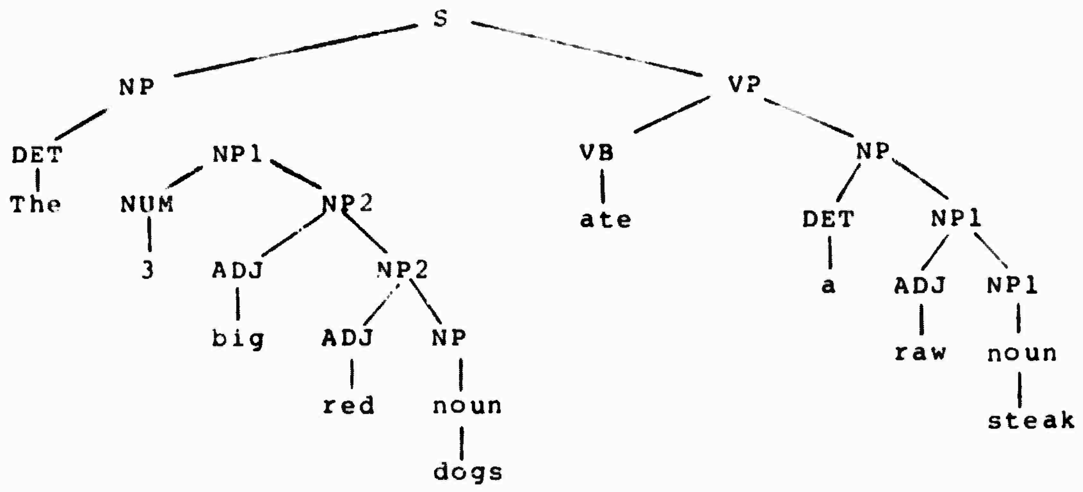
One of the mechanisms is a set of registers. An arc predicate can refer to them and the traversal of an arc may change their values. In this way information can be passed from one step to another.

The other mechanism is recursion. Arcs are not limited to being conditions on words. They may refer to structural conditions that are themselves recognized by augmented transition graphs, including the one to which the arc belongs. Thus, instead of an arc looking for a noun, there can be one looking for a noun phrase. Together with functions for passing register values through such calls, these two mechanisms make the transition graph powerful enough to handle the syntax of a natural language like English.

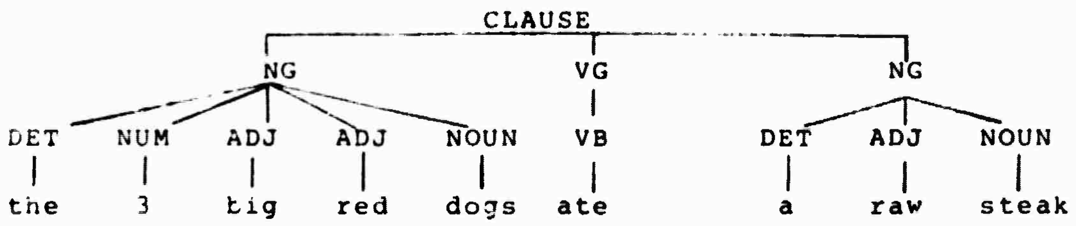
The role of semantics in this system is to be a check and guide for syntactic analysis. At certain points semantic routines are referenced in order to try and help out syntactic decisions, but because semantic routines tend to be powerful but slow, it is the job of the designer to balance the use of semantic and syntactic means, so that semantics is done only when it can improve upon fairly fast syntactic routines.

Backtracking is very simple in graphs. It is done whenever the parser finds itself at a node from which no exit arcs have conditions which are satisfied. When this occurs the parser returns to the node previous to this one and tries another possible exit from that. If there are none, backtracking occurs again and so on.

Winograd's work has a syntax based loosely on the linguistic system of Michael Halliday (1970). Sentence are looked upon as sets of choices of features, features which are grouped according to the rank order of the sentential unit to which they apply. Thus, there are features of words, of phrases, and of clauses. Some features are possible options (in the generative grammar sense) only if certain other features are present or absent. A sentence like, "The three big red dogs ate a raw steak," is parsed not into a syntactic tree, such as:



but into levels of features as below:



The parsing task is to find patterns in sentences and decide what features they indicate. Once these features are found, semantic routines that are attached to them are executed. These routines, which for Winograd are the meanings of the features they are attached to, are not concerned with building the parse tree as they are in Woods, but with the manipulation of a world model. And although there are features that apply to the sentence as a whole, e.g. declarative and interrogative, there are others that occur closer to the word level. Because of this, some semantic routines are executed before the processing of the sentence is completely finished.

Further in contrast to Woods, the homogeneity of processes is lessened even more by the attaching of specialized routines called <u>demons</u> to certain function words, like "and"

and "or". These demons are procedures that interrupt the normal flow of parsing to take care of the special needs of the words to which they are attached.

Backup is greatly affected by this approach where a feature once recognized can cause fairly complex programs to be executed. Unlike the transition graph model, the flow of actions in Winograd's parser is not easily describable. A benefit of using a homogeneous, restricted set of functions such as Woods' is that decisions can be retraced automatically. But in a system where procedures are of many kinds, other means must be used to meet the problem. One possibility is to use the decision point capability of a language like PLANNER. With this device decisions can be marked as being places from which to start over if something goes wrong later. However, the approach favored by Winograd is that an intelligent analyzer should be able to decide where it should start over, by looking at the nature of the mistake.

Comparing my analyzer to the work described above brings out some similarities. For instance, the control structure I have used is similar to the augmented transition graph of Woods. In both cases the process of analysis moves from one set of actions, each contingent on some situation, to another, and a large part of the analysis consists only of these transitions. But the control structure is about the only thing in common between the two systems. The content and intent of the functions involved are very different. In content, Woods' functions ask questions of syntax. In intent those functions are meant to build parse trees, syntactic descriptions of sentences. In contrast, the functions of the analyzer to be described here ask questions about the relationships of words and concepts, and about the conceptual structures communicated by the sentence whihh have been assumed so far. This is the content of the analyzer's functions. In intent, the analyzer is trying to build a unified conceptual structure that is the meaning of the sentence. For this, syntactic structures are a tool and no more. And when it comes to backup, there is no intention that this should be a simple retracing of the decision path. Given the number of different things that the functions of the analyzer do, retracing does not seem at all reasonable.

The heterogeneity of functions is a similarity between the analyzer and Winograd's system. But there is still in Winograd's work an emphasis on syntax. Winograd's is a language processing program that looks for certain syntactic constructions. This program can be interrupted, but the basic flow of the system is still one of syntactic analysis. In contrast, my analyzer can be considered to be nothing but interruptions. Some words have more drastic routines than others, but all of them can play an active role.

This difference between Winograd's work and my own is reflected in the fact that there are no explicit specifications of global syntactic structures in my analyzer. Structured patterns, patterns that can be held apart from the words involved, do not occur here. Implicitly such patterns can be there in the same way that any consistently applied process can be described by the patterns of its behavior, but explicitly there are no such patterns.

Backup in the analyzer, as it is planned, is distinguished from the approach of Woods and like that of Winograd in that there is no backing up to decision points. Rather the analyzer begins again, starting with some point in the surface of the sentence, with certain choices ruled out and certain subanalyses left alone. The choices that are forbidden, the subanalyses that are left alone, and the point at which reanalysis begins, all this should be determinable by routines that look at the problem that has arisen and make guesses about the cause. This is counter to a decision point mechanism, the use of which would be making an implicit claim, that when people make decisions, they expect them to go wrong. But the problems that cause backup are not expected They are surprises. A joke such as, "I was on a hunting expedition. I shot two bucks. It was all I had," depends on this element of surprise. Unfortunately saying this about backup is really saying only that backup is too serious a problem to be handled with one simple mechanism. Intelligent routines that can direct reanalysis are still a thing of the future.

The representation of meaning is another dividing point between this work and the others. For Woods, the semantic base is defined specifically for the task. Thus there is a highly specialized semantic base for handling lunar samples. This allows him to represent, in a manipulable but non-extendible form, fairly complicated entities and relationships. The manipulations however are for the most part information retrieval actions, that is, search and pattern match routines on canonical forms. Winograd's base, on the other hand, deals with more concrete entities and relationships, namely blocks and their positions in space, and actions that involve changes in these relationships. With this base the program is able to perform more complex inferences than Woods', and do more than retrieve pre-stored information. However it is limited to simple physical objects and relationships, and it is not clear how well the system could be extended to a data base involving beliefs about people and objects, about probabl intentions and normal functions.

In the analyzer the base is the Conceptual Dependency

(CD) representation system, which has developed from the work of Roger Schank (1972). The primitive units and basic relationships have been worked out independent of particular tasks. The overall criterion is that CD graphs should differ if and only if the meaning being represented differs, excepting differences due to logical connectives. It is thus intended that two sentences have the same CD representation if and only if they are paraphrases.

This representation has been designed with an emphasis on the kinds of concepts people use in dealing with other people. Concepts like beliefs, communication, intention, reasons, and results, are the focus of study. More abstract systems like set theory can be elegantly formalized and efficiently programmed, but have little to do with human concerns.

The distinctions between the analyzer and the other two systems are, I believe, quite basic. They stem from differences in philosophies. The goal of the development of the analyzer has not been one of building an immediately practical understander of the English language. Nor has it been an experiment to see how much can be done with a small, restricted set of functions. It has been a search for mechanisms that seemed reasonable for people to use when they understand language. There were basic assumptions about the nature of the output of analysis, such as how deep comprehension should be taken, before it became a matter of non-linguistic cognition rather than language analysis. There were also assumptions about the nature of the input to the analyzer, about what information came from syntactic patterns, what from inference, what from world knowledge and so on. The designing of mechanisms specific enough for programming was directed by some assumptions, led to the recognition of others, and caused the modification of some that had been accepted.

Besides the work of Woods and Winograd, there is another effort involving language analysis that has not yet had the influence upon the field that these two have had. However, it shares several basic features with the work to be described. It is a very distinctive approach; it shuns the use of syntax in the extraction of meaning; it has a well-defined, broad-base semantic representation system, and its mechanisms are the basis for describing a theory of natural language inference. I refer to the work of Yorick Wilks (1973a, b) at Stanford University.

His effort has been oriented not about analysis itself but about the total task of machine translation. The stress of his work though has been on the analysis portion. It is up to the analysis routines to take English texts, disambiguate the words and semantic relationships involved, and settle questions like anaphoric reference, to the point where the generation routines can construct French texts as output. There is a working program that is quite impressive.

Briefly described, and ignoring the formalism, the system consists of a number of routines, each of which takes the output of some other one and creates a new structure with more relationships between the semantic elements than before. Eventually there is a structure ready for generation into French. The first of these routines takes the basic input text and fragments it, using various _key_ words and rules about special cases. The fragments of text are then passed to a routine that looks at the possible senses each word can have. These senses are expressed by _formulas_. Formulas are list structures made up of _elements_. There are sixty primitive semantic elements, divided into five different classes: entities (MAN, STUFF), actions (FORCE, DO), types (KIND, HOW), sorts (CONTAINER, GOOD), and cases (SUBJ, TO). Each formula, i.e. each word sense, has one element that is its <u>head</u>. The first step in the choosing of word meanings is to form templates from the heads of the formulas that appear in the text fragment. <u>Bare templates</u> are simple triples of elements in the pattern (actor action object). There is a list of permissible bare templates. Each of the possible combinations of word senses forms a template, but only those which appear in the list are kept. This causes some reduction in the number of possible word sense combinations. Then each of the sense-formulas in the template is checked for specifications of preferences about the other elements in the template. The templates that have the greatest number of preferences satisfied are kept for the next stage. Eventually, if there still exists ambiguity, i.e. several templates, a set of Common Sense Inference Rules (CSIR) is applied. These take a template and produce a new one, an inference from the old. These new templates are then used to try and resolve the difficulties. For example, in "The soldiers fired at the women and I saw several fall," there is an ambiguous reference with "several". There is a template for the fragment "soldiers fired at the women" and two for "several fall". There is a CSIR that takes the template "one strike another" and produces the new template "other falls". Applied to the first template for the sentence, the CSIR yields the template "women fall" and this matches one of the templates for the second sentence fragment. If no shorter chain of CSIRs is found, this match is preferred and "several" is assumed to refer to "women".

There are several aspects of this work by Wilks that are aspects of my own work as well. He claims that much of syntax can be handled by semantics. The semantics he uses is not devised for a particular topic domain but is based on a set of general primitives. The treatment is intended to be applicable to texts both shorter and longer than single sentences. The mechanisms, particularly the templates, are part of the vocabulary for his theory of context.

However, anyone looking at the two systems will undoubtedly see more differences than similarities. The basic scheme of analysis, which takes a chunk of text and reprocesses it over and over, is totally counter to the word-by-word, left-to-right method of my analyzer. All the active parts of his analyzer are in routines which manipulate the static structural descriptions associated with each word. Knowledge about language analysis, such as the passive construction, ("John was hit") or the similar "give" construction ("John gave Mary a beating") would appear not in the definitions of "be" and "give" but in routines that noted the occurrence of "be" and "give" and performed some kind of transformation. This also is counter to the approach of my analyzer.

Despite agreement by Wilks and myself on topics like semantic structures versus syntactic ones, the two analyzers are very different in design. This is because of the difference in task. For Wilks the task is machine translation. For myself the task is developing a model of human comprehension of natural language. Nothing about these two tasks demands that they be handled differently. The analyzer of this work could be the front-end of a translation system (as in a sense it was with the MARGIE project--see "History of the Analyzer") and one could claim that

Wilks' analyzer is a model, if not of human language comprehension in general, at least of human comprehension when translation is being done. It is certainly the case, though, that the two different concerns did lead to very different results.

The point of this is to emphasize a theme that is in the background of much of the work in this theses. The theme is the importance of keeping a goal in mind during development work. The differences between my work and that of Woods, Winograd and Wilks stem from the fact that the goals behind my analyzer, which are described in the next section, are not the goals of the others. Differences ingoals, even if the various goals are consistent with each other, can lead to drastically different results. For this reason the goals and assumptions that will be described are of crucial importance in understanding what has been done. And this point shall reappear throughout this work, either explicitly or implicitly, when a discussion is raised not on how something <u>can</u> be done but on how it <u>should</u> be done.

CHAPTER 4

BASIC IDEAS IN THE ANALYZER

4.1 GOALS AND ASSUMPTIONS

The form of the analyzer described is the result of assumptions of two different types. One set of assumptions is concerned with how the analyzer should carry out the task of assigning meaning representations to sentences. This set, however, is dependent upon another set of assumptions. These are the ones that specify the purpose of writing an analysis program at all. That is, the goals of the program are determined in part by the goals of the theorist.

The goal of this work as a theory is to investigate how language might be comprehended by humans. Given certain intuitive beliefs about the process in general, what specifically would an analyzer based on such principles look like? The questions to be answered are:

1) What assumptions and decisions are made during the comprehension of a sentence?
2) When are the assumptions and decisions made?
3) What are the reasons for these assumptions and decisions?

The assumptions of concern in (1) are primarily those that are about the meaning of a sentence. There are other conclusions that may be drawn by someone when hearing a sentence, such as the educational status of the speaker, but these are not directly related to the meaning the hearer finally gives to the sentence. The answers to (2) are not ones of time but of position. That is, if a sentence could be said to take five seconds to understand, the answer being looked for is not that a decision took place two seconds after the sentence analysis began. Rather, answers to (2) take the form "as soon as a decision is made about A, then a decision is made about B, but not before." This, of course, is related to (3) because if a decision A waits for some other decision B to be made then decision A probably has as one of its reasons some information from decision B. And in (3) as in (1), the interest is on decisions made for reasons that are related to meaning. Thus the theory does not try to deal with an effect on the analysis due to something like knowing that the speaker always uses certain words in ways which are different from most other speakers.

These assumptions about the object of study stress the decision aspect of language analysis. This is reflected in the form of the data/control structure of the analyzer program. Decisions and their reasons are set apart from the rest of the system so that they can be seen more clearly and be more easily expanded and modified. The basic data about language is contained in specifications of language situations and appropriate actions for these situations.

Once it is decided that the analyzer should be a concrete expression of a theory of human language processing, then the tasks of the program are defined by the assumptions of this theory. There were four initial assumptions made about human comprehension which were relevant to the program design:

1) The primary goal in the analysis of a sentence is to find an interpretation for that sentence, to find ideas that are the same as or similar to those which the speaker wanted to communicate with the sentence.

2) Decisions about concepts and conceptual relationships for the sentence are made while the sentence is being read. One does not wait until everything is present to start making decisions about the meaning of the first word of the sentence.

3) Previous choices have prepared for later choices by making predictions about likely situations to watch out for. The end of a sentence is guessed at after the beginning is understood. For example, the objects of a verb are predicted to be in conformity with restrictions specific to the verb and to the context of the sentence.

4) The first things recognized in a sentence are its words. Once the words are seen, meanings can be assumed and predictions can be made. But first the words have to be seen. The word is thus a basic element. Associated with the words are not only the concepts that the words refer to, but also the expectations that predict what words and concepts might co-occur and how these words and concepts are related to the total meaning being communicated.

Assumption (1) includes a claim about what people do with language. It says that they communicate thoughts with sentences. Basic to Conceptual Dependency theory is this assumption that thoughts are different than sentences, that ideas are built from language-free concepts and language-free relationships between those concepts. Communication is the transformation from a thought to an utterance by one person, back to something like the original thought by another person. The transformations are between structures of two very different types, between concepts and sound patterns.

This emphasis on meaning does not mean that syntax might not play an important part in the analyzer. However, the next three assumptions do greatly limit the role that syntax plays in the system described.

Assumption (2) says that concepts are chosen while the sentence is still being read or heard. This does not directly say that syntactic structures should not be built, but it does reduce their importance. Syntactic structures that apply to the sentence as a whole will be recognized only at the end of the sentence, by which point, by (2), the meaning of the sentence should be known already. Such large structures thus are seen too late to affect the flow of the analysis.

Assumption (2) provides, with the idea of predictions, a mechanism for analysis. Predictions are links that tie the words together in a sentence (and tie sentences together in a text). An augmented transition graph, such as Woods', uses a prediction mechanism to do syntax analysis. Coupled with assumption (2) however, assumption (3) leads to predictions concerned not with syntactic structures but with conceptual ones. It turns out that this further reduces the use of syntax. Instead of predicting a syntactic structure which has a certain meaning the analyzer predicts the meaning directly. For example, with a prepositional phrase such as "on the door of the house on the hill" it is not necessary to preserve (or predict) syntactically relationships that have to be made clear conceptually anyway. The phrase can be treated as three separate cases of prepositional government, "on the door", "of the house", and "on the hill".

The fact that the door is part of the house which is on the hill is conceptual information, not syntactic.

Assumption (4) emphasizes two concrete elements of language, word and meaning. Words and not syntactic structures are the source of expectations in a sentence. This is because syntactic structures, insofar as they can be said to exist in this analyzer, are themselves predictions. They originate in words that predict the appearance of certain words and word-classes. Therefore, if a syntactic structure makes an expectation when it occurs, then the words that predict that syntactic structure could make this expectation when they predict the structure

The analyzer that arises from these four assumptions is from the start different from others. It differs from those that generate syntactic structural descriptions, and it differs from those that look for patterns in sentences. I believe that all previous efforts fall into one or both of these sets. Most programs have generated syntactic descriptions of sentences. Those that didn't, such as the first attempt at a CD analyzer (Schank and Tesler (1970)) relied on pattern matching to extract the features they needed from the sentences they analyzed.

4.2 PATTERN MATCHING

Pattern matching is a very flexible mechanism and, in a suitably designed programming language, some algorithms can be easily given in terms of input and output patterns (see Enea and Colby (1973), and Tesler, Enea and Smith (1973)). However pattern matching was not used to a great degree in the analyzer. There were four characteristics of pattern matching systems that made them inappropriate for the approach used:

1) Lack of communication-When a pattern match fails there has been work done prior to this failure and information gained that could save labor in trying the next pattern. In a pure pattern matching system however, each pattern would be a fresh start and the information would be lost.
2) Ordering demands-It may be the case that the mere presence of several features, regardless of their order of appearance, is the crucial factor to look for. In a pure pattern matching system, however, patterns are specified as linear, hence ordered, strings of elements.
3) Inflexibility-It may be the case that elements in the input need to be classified according to a feature that has been dynamically produced and would not be feasible as a normal feature, e.g., "object broken by bottle dropped by John." A pure pattern matcher however is based upon a static classification.
4) Rewriting-Pattern is often viewed as a form of rewriting, that is, the input pattern is rewritten as the output. It is unnatural, I feel, to look at a meaning as being the re-

write of a word. Certainly the word is still present after its meaning has been assigned. The word is not converted into a concept, but rather, at the most, it is associated with one.

These characteristics are only biases, not insurmountable limitations. However, the more mechanisms that are introduced to remove these aspects and the more they are used, the less distinction there is between a pattern matching system and any other programming device. Finally, it is my belief that natural language analysis is a domain where such extension mechanisms would be necessary. Hence pattern matching is not a central aspect of the analyzer.

4.3 AMBIGUITY

The problem of ambiguity has been a major one for computational linguistics. Usually two types of ambiguity are recognized: semantic and syntactic. An example of semantic ambiguity is "The prince held a ball" where "a ball" might mean a round physical object or it might mean a type of gathering. An example of syntactic ambiguity is "I told Mary to keep her quiet" where the interpretation of the sentence might be that I told Mary something in order to keep Mary quiet or that I told Mary that she should keep someone else quiet. In the analyzer, however, this second type of ambiguity is subsumed under the first type. This is because the syntactic processes that were incorporated were those performed by programs attached to words and concepts. Thus, in making semantic decisions, i.e. choosing the meanings of words, the analyzer is simultaneously making syntactic decisions, because part of the meaning of a word is the role the word plays in the sentence.

There are a number of questions that make up the problem of ambiguity: foremost, of course, is the question of what the processes are that produce (which is not necessarily a process of choosing from a set of possibilities) the most reasonable, in human terms, interpretation of a sentence. Related to this is the question of whether ambiguities should be seen by an analyzer and resolved, or whether they should not be seen by the analyzer at all. If the former approach is taken, there is another question about whether the resolution of the ambiguity should be done immediately or at the end of the clause or sentence or paragraph.

The analyzer makes no claim to having answered the first question. But specific types of processes that do affect decisions about word meanings will be described.

As to the questions of when, if ever, ambiguities are seen ar . resolved, the analyzer takes a mixed approach. The key is the distinction between a sense of the word, as the term is used in the analyzer, and what intuitively, would be called the meaning of that word. Only one sense of a word is normally seen by the analyzer, and if more than one are found, one is chosen immediately. However a sense is itself a set of programs each of which can produce a distinct conceptual representation if the right circumstances occur. Thus, even though one sense is chosen for a word, there may still be, so to speak, several meanings available. But note that the circumstances that lead to these different meanings are distinct. One sense does not predict two contradictory actions for the same expected situation. This is true, by fiat, for requests in general. There will not be for a situation, S, two requests each testing for S but causing contradictory actions if S is found. In the special case, where the situation S is the occurrence of a single word, this is saying no more than that a word has only one sense expected at a time. The principle is extended, then, to cover also those situations where S is not a word.

CHAPTER 5

CONCEPTUAL DEPENDENCY

5.1 GENERAL ASPECTS

Conceptual Dependency as a term applies to several different aspects of work that has been carried on for the past four years. The least crucial of these aspects is the notation involved, but it is that which, of necessity, the most space must be devoted for description. For perspective however, there should also be a brief mention of the other things that make up Conceptual Dependency. They are:

1) A claim--The claim is that an interlingua should be a system of structures suitable for the representation of language-independent concepts. Further, the understanding of language means the association of such structures with linguistic ones.
2) A methodology--The methodology (by no means well-defined) makes heavy use of introspection about what simple natural language utterances mean. Then there is a search for primitive elements and relationships, and ways of combining the two, that capture both what a single phrase means, and also what similarities and differences in meaning the phrase has when compared with other phrases. The paper by Schank et al. (1972) is a good example of the results of this kind of approach.
3) A system of primitives--In CD it is assumed that the interlingual structures are built with elements from a well-defined set of building blocks, elements

which are themselves not reducible to other structures. This is opposed to a system where all elements are defined in terms of other elements, such as Quillian (1968). These primitives are of two types--relationships and entities.

4) A notational system--The description of this follows. It does not describe all of CD but rather just enough to explain the structures that are used in this work. Alternative descriptions exist in Schank (1972), Goldman (1974), and Rieger (1974), but the reader is warned that he will encounter minor notational variants between those descriptions and this one.

The Two Formats

There are two formats in which CD structures will be written. One is the two-dimensional graph format that is the standard form used in the papers that have appeared on CD. The other is a one-dimensional list structure format that is used by various programs for the input and output of CD structures. These two formats will be given simultaneously. However, as a guide to the reader in parsing the one-dimensional Lisp form, the following BNF description can be given:

```
⟨CONCEPTUALIZATION⟩:= (⟨FORM⟩ ⟨MODIFIERS⟩)
⟨FORM⟩:= (⟨MODIFIERS⟩) | LISP-ATOM
⟨MODIFIERS⟩:= empty |⟨MODIFIERS⟩ LISP-ATOM ⟨CONCEPTU-
                                             ALIZATION⟩
```

The basic element is the ⟨CONCEPTUALIZATION⟩. It has two parts, a ⟨FORM⟩ and a string of ⟨MODIFIERS⟩. The ⟨FORM⟩ is either an atom or else a list of the same form as ⟨MODIFIERS⟩. This latter form consists of an even number of elements. The first, third, fifth, etc. elements are atoms and specify various one argument roles. The second, fourth, sixth, etc. elements are ⟨CONCEPTUALIZATION⟩s and specify the fillers of the roles that precede them. The roles that can appear in a ⟨FORM⟩ are distinct from those that can appear in a ⟨MODIFIERS⟩ (unless, of course, the ⟨FORM⟩ is in a ⟨CONCEPTUALIZATION⟩ in a ⟨MODIFIERS⟩).

A syntax in BNF cannot be written of the CD graph structures because of its two-dimensional format. Basically two kinds of links are used to express the relationships that are handled by the roles in the list format. They are:

1) Arrows--An arrow has at least one head and at least one tail, and the elements in a graph next to the head(s) and tail(s) of an arrow are arguments of the relatiohship associated with that arrow. The exceptions to nextness are the $\xleftarrow{o}$, $\xleftarrow{I}$, $\xleftarrow{R}\left[\begin{smallmatrix}\rightarrow\\ \leftarrow\end{smallmatrix}\right.$ and $\xleftarrow{D}\left[\begin{smallmatrix}\rightarrow\\ \leftarrow\end{smallmatrix}\right.$ arrows. When their left-hand heads all point to the same object, these arrows are written along a straight line for readability, and their left-hand heads point "through" the other elements. For example in:

$$\begin{tikzcd} & & & \text{JOHN} \\ \text{ATRANS} & \text{BOOK} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\ & & & \text{MARY} \arrow[ul] \end{tikzcd}$$

both links have ATRANS as the argument of their left-hand arrow head.

2) Parentheses--A few relationships are specified by writing the two arguments adjacent to each other, with the second one in parentheses, e.g. HAND (JOHN).

5.2 CD ELEMENTS

A conceptualization is the most complex CD structure possible. It is built using elements of three types:

1) Primitives
2) Relationships
3) Conceptualizations

A relationship links one or more primitives or conceptualizations, called the dependents, with one primitive or conceptualization, called the governor. (There are several relationships that are several links combined and those have more than one governor.) Every relationship is fixed in the number of dependents it has (with the exception of a link used with the action MBUILD, which will not be described here), and is fixed also in which of its arguments must be conceptualizations and which must be primitives. (There are two relationships, $\Longleftrightarrow$ and $\overset{o}{\longleftarrow}$, which allow either for one of their arguments.)

Special Primitives and Relationships

The set of primitives is not a closed set (see "Conceptual Semantics"), but there are several important closed subsets. One of these is the set of basic ACTS, or actions. There are about a dozen of these, and the ones which shall be needed here are: ATRANS, PTRANS, MTRANS, PROPEL, GRASP, MOVE and INGEST. Their usages will be described shortly.

Another subset is important for conceptualizations dealing with mental activity. This is a subset of the PPs, or things. They describe body parts capable of containing information, i.e. conceptualizations. These parts are:

1) Brain
   i) CP-for Conscious Processor--This is the part of the brain that carries on conscious

thought;

ii) IM-for Immediate Memory--This is the part of the brain that carries on subconscious thought;

iii) LTM-for Long Term Memory--This is the part of the brain where knowledge is stored;

2) Sense Organs

i) EYES

ii) EARS

More parts and details about their functions appear in Schank et al. (1972).

The atom MLOC (for mental location) followed by one of these parts in parentheses is used in state descriptions for predicating that information is present in that part. The person to whom this part belongs is specified by placing him in parentheses following the part. Thus the graphic structure MLOC (LTM (JOHN)) refers to the mental location of John's Long Term Memory. The linear form of this is (MLOC VAL (LTM PART (JOHN))).

Three other, non-mental, body parts that shall be used are: HAND, MOUTH, NECK and INSIDE. The last refers variously to the lungs, the stomach, or the blood stream, depending on whether something was put INSIDE someone by breathing, eating or injecting, respectively.

It is important to note, particularly with respect to the brain, that these body parts (and the emotional states described next) are not meant to be correct descriptions of the way people are built. They are instead intended to reflect a naive everyday picture of how people are built, a view as it is reflected in the common vocabulary.

The emotional scales are another subset of the set of primitives. A particular emotional state is given by a point on an emotional scale. The only scales used here are JOY and

ANGER. The scale for the former runs from -10 to 10. The scale for the latter runs from 0 to -10. A mild feeling of JOY is about 2 on the scale and is written graphically as JOY (+2). A mild feeling of ANGER is written graphically as ANGER (-2). The linear forms for these constructions are (JOY VAL (+2)) and (ANGER VAL (-2)) respectively.

Another scale is HEALTH which describes the general healthiness (or lack of it) of a person. The scale runs from -10 (by convention this is used for death) to +10.

5.3 CD STRUCTURES

We now describe the structure of conceptualizations. Conceptualizations play a central role in the representation of thoughts. A conceptualization is used to represent a belief about either a state or an event. An event is either a change in state, an action done by an actor, a conjunction of one or more events, or the causing of one event by another.

In the examples of conceptualizations given below, the graphic forms precede the linear ones.

States

A state is a predication about a static condition of something. This thing might be either a simple object or a conceptualization. (There is no predication that can apply to both types.) The predication involves three elements: the thing, T, predicated about, a property, P, and a value, V, of that property when applied to T. The CD structure used is:

T⟺P (V)   or ((ACTOR (T)⟺(P VAL (V))))
            or ((CON (T)⟺(P VAL (V))))

ACTOR is used in the linear format if T is a PP and CON is used if T is a conceptualization.

An example of a property is COLOR. The structure underlying the sentence "The book is red" is:

BOOK⟺COLOR (RED)   or   ((ACTOR (BOOK)⟺(COLOR VAL(RED))))

For "John is in New York" we write:

JOHN⟺LOC (NEW YORK) or

((ACTOR (JOHN)⟺(LOC VAL (NEW_YORK))))

For "Bill has the knowledge (or knows) that John is in New York" we write:

JOHN
⇕⟺MLOC (LTM (BILL))
LOC (NEW YORK)

or ((CON ((ACTOR (JOHN)⇔(LOC VAL (NEW_YORK))))
⇔ (MLOC VAL (LTM PART (BILL)))))

"John is furious" is written as a point on the ANGER scale:

JOHN⇔ANGER (-8) or ((ACTOR (JOHN)⇔(ANGER VAL (-8))))

<u>Events</u>

A) <u>State Changes</u>

A state change is the transition by something from one state to another, or more specifically, from one value of a property to another value of that property. The basic structure used for a change from value V_1 to value V_2 is:

$$\begin{tikzcd} & P\ (V_2) \\ T \arrow[ur, Leftrightarrow] \arrow[dr, Leftrightarrow] & \\ & P\ (V_1) \end{tikzcd}$$

or ((ACTOR (T)⇔T (P VAL (V_2))
⇔F (P VAL (V_1))))

or ((CON (T)⇔T (P VAL (V_2))
⇔F (P VAL (V_1))))

Thus, to represent a change of ownership of a book from Bill to John (but <u>not</u> the action of changing ownership) we write:

$$\begin{tikzcd} & \text{POSS (JOHN)} \\ \text{BOOK} \arrow[ur, Leftrightarrow] \arrow[dr, Leftrightarrow] & \\ & \text{POSS (BILL)} \end{tikzcd}$$

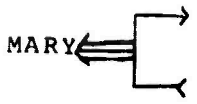
or ((ACTOR (BOOK)⇔T (POSS VAL (JOHN))
⇔F (POSS VAL (BILL))))

For "John became angrier" we write:

$$\begin{tikzcd} & \text{ANGRY (X+2)} \\ \text{JOHN} \arrow[ur, Leftrightarrow] \arrow[dr, Leftrightarrow] & \\ & \text{ANGRY (X)} \end{tikzcd}$$

or ((ACTOR (JOHN)⇔T (ANGER)
⇔F (ANGER)) INC (+2))

With state changes on scales it is also possible to make use of the convention that a positive scale, like JOY, goes to +10, while a negative scale, like FEAR or ANGER, goes to -10. Hence we can specify a positive or negative state change without designating a particular Scale. Thus to represent "Mary's situation worsened" we write:

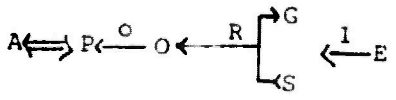


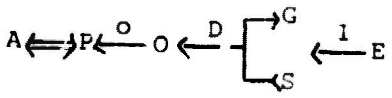
- or ((ACTOR (MARY)$\Longleftrightarrow$T (UNSPEC)
$\Longleftrightarrow$F (UNSPEC))
INC (-))

B) Actions

An action is someone doing something, usually involving some other thing, some directic of the action and some sub-actions that tell how the action was done. The graphic versions for these relationships are $\Longleftrightarrow$, $\xleftarrow{O}$, $\xleftarrow{R}$[(or $\xleftarrow{D}$[ ) and $\xleftarrow{I}$, respectively. The linear versions are <==>, OBJECT, TO and FROM and INST, respectively.

The basic structure for an actor A doing an action P to some object O, in a direction from a source S to a goal G, by means of the event E is:



or 

or ((ACTOR (A)$\Longleftrightarrow$(P) OBJECT (O) TO (G) FROM (S)
INST (E)))

A, P, G and S are always atoms, i.e. primitives. An E is always a conceptualization. An O is always a primitive for some actions and a conceptualization for others. (In the linear form MOBJECT is used when O is a conceptualization.) An A is always either an animate being (and with some actions A must be human) or a physical force, such as gravity or a machine. A rock, in other words, could not be an A. The INST is the one case which may be left unspecified. Since every action can have instruments, which in turn can have instruments, the decision of when not to specify the INST depends on how deeply something is being considered, not on the syntax of full conceptualizations. When other cases are not filled in, their links are written but in the place where the information should be a ▭ is put in the graphs and a NIL in the lists.

There are seven primitive actions that will be used here, plus two special ones. The actions are:

1) ATRANS-takes the R (rather than D) case (R is for recipient)--This is the transfer of owner-ship of O (which must be a PP) from person S to person G. Usually A is equal to either S or G. For example, "John gave Mary the book" is:

$$\begin{tikzcd}
 & & & & \text{MARY} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{BOOK} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

or

((ACTOR (JOHN) ⟺(ATRANS) OBJECT (BOOK) TO (MARY) FROM (JOHN)))

2) PTRANS-takes the D (for directive) case--This is the transfer of control of (or vicinity of) O (which is a PP) from S to G (both of which must be PPs). For "John passed Mary the book" we write:

$$\begin{tikzcd}
 & & & & \text{MARY} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{PTRANS} & \text{BOOK} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

or

((ACTOR (JOHN)⟺(PTRANS) OBJECT (BOOK) TO (MARY) FROM (JOHN)))

We also use PTRANS for simple motion, such as "John went to New York" which is written as:

$$\begin{tikzcd}
 & & & & \text{NEW YORK} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{PTRANS} & \text{JOHN} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[ur] & \\
 & & & & \square \arrow[ul]
\end{tikzcd}$$

or

((ACTOR (JOHN)⟺(PTRANS) OBJECT (JOHN) FROM (NIL) TO (NEW_YORK)))

3) MTRANS-takes the R case--This is the transfer of the information represented by O (which must be a conceptualization) from S to G (which must both be of the set of mental locations listed

in "Special Primitives and Relationships").
This is a multi-purpose action, covering basically all kinds of transfers of information. For example, for "John told Mary that Bill was here" we write:

$$\begin{tikzcd}
 & & \text{BILL} \arrow[dd, Leftrightarrow] & & \text{CP (MARY)} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{MTRANS} & {} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & \text{LOC (HERE)} & & \text{CP (JOHN)} \arrow[ul]
\end{tikzcd}$$

or

```
((ACTOR (JOHN)<=> (MTRANS)
  MOBJECT ((ACTOR (BILL)<=>(LOC VAL (HERE))))
  TO (CP PART (MARY)) FROM (CP PART (JOHN))))
```

We also use MTRANS for internal information transfer. Thus we write for "John recalled that Bill was here":

$$\begin{tikzcd}
 & & \text{BILL} \arrow[dd, Leftrightarrow] & & \text{CP (JOHN)} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{MTRANS} & {} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & \text{LOC (HERE)} & & \text{LTM (JOHN)} \arrow[ul]
\end{tikzcd}$$

or

```
((ACTOR (JOHN)<=> (MTRANS)
  MOBJECT ((ACTOR (BILL)<=>(LOC VAL (HERE))))
  TO (CP PART (JOHN)) FROM (LTM PART (JOHN))))
```

And MTRANS also serves for perception, so that "John saw Bill was here" is represented as:

$$\begin{tikzcd}
 & & \text{BILL} \arrow[dd, Leftrightarrow] & & \text{CP (JOHN)} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{MTRANS} & {} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & \text{LOC (HERE)} & & \text{EYES (JOHN)} \arrow[ul]
\end{tikzcd}$$

or

```
((ACTOR (JOHN)<=>(MTRANS)
  MOBJECT ((ACTOR (BILL)<=>(LOC VAL (HERE))))
  TO (CP PART (JOHN)) FROM (EYES PART (JOHN))))
```

4) PROPEL-takes the D case--This is the application of a force to O (which must be a PP) in a direction towards G away from S (which must both be PPs). To represent the meaning of "John pushed

Mary" we write:

$$\begin{tikzcd} & & & & \square \\ JOHN \arrow[r, Leftrightarrow] & PROPEL & MARY \arrow[l, "O"'] & \cdot \arrow[l, "D"'] \arrow[ur] & \\ & & & & JOHN \arrow[ul] \end{tikzcd}$$

or

((ACTOR (JOHN)⟺(PROPEL) OBJECT (MARY) TO (NIL)
FROM (JOHN)))

5) GRASP-takes neither R nor D--This is the action of taking hold of O (which must be a PP). Thus to represent "John grabbed the block" we write:

$$\begin{tikzcd} JOHN \arrow[r, Leftrightarrow] & GRASP & BLOCK \arrow[l, "O"'] \end{tikzcd}$$

or

((ACTOR (JOHN)⟺(GRASP) OBJECT (BLOCK)))

6) INGEST-takes the R case--This is the action of ingesting O (which must be a PP). The normal use of this act is to represent "eating" and "drinking", using INSIDE and MOUTH. Thus to represent "John ate a banana" we write:

$$\begin{tikzcd} & & & & INSIDE\ (JOHN) \\ JOHN \arrow[r, Leftrightarrow] & INGEST & BANANA \arrow[l, "O"'] & \cdot \arrow[l, "R"'] \arrow[ur] & \\ & & & & MOUTH\ (JOHN) \arrow[ul] \end{tikzcd}$$

or

((ACTOR (JOHN)⟺(INGEST)
OBJECT (BANANA) TO (INSIDE PART (JOHN))
FROM (MOUTH PART (JOHN))))

7) MOVE-takes the D case--This is the action of someone moving the body part O (which must belong to the set of body parts, naturally). This is commonly used for specifying the INST of physical actions. For example to represent "John hit Mary with his hand" we write:

$$\begin{tikzcd} & & & & \square & & & JOHN \arrow[d, Leftrightarrow] \arrow[ll, "I"'] & \\ JOHN \arrow[r, Leftrightarrow] & PROPEL & MARY \arrow[l, "O"'] & \cdot \arrow[l, "D"'] \arrow[ur] & & & & MOVE & \\ & & & & JOHN \arrow[ul] & & & HAND\ (JOHN) \arrow[u, "O"] & \\ & & & & & & & \cdot \arrow[u, "D"] \arrow[dr] & \\ & & & & & & JOHN \arrow[ur] & & MARY \end{tikzcd}$$

or

```
((ACTOR (JOHN)⟺(PROPEL) OBJECT (MARY)
  TO (NIL) FROM (JOHN)
  INST ((ACTOR (JOHN)⟺(MOVE) OBJECT (HAND PART (JOHN))
                             TO (MARY) FROM (JOHN)))))
```

8) DO--This is for representing cases where it is known that someone performed an action, but the action is left unspecified. Thus to represent "John did something to this book" we write:

$\text{JOHN} \Longleftrightarrow \text{DO} \xleftarrow{o} \text{BOOK}$

or

```
((ACTOR (JOHN)⟺(DO) OBJECT (BOOK)))
```

9) MFEEL--This is an <u>ad hoc</u> notational device convenient for an example that appears in the section "Multi-Sentence Analysis". It is the "action" of feeling some emotion O (which must belong to the list of emotions) towards person G. It is used here to represent "John hated Mary" as:

$$\begin{tikzcd}
 & & & \text{MARY} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{MFEEL} & \text{ANGER} \arrow[l, "o"'] & \arrow[l] \arrow[u, "R"] \arrow[d] \\
 & & & \text{JOHN}
\end{tikzcd}$$

or

```
((ACTOR (JOHN)⟺(MFEEL) OBJECT (ANGER) TO (MARY)
                        FROM (JOHN)))
```

C) <u>Conjunction of Events</u>

The description of two events as one event is necessary sometimes. This happens when two events together specify one instrumental event, or when the two events together specify one of the events in a causal relationship, or when the two events are specified by one utterance. The general form for representing the conjunction of events E_1 and E_2 (both conceptualizations) is:

$E_1 \wedge E_2$ or $((\text{CON } (E_1) \wedge (E_2)))$

Thus to represent "John and Mary are here" we write:

JOHN ⟺ LOC (HERE) ∧ MARY ⟺ LOC (HERE)

or

```
((CON ((ACTOR (JOHN)⟺(LOC VAL (HERE))))
  ∧   ((ACTOR (MARY)⟺(LOC VAL (HERE)))) ))
```

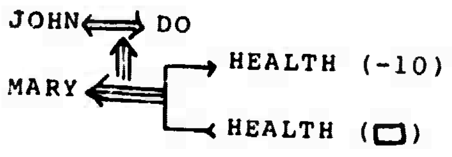
D) <u>Causation</u>

Causation is a relationship between two events that says that one event, the <u>antecedent</u>, is in some way responsible for another event, the <u>effect</u>. What kinds of causation there are is a matter of current study. In this work three types of causality are referred to. They are:

1) Simple causation--The occurrence of event E_1 led to the occurrence of event E_2. The general form for this is:

$$\begin{tikzcd} E_1 \\ E_2 \arrow[u, Rightarrow] \end{tikzcd}$$ or ((CON (E_1)⟸ (E_2)))

The representation of "John killed Mary", is:



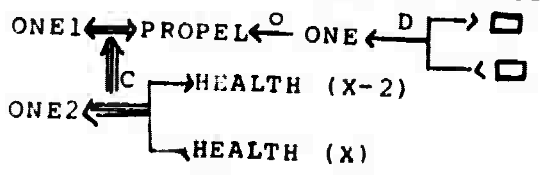
or

```
((CON ((ACTOR (JOHN)⟺(DO)))
  ⟸ ((ACTOR (MARY)⟺T (HEALTH (-10))
                  ⟺F (HEALTH (NIL)))) ))
```

2) Conditional Causation--The occurrence of E_1 usually leads to the occurrence of E_2. The general form is:

$$\begin{tikzcd} E_1 \\ E_2 \arrow[u, Rightarrow, "C"'] \end{tikzcd}$$ or ((CON (E_1)⟸C (E_2)))

To represent "Being hit hurts" we write:



or

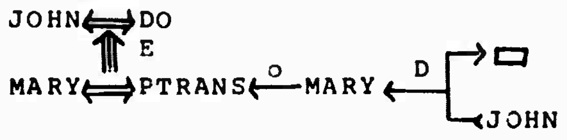
```
((CON ((ACTOR (ONE1)⟺(PROPEL) OBJECT (ONE2)TO (NIL)
                                           FROM (NIL)))
 ⇐C ((ACTOR (ONE2)⟺T (HEALTH)⟺F (HEALTH))
                                          INC (-2)) ))
```

where ONE1 and ONE2 are dummies representing unknown people.

3) Enabling causation--The occurrence of E_1 allows the occurrence of E_2. The general form is:

$$\begin{tikzcd} E_1 \\ E_2 \arrow[u, Rightarrow, "E"'] \end{tikzcd}$$ or ((CON (E_1)⇐E (E_2)))

To represent "John allowed Mary to go" we write:



or

```
((CON ((ACTOR (JOHN)⟺(DO)))
 ⇐E ((ACTOR (MARY)⟺(PTRANS) OBJECT (MARY) TO (NIL)
                                 FROM (JOHN))) ))
```

<u>Modifiers</u>

The graphs given so far have been omitting certain modifying relationships that are normally included. There is one general modifying link for both primitives and conceptualizations, several links specific to conceptualizations, and one pseudo-link for primitives only.

A) <u>General Modifying Link</u>

The general modifying link is the relative relationship. A primitive or conceptualization O is modified by a conceptualization C in which O must appear. The general form is:

$$\begin{tikzcd} O \arrow[d, leftrightarrow] \\ C \end{tikzcd}$$ or (O⟷(C))

For example, to represent "The red book is here" we write:

$$\begin{tikzcd} & \text{BOOK} \arrow[r, Leftrightarrow] \arrow[d, leftrightarrow] & \text{LOC (HERE)} \\ \text{BOOK} \arrow[r, Leftrightarrow] & {} \arrow[r, phantom] & \text{COLOR (RED)} \end{tikzcd}$$

or

((ACTOR (BOOK⟷((ACTOR (BOOK⟺(COLOR VAL (RED)))))
  ⟺(LOC VAL (HERE))))

To represent "John's book is here" we write:

$$\begin{tikzcd} & \text{BOOK} \arrow[r, equal] \arrow[d, leftrightarrow] & \text{LOC (HERE)} \\ \text{BOOK} \arrow[r, Leftrightarrow] & {} \arrow[r, phantom] & \text{POSS (JOHN)} \end{tikzcd}$$

or

((ACTOR (BOOK⟷(ACTOR (BOOK)⟺(POSS VAL (JOHN)))))
  ⟸(LOC VAL (HERE))))

B) <u>Conceptualization Modifiers</u>

The two main conceptualization modifiers are called TIME and MODE in the linear format. TIME is the time at which the conceptualization was true. The MODE is used to specify either that the conceptualization is only a potential one, or that this conceptualization did not occur at all. In the graphs the time modifier is written only if it is not obvious, or if it is referred to by something else. The MODE is given only if the conceptualization is not positive and actual. In the lists, TIME and MODE are always specified.

However, the graphic specifications of TIME and MODE are substantially simpler than those of the list format. Therefore they shall be described first and then the list format system will be given separately.

i) <u>Graphic Format for Times and Modes</u>

The time of a conceptualization is written above the main link of that conceptualization. A simple specification is to say that some action occurred in the past. For example, to represent "John pushed Mary" we write:

$$\begin{tikzcd} \text{JOHN} \arrow[r, Leftrightarrow, "\text{past}"] & \text{PROPEL} & \text{MARY} \arrow[l, "o"'] \end{tikzcd}$$

Sometimes we want to compare the times of two events. To do this we use a special comparative construction (a coherent system for representing comparatives in CD structures has not yet been developed), and place names where the time modifications should be. To represent "John was here before Mary" we write:

$$\begin{tikzcd} \text{JOHN} \arrow[r, Leftrightarrow, "t_1"] & \text{LOC (HERE)} \\ {} \arrow[u, phantom, "\wedge"] \\ \text{MARY} \arrow[r, Leftrightarrow, "t_2"] & \text{LOC (HERE)} \\ {} \arrow[u, phantom, "\wedge"] \end{tikzcd}$$

$$t_1 < t_2$$

The mode specifications are also placed above (or below, depending on how crowded the graph is) the main link. To represent that an action is possible we use a "c". Thus to represent "Mary can leave" we write:

$$\begin{tikzcd} \text{MARY} \arrow[r, Leftrightarrow, "c"] & \text{PTRANS} & \text{MARY} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[r] \arrow[d, dash] & \square \\ & & & {} & \text{HERE} \arrow[l] \end{tikzcd}$$

To represent an impossible action we use a "¢". Thus to represent "Mary cannot leave" we write:

$$\begin{tikzcd} \text{MARY} \arrow[r, Leftrightarrow, "\cancel{c}"] & \text{PTRANS} & \text{MARY} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[r] \arrow[d, dash] & \square \\ & & & {} & \text{HERE} \arrow[l] \end{tikzcd}$$

To represent negation we use a "/" through the link. Thus to represent "Mary didn't leave" we write:

$$\begin{tikzcd}
MARY \arrow[r, Leftrightarrow, "/" description] & PTRANS & MARY \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[r] & \square \\
 & & & & HERE \arrow[ul]
\end{tikzcd}$$

ii) <u>Linear Format for Times and Modes</u>

The representation for times and modes is done by using the roles TIME and MODE. In the BNF syntax that was given, if the conceptualization is in ⟨CONCEPTUALIZATION⟩ form, then these two roles appear in the ⟨MODIFIERS⟩ string.

MODE is followed by a filler which is a list of mode specifications. This filler is the only exception to the BNF syntax since it is just a simple list. The possible entries on this list are CAN or CANNOT, ?, and/or NEG (for negation). Thus the three sentences, "Mary can leave", "Mary cannot leave", and "Mary didn't leave" are represented by the form:

```
((ACTOR (MARY)⟺(PTRANS) OBJECT (MARY) TO (NIL)
                         FROM (HERE)) MODE X)
```

where X is (CAN), (CANNOT), and (NEG) respectively.

Times are a bit more complicated than what has been described. In the list structures representing CD graphs, the TIME atom (link) is followed by (TIMnn) where <u>nn</u> is a number from ØØ to 99. This atom TIMnn has a Lisp value which is a list of temporal relationships that this time has to other points in time. These relationships are:

1) (VAL X) -TIMnn is the same time as X.

2) (BEFORE X y)-TIMnn is before time X by

amount y.

3) (AFTER X y) -TIMnn is after time X by amount y.

No thought has been spent on systematizing concrete specifications of time, such as "Wednesday morning 3:00 A.M.", "an hour from now", and "four score and seven years ago". The only values X takes on, at the moment, in the analyzer, are T, for "now", i.e. the time of the utterance, and other time atoms of the form TIMnn. The only value y takes on is "X" which means a positive but unspecified amount. Thus, as an example, TIMØ2 might have the value:

```
((AFTER TIMØØ X) (BEFORE TIMØ1 X))
where TIMEØ1=((AFTER TIMØØ))
and TIMØØ=((VAL T))
```

Note that the amount X is not assumed to be the same between any of its occurrences. TIMØ2, by the above, is some time between now, T, and a future time, TIMØ1.

iii) Pseudo-Modifier and Special Cases

The analyzer makes use of a pseudo-modifier to pass information to the memory. This modifier is called REF and appears only in the list format. The general form is:

```
(P REF (D))
```

where P is a primitive and D is either A or THE. The analyzer does not choose tokens to be the referents of various primitives. That is, it says the sentence is about a book, but it doesn't say which particular book. Part of the information that memory needs in choosing a token is whether the item is supposed to be a new one or a previously mentioned one. "A" indicates the first case and "THE" indicates the second. The representation of "John has a book" is this:

((ACTOR (BOOK REF (A))$\Leftrightarrow$(POSS (JOHN))))

while for "John has the book" it is:

((ACTOR (BOOK REF (THE))$\Leftrightarrow$(POSS (JOHN))))

Besides REF, there are several special modifiers that appear in this work that are not part of a general scheme. They are placed under the role MANNER, which appears in the same places that modes and times do.

The two values of MANNER which are needed are FAST and REPEATEDLY. The former applies to actions of physical motion, such as PTRANS, PROPEL, and MOVE. To represent "John ran" we write (ignoring the instrumental):

$$\begin{tikzcd}
 & & & & \square \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{PTRANS} & \text{JOHN} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[ur] & \\
\text{FAST} \arrow[u, "\text{MANNER}"'] & & & & \square \arrow[ul]
\end{tikzcd}$$

or

((ACTOR (JOHN)$\Leftrightarrow$(PTRANS) OBJECT (JOHN)
 TO (NIL) FROM (NIL)) MANNER (FAST))

REPEATEDLY applies to actions and means that the action occurred repeatedly. Thus to represent "John beat Mary" we write (ignoring the instrumental):

$$\begin{tikzcd}
 & & & & \square \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{PROPEL} & \text{MARY} \arrow[l, "o"'] & {} \arrow[l, "D"'] \arrow[ur] & \\
\text{REPEATEDLY} \arrow[u, "\text{MANNER}"'] & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

or

```
((ACTOR (JOHN)⟺(PROPEL) OBJECT (MARY) TO (NIL
  FROM (NIL)) MANNER (REPEATEDLY))
```

<u>Special Aspects of the Linear Format</u>

The linear format has two aspects about it which are not shared with the graphic format. First, there is the concept of a linear path through a conceptualization. Second, there is the capability of having several different places in a conceptualization point to just one substructure.

i) <u>Paths in the List Format</u>

Several of the functions in the analyzer use the notion of a path through a conceptual structure. A path is simply a list of Lisp-atoms, each of which is the name of a CD link. The path through a particular conceptualization is followed by the algorithm:

1) If the path list is empty, then the current conceptualization is the answer, else take the first element of the path list.

2) This element should appear either in the list of atoms in the MODIFIERS of this conceptualization, or in the list of atoms in the FORM. The semantics of the structures are such that it cannot appear in both. If it is in neither, then an error has occurred, and the path-following is aborted.

3) Reset the current conceptualization to be the conceptualization immediately following the appearance of the first path element that was found in step 2.

4) Go to step 1.

Thus, in the structure:

```
((CON ((ACTOR (JOHN)⟺(DO)) TIME (TIMØ1))
   ⟸ ((ACTOR (MARY)⟺(PTRANS) OBJECT (MARY))
                                     TIME (TIMØ2))))
```

the path (CON ACTOR) would lead to (JOHN) while the path (⟸ ⟺) would lead to (PTRANS).

ii) <u>Identities in the Graphs</u>

Another feature of the notation, and one which cannot be seen from normal printed output, is the sharing of common nodes within the structures. It is possible, for example, that, in:

```
((CON ((ACTOR (JOHN)⟺(DO)) TIME (TIMØ1))
   ⟸((ACTOR (MARY)⟺(PTRANS) OBJECT (MARY))
                                     TIME (TIMØ2))))
```

the two occurrences of (MARY) will actually involve two references to one occurrence of (MARY). That is, the path (⟸ ACTOR), and the path (⟸ OBJECT) will lead to exactly the same point. If, then, the analyzer follows the first path, (⟸ ACTOR), and there it changes (MARY) to (RITA), the final list structure as printed will be:

```
((CON ((ACTOR (JOHN)⟺(DO)) TIME (TIMØ1))
   ⟸ ((ACTOR (RITA)⟺(PTRANS) OBJECT (RITA))
                                     TIME (TIMØ2))))
```

A function, FIXUP, creates these shared nodes by interpreting Lisp lists in the CD format with the extra pseudo-forms (#X Y Z...)and/or (= X Y Z...). The (X Y Z...) is treated as a path, and a pointer to the end of that path is placed by FIXUP at the point where the pseudo-form would have appeared. # means the path is followed in the structure in a variable called CONCEPT, described later. ≡ means the path is followed in the structure which FIXUP has built so far. The forms (≡X), where X is SUBJ, OBJ, or RECIP, are the same as the pointers returned by (CHOICE X), a function described in the subsection "Conceptual Dependency Graph Manipulators." Thus, with # and ≡ it is possible to build structures where a single pointer to a substructure appears several times.

This brief survey concludes the description of Conceptual Dependency structures. Readers interested in more details about such structures, or in the reasons why these structures have been developed, should read the various papers suggested at the beginning of this chapter.

CHAPTER 6

SENTENTIAL ANALYSIS MECHANISMS

6.1 OVERVIEW OF EXPECTATIONS AND ACTIONS

The analyzer has two parts:

1) A dictionary of words and the expectations associated with them.
2) A monitoring program that keeps track of which expectations have been made and performs the actions associated with an expectation when the condition of the expectation is fulfilled.

The dictionary is not the only source of expectations. Expectations can come from a number of sources during analysis, such as:

A. 1) the words in a sentence;
   2) the concepts referred to by words in the sentence;
   3) the conceptual structure built during the analysis of the sentence;
   4) the clusters of expectations organized about topics, which are concerned with:
      a) the content of the sentences understood, e.g. hunting, contests, or
      b) the environment of the comprehension event, e.g. a joke, a lecture;

An expectation from any of these sources can be about:

B. 1) particular words and their meanings that might be seen next;
   2) particular concepts that might be referred to next;

3) particular conceptual structures that might might be referred to next;

Associated with an expectation is a set of actions that are performed if the expectation is fulfilled. In general these actions can be any kind of behavior available to the entity doing the comprehending, but the only actions relevant here are those that further the process of comprehension.

Such actions are:

C. 1) build conceptual structures from the concepts referred to by the words;

2) build syntactic descriptions of the surface structure of the sentence;

3) add or delete expectations to the set currently active, or modify those already present;

4) modify the actions associated with the current expectations;

The analysis of single sentences involves those expectations referred to by words (A.1 and A.2). These expectations can be about any of the items listed under (B), i.e. words, concepts, or conceptual structures, and the actions about anything under (C). It is this work that shall be described in the most detail. The analysis of sentences in context involves (A.3) and (A.4) and shall be described briefly in section 8.1, "Multi-Sentence Analysis", and in detail in Part II.

The Conceptual Dependency system of representation is the means by which communication is possible between various parts of the analysis. The features of the CD system relevant for discussing the analyzer are:

1) The primitive units and relations are language independent.

2) These elements are intended to represent conceptual information rather than semantic. By this is meant that the structures are created and manipulated not only in language processing but in other deductive mental processing.

3) Central to the system is the actor-action entity. Conceptual Dependency is organized around the concept of people doing things. This is in opposition to the usual predicate calculus state-based systems that have been proposed as conceptual bases.

6.2 ANALYZING WITH EXPECTATIONS

What follows now is a description of a language analysis program based on the use of these language expectations that are associated with the words of the language. When a term like feature or expectation is used, it will refer to some specific piece of the analyzer that has the function of a feature or expectation as described below. We will first describe a number of the routines that have been created to implement the theory described above. Then how they are used in the analysis of sentences will be described.

All the examples are done by the analyzer in the manner described. The vocabulary is limited but, as will be seen, there are non-trivial tasks being performed by the definitions that are present.

6.3 FEATURES AND EXPECTATIONS

Words have associated with them both features and expectations. The features of a word are facts associated either with that word itself or with the concept referred to by that word. That "John" is a proper name is a fact about the word "John". That "John" is a male human is a fact about the concept referred to by the word "John". Features are represented in the system in the CD notation. They are not special flags or marks built specifically for tha analyzer, and though they are used primarily by the analyzer, they are still pieces of the program's world knowledge and are represented like other pieces of world knowledge.

While the features are described with primitives and relationships that are generally used in representing information, the expectations are described with functions that are oriented more towards language processing. The functions that specify conditions and actions are ones that have been found useful for analysis. As our knowledge of memory processes increases, some will remain as they are and others will be generalized to do more than language processing. The functions that have been developed fall into several groups.

6.4 FUNCTIONS IN THE ANALYZER

Conceptual Dependency GraphManipulators

These functions create and change internal counterparts of Conceptual Dependency representations. Graph locations, which can be fully specified by strings of conceptual role markers such as "the actor of the caused event", are holders of information. That is, the graph is both the final analysis result and also an object of many of the expectations that are made while analysis is going on.

A retrieval function, called CHOICE, takes a path as described before, e.g. "($\Leftarrow$ ACTOR)", and returns the conceptual piece found at the end of that path. A storage function, called CHOOSE, follows such a path and puts in a conceptual piece. Both of these functions work with a conceptualization. There exist two related conceptualization builders, REPLACE and IMBED. REPLACE replaces the current conceptual graph (which may be empty) with a new one, that might, but need not, include all or part of the old. This is called, for example, when the verb found in an utterance provides a conceptual network tying together the other elements in the sentence, or when some word, like "again", tells the analyzer that the conceptual network from the verb is a subpart of some other network.

IMBED doesn't change the conceptual graph itself but affects how the above functions behave. Basically when IMBED is called with a path, it resets the conceptualization referenced by CHOICE, CHOOSE, and REPLACE to the conceptual piece indicated by that path. Suppose the analyzer had so far built a network involving the communication of a causal conceptualization, e.g. "advise", which is the communication of the belief that if the person being told does something he will be happier for it. Now IMBED would be called with the argument "(MOBJECT CON)" to reset the conceptualization to be the action, in the communicated idea, which would cause pleasure. Any further work done by CHOOSE, CHOICE, and REPLACE would be in building up this "advised" action. There is of course a function complementary to IMBED called RESET_ALL which resets the conceptualization to be the one before IMBED was called. At the moment there can not be an unlimited stacking of these embeddings and there is a disinclination to allow such. Stacking is a mechanism that can be programmed in a straight forward way, and it has been the basis of many programs for operating on data bases. However, its intuitiveness is questionable. Some kind of mechanism for setting certain processes temporarily aside, however, is certainly needed.

In the analyzer recursion is not a basic mechanism. If the analyzer IMBEDs more than once it will be able to reset only to the most recent embedding or else to the outermost level of the conceptualization. Such an approach is related to the representation we have chosen. Had our system been based on graphs of a more mathematical nature, with a few primitives and a great number of tree structures to represent everything, then embedding would be occurring constantly and the natural way to work with these trees would be with recursive routines. However Conceptual Dependency is oriented about structures where closely related elements of a conceptualization appear together at the same level, where a processor doesn't have to keep looking up and down a tree for information. The focus of manipulation changes much less often in such a representational system. Sometimes, when the analysis leads to a shift in levels, it means that work on the previous level is finished for good. And, in the cases where the level change is only temporary, using only one temporary holding area has been sufficient so far.

As we shall see, often the verb will explicitly provide REPLACE and CHOOSE with the conceptual pieces that it needs. However there are also times when there are significant conceptual structures coming from other words in the sentence. For example, in "John gave Mary a headache," "a headache" is the name of a conceptual structure for feeling pain in the head, and the analyzer, in doing this sentence, needs to take this structure and say "John caused Mary to feel pain in her head." Hence there also exists a routine, called UTILIZE, that takes the structures referred to by words and prepares them for incorporation by REPLACE. Both REPLACE and UTILIZE call the function FIXUP and so they are capable of returning S-expressions with shared nodes.

Finally, there are several functions for manipulating times in the graphs. Two functions, BEFORE and AFTER, each take three arguments. The first two are specifications of points in time and the third is the amount by which the first is before or after the second. This relational information is added to the list of relationships that makes up the value of the first argument. The time atom that was the first argument is then returned as the value of the BEFORE or AFTER. Thus, saying (BEFORE TIMØ2 TIMØ1 X) would add the relation (BEFORE TIMØ1 X) to the value of TIMØ2 and return TIMØ2 as the value of the function call. Frequently the first argument is supposed to be a new time atom. For this a function called NEW_TIME, which is like the Lisp GENSYM, is used which returns a new atom of the form TIMnn whenever it is invoked. Thus when the form (AFTER (NEW_TIME) TIMØ2 X) is evaluated, a new time atom, say TIMØ3, will be created, with a value, (AFTER TIMØ2 X), and TIMØ3 will be the value of the AFTER function call.

Syntactic Structure Manipulators

Another set of functions is needed to operate on the syntactic structure of a sentence. The description of these functions will be somewhat brief. They have not been the main focus of our effort. This is because much work has already been done on syntactic analysis. Most other approaches, computational and linguistic and even psychological, have been concerned with what could be obtained using just syntax, until it became necessary to add semantics. The approach here is the exact opposite, to see what can be done from the conceptual side and only include syntactic aspects when they seem needed. The first form of the analyzer didn't even have word order.

The syntactics used by the analyzer are quite simple. This is partly because less time has been spent on them and partly because the existence of a conceptual network means the syntax doesn't have to carry the semantic load that it does in a syntactically based system.

There are three surface cases used, SUBJ, OBJ, and RECIP, which save places for items until they can be given conceptual roles to play. These roles are primarily determined by word order, with a secondary distinction between humans and objects, so that RECIP is generally a human, if it occurs at all. (The cases are of course specific to English.) The information in these cases is saved by IMBED when it is called and later reset by RESET_ALL, with the same comments about starting applying. Further, CHOICE and CHOOSE both know how to handle these cases, and the analyzer can add and extract information from them just as it does with the conceptual structure it is building.

These word order cases should be supplemented by the use of prepositional markers. This has not yet been implemented in the program beyond one experiment. All that would be done is to save under the name of the preposition the sense of the phrase that it governs, just as the sense of the first noun phrase is saved under the name SUBJ. Nothing conceptual is being done at this point. For the conceptual content of the preposition, the analyzer must decide what relationship a preposition is expressing from what has already been understood and from the nature of the object of the preposition. The verb, which plays a central role in this system, usually does most of the work in giving an expected meaning to the use of a preposition. Still, the analyzer needs to save the fact that such and such item was governed by such and such preposition, particularly to handle prepositions introducing a sentence ("By the car was a...") and to provide backup routines with this information.

There is another place where simple syntactic action occurs: while building noun phrases. Starting with the recognition of an article or adjective, words as they are brought in are not converted into a unified conceptualization until something is seen that indicates the noun phrase is ended. The end of the sentence, a verb, or the start of a new noun phrase are some of these signals. Knowing what the main item is that is being modified by the previous string of adjectives and nouns the analyzer can make a conceptual whole. But many adjectives used commonly, like "short" or "sweet", cannot be said to have meaning until they have something to modify. Granted there may be something in common between "a short stick" and " a short pause", between "a sweet candy" and "a sweet voice", but the common elements involved are too vague to sufficiently determine a particular use of these adjectives. That is, given some such unifying theme, we still couldn't predict reliably what modification the adjective meant with many nouns. Admittedly there are times when we do use rules to generalize word usages, when metaphors are involved, but for the moment we are concerned with the common, ingrained uses of words. Adjectives are fairly ambiguous words, and the major source of information on what to do with them comes last. There is also the complicating factor of noun pairs, such as "kitchen table" and "police state". There exists a program by Sylvia Weber Russell (1972) that handles a number of these, and eventually it may be tied in with the analyzer.

There are two functions for handling noun phrases. One called SAVE takes new words and collects them into a simple list, waiting for the end of the phrase. EVAL_PHRASE, the other, is called when the phrase end is noted and converts this list into a normal conceptual structure. This new structure is then returned as the meaning of the noun phrase and behaves as a unit for such functions as CHOOSE and FEATURE, which is described next.

Memory Interface Fucntions

FEATURE brings us to a probably open-ended set of functions, which interrogate the memory's world knowledge for information about things. These things may be either words or concepts. FEATURE is the only memory interrogation function currently used by the analyzer. It takes as one argument either a word or a simple conceptual piece, i.e. consisting of a PP plus modifying conceptualizations, and as the other argument some property value, such as "HUMAN" or "PROPER" (for proper nouns). These property values belong to contrast sets of distinctive features, such as "(human, animal, physical object)". These contrast sets are needed because there are often times when the analysis depends on which element of the set a particular word or concept is associated with. It is important to note that these contrast sets are anti-hierarchical, at least to some extent. Although being a human implies being an animal which implies being a physical object, the way in which a word is handled in language differs depending on whether it refers to no more than an object or no more than an animal. FEATURE is a very simple information retrieval function. It takes a particular complex of features which has been chosen for some reason--usually because a word referencing the complex was seen--and FEATURE is used to find out what appears in this complex. Thus, if "John" is chosen as referring to "JOHN1" which is "the man called John" sense of "John", FEATURE then can be used to find that "JOHN1" is a man, and that an English name is involved.

Although the function FEATURE is called with a simple pair of arguments, like "JOHN1" and "HUMAN", what it actually looks for is a full conceptualization of the form:

((ACTOR (JOHN1)$\Leftrightarrow$(CLASS VAL (HUMAN))))

Attached to JOHN1 is a list of features and they have the form:

```
          (KEYWORDS FEATURECON)
```

where KEYWORDS is simply a list of elements that appear in FEATURECON, e.g. JOHN1, CLASS, HUMAN. To save time FEATURE checks this list first before doing a pattern match between the conceptualization it has and the conceptualization that makes up FEATURECON. The form of the actual feature conceptualization, FEATURECON, is usually:

$$((\text{ACTOR } (X) \Longleftrightarrow (\text{FCLASS VAL } (F))))$$

where F is a feature like HUMAN and FCLASS is the contrast set to which F belongs. The reason for this representation of simple features was that it allowed features in general to be any conceptualization associated with an object, not just predications about properties of the object. For instance, a feature of a "gift" could be that it is an object which one person originally obtained in order to give to another person.

6.5 CONCEPTUAL SEMANTICS

It should be mentioned at this point that the semantics of nouns in Conceptual Dependency is handled only superficially. The stress in representational work has been on conceptual actions and conceptual relationships, rather than on conceptual objects, i.e. PPs. And the bulk of the work on analysis and generation of English has been concerned with verbs. English nouns that name actions or relationships, like "a beating" or "a walk", are recognized as such, and they are analyzed into conceptual structures involving full conceptualizations. However, nouns that name actual physical things, like "a dog" and "John", are analyzed normally into non-primitive PPs like DOG1 and JOHN1. These are not words, for they do not have the same characteristics that words have, characteristics like ambiguity and morphological composition. These PPs are conceptual and appear in conceptual structures in relationships with acts and other PPs. What is not well developed is how, in memory, PPs relate to each other in terms of meaning, i.e. how does the concept of a chair relate to that of a table, what does it mean to use a cup for a hammer, and so on. What is lacking is a well-defined internal structure for PPs. Presumably a PP is a bundle of features, but how many features there are, how many it takes for an object to qualify as a certain kind of PP, how features relate to each other, how features which are discrete relate to the perception of a world that is not, all these questions are unanswered.

Fortunately it has turned out that it is possible to do a substantial amount of work with only a small amount of concern for the nature of physical objects. Certain features, such as humanness and physicalness, have been enough to allow various programs, including the analyzer, to manipulate conceptual objects, to be able to decide what whould be done with them in a given situation. And since features themselves are expressed in terms of conceptualizations, work on the latter can not help clarifying the nature of the former.

6.6 THE MONITOR

There are other functions in the analyzer, but they are subservient to the ones that have been discussed. Only one more piece of the analyzer needs to be described before some examples are given. This piece is the monitor, or supervisor, the piece that takes definitions of words, which are combinations of these functions, and executes their instructions. This monitor is, and is meant to be, very simple. Its job is to do bookkeeping on the following variables:

SENTENCE	- This is the utterance being analyzed. It is constant throughout the analysis.
WORD	- This is the current word in the sentence that is being looked at. Normally WORD is set to each successive word in SENTENCE, going from left to right.
PLACE	- This is the rest of SENTENCE, from WORD to the end.
SENSE	- This is the current sense that is being worked with. It is usually either the sense of WORD or of the noun phrase of which WORD is the head.
ACTIVE	- This has the value T or NIL. At the start of a sentence ACTIVE has the value T. Whenever ACTIVE has the value T, the requests that are attached to the words the monitor finds are added to the list that is the value of the variable REQUESTS. However, when ACTIVE is set to NIL, which is done by requests attached to words like articles, this addition is inhibited and instead a function SAVE is called. SAVE collects the words that follow, until ACTIVE is reset to T, in preparation for the construction, by a function called EVAL_PHRASE, of a noun phrase.

REQUESTS - This is a list of requests which is unordered (with one exception). The monitor continually rechecks this list to see if changes to WORD, SENSE, CONCEPT, or REQUESTS itself have caused any of the requests to become applicable. The unordered rechecking is meant to be a simulation of a parallel control structure where each request looks to see if it should do anything, independent of tne other requests. The only exception to this concerns those requests that are activated when some phrase or clause ends. For example, in "John wanted Mary...", tne analyzer assumes that "Mary" is beginning a clause about something involving Mary that John wants. If instead the sentence ends here, then a request triggered by the end of the sentence makes a default assumption that the event which John would like is for Mary to come to him. These requests that are called by the end of something are always placed at the end of the request list. This is equivalent to considering them as independent processes that, in being called by the absence rather than the presence of something, wait to make sure that "more real" requests have had their say.

ANSWER - This is the conceptual representation of the meaning of SENTENCE that the analyzer is building. It is the variable whose value is returned by the analyzer.

CONCEPT - This is a pointer to either ANSWER or to some subconceptualization in ANSWER. This points to the place where the building activity is going on at any point in the analysis. Thus it starts off the same as ANSWER but, when an embedded conceptualization is being built, it points to that instead.

Attached to each word that appears in SENTENCE are one or more senses, that is, labels of sets of features and requests. Requests are of the form "(TEST ACTION FLAG)". TEST and ACTION are the crucial elements of a request. TEST is a (Lisp) predicate and ACTION is a (Lisp) function, both built from Lisp functions and those functions that have been described above. When WORD changes, the monitor first checks REQUESTS for instructions, using a function called CONSIDER, adds any requests attached to WORD, then finds the current sense for WORD (setting SENSE equal to it), then checks REQUESTS again, then adds the requests that are part of SENSE to REQUESTS and steps WORD along in SENTENCE. In general, TEST predicates make reference only to CONCEPT and the feature aspects of WORD and SENSE. Checking a request means evaluating the TEST. If TEST is not true, nothing happens and the monitor goes on to the next request. FLAG is a bookkeeping mark. When it is NIL it means the request has not been used yet, while T means that the request has already been used. The only requests whose TESTs are evaluated are those whose FLAGs are NIL.

The requests that were described as being directly attached to the WORD itself, rather than being part of the SENSE, are fulfilling a stopgap role. They are substitutes for the results that should be returned from a morphological analysis of that word. Routines for doing such were not written, however. Instead, the answers, i.e. specifications about matters like tense, were attached directly to individual word forms. A first approximation to a morphology routine for determining the tensed form of a verb has been written by Paul Martin, and it replaced most of these requests in the analyzer, but it will not be described here.

REQUESTS is the source of basically all the actions that occur. It is also the object of some of these actions. There are several ways ACTIONs can change REQUESTS. One is through the function IMBED, used mainly when entering a new clause. IMBED saves the current REQUESTS in another variable and replaces REQUESTS itself with a new set, specified by the third argument of IMBED. RESET_ALL restores REQUESTS to its original value when it is called. From what was said previously it can be seen that IMBED and RESET_ALL work with three information sets: the conceptualization being built (ANSWER and CONCEPT), the syntactic structure being built (SUBJ, OBJ and RECIP), and the expectations being made (REQUESTS). This last manipulation, the storing away temporarily of REQUESTS, is done also by the requests on articles and prepositions. They don't use the full power of IMBED and RESET_ALL however. Rather, a request on an article will save the content of REQUESTS in a variable called ART_INT, and set REQUESTS to be a request looking for an end to the noun phrase, at which point the old value of REQUESTS will be returned. A request or preposition does the same thing but saves REQUESTS on a variable called PREP_INT. Finally, as a way of changing REQUESTS, there is a function for adding new requests to the list, and this is called ADDREQ. In addition to all this, REQUESTS is initialized by the monitor to a request which looks for a noun phrase to be the subject. This is done whenever a new sentence is begun.

The best way to describe how requests are formed from these functions and how requests interact is by examples. This is the content of the next chapter.

CHAPTER 7

EXAMPLES

The first example is a very simple one, to demonstrate some of the basic elements of the analyzer in action. The sentence is "John gave Mary a book."

There are two sets of information associated with the words in a sentence, the requests and the features. The requests are of two types: those attached directly to word forms (the pseudo-morphological requests), and those attached to the senses of words.

Of the first type of request, there is one in this example. It is attached to "gave":

```
        Gave:  (T (CHOOSE TIME (BEFORE (NEW_TIME)
                                (CHOICE TIME) X))
                NIL)
```

The "T" is the TEST, the "(CHOOSE...X") is the ACTION and the "NIL" is the FLAG. This request says that the TIME, that is, the time of the conceptualization being built, should be set to some point before the time presently associated with the conceptualization. None of the other words in this sentence have the first type of request attached.

The second type of requests, those that belong to a more general sense of a word, is found, in this sentence, with the verb and article. The verb "gave" has the sense GIVE1, which contains seven requests. Four of them are:

```
        GIVE1:
    ((FEATURE SENSE (QUOTE HUMAN)) (CHOOSE RECIP SENSE) NIL)
    ((FEATURE SENSE (QUOTE POBJ)) (CHOOSE OBJ SENSE) NIL)
```

```
   (T (DEFPROP TO TO1 CURRENT) NIL)
  ((FEATURE SENSE (QUOTE POBJ))
   (REPLACE CONCEPT
                  (QUOTE ((ACTOR (#SUBJ)<=>(ATRANS)
                          TO (#RECIP) FROM (#SUBJ)
                          OBJECT (#OBJ))
                          TIME (NIL)
                          MODE (NIL))))
       NIL)
```

The first request chooses the RECIP case to be the sense of the first noun phrase following the verb (it must be following, since this request doesn't appear until the verb does) that has the feature of being human. The second request chooses the OBJ case to be the sense of the first noun phrase following the verb that has the feature of being a physical object. Remember that physical objects and humans are disjoint sets. The third request, using the Lisp function DEFPROP, says that the word "to" is to be associated with a particular sense TO1. Since this sense will not be used in the examples I won't include its definition, but basically this sense puts the phrase following into the conceptual TO case of an "ATRANS" conceptualization.

There is an alternative to setting the sense of the word "to" to TO1. We could write a request for GIVE1 that looked for the word "to", and, when it found it, performed the same actions that the sense TO1 does. However, since this use of "to" occurs with many "ATRANS" related verbs, a separate sense is created for "to" so that it can be shared.

The fourth request above assumes that the conceptual structure for the sentence is the linear equivalent of the following graph, if a physical object is seen:

$$\begin{tikzcd}
 & & & & \text{(RECIP)} \\
\text{(SUBJ)} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{(OBJ)} \arrow[l, "O"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{(SUBJ)} \arrow[ul]
\end{tikzcd}$$

That is, the giving is the transfer of some physical object. Assuming this structure, however, does not mean that it has to be kept for the rest of the sentence. In this example, it will indeed be part of the final result but the fifth request on "GIVE1" can overwrite this structure. The fifth request is:

```
     ((NORM_FIT SENSE ((ACTOR ONE1<=>ONE2 OBJECT ONE3)) NIL)
      (INPLACE CONCEPT
                 (UTILIZE (NORMAL_MEANING SENSE)
                          (QUOTE (((ACTOR) CHOICE SUBJ))
                                   ((OBJECT) (CHOICE RECIP))))))
       NIL)
```

NORM_FIT is a function that compares the conceptual structure referred to by the form given by the second argument. In this instance, the TEST is asking if the sense of the word (or noun phrase) currently being read, refers to a simple action. The "ONE1", "ONE2", and "ONE3", are dummy elements that will match any Lisp S-expression the first time they occur. If a dummy occurs again in the pattern it will match only the same S-expression that it did before. INPLACE is like REPLACE except that it does not perform some Lisp pointer manipulation that REPLACE does. UTILIZE, as mentioned, takes the conceptual form, which NORMAL_MEANING finds in the bundle of features and requests that make up a sense, replaces certain elements in the form with other elements, and returns the modified form. In this case, the one element paths, (ACTOR) and (OBJECT), are followed and the choices for SUBJ and RECIP, respectively, are placed at the ends of these paths. This request will not be activated for this example. The sixth and seventh requests are basically like the fifth, but the forms looked for are slightly different. These requests will not be needed for our examples.

The other set of requests in the example belongs to the word "a". There are no pseudo-morphological requests in the example on the word "a" itself but there are requests for its sense, A. These requests are:

```
          A:
     (T (PROG NIL (SETQ HOLD NIL)
                  (SAVE (QUOTE REF) (QUOTE A))) NIL)
     (T (PROG NIL (SETQ ART_INT REQUESTS) (SETQ REQUESTS NIL)
                  (SETQ ACTIVE NIL)) NIL)
     (T (ADDREQ ((PHRASE_BREAK)
                 (PROG NIL (SETQ SENSE (EVAL_PHRASE HOLD))
                           (SETQ PLACE (CONS (QUOTE PERIOD)
                                             PLACE))
                           (SETQ WORD NIL)
                           (SETQ REQUESTS ART_INT)
                           (SETQ ACTIVE T)
                           (CONSIDER)) NIL))
     NIL)
```

The first request initiates the saving of the words that will be used for the noun phrase. The words are put in a variable called HOLD. The SAVE function call puts in HOLD a mark that the phrase was introduced by "a". This information will be needed by the memory in generating a referent for the phrase. The second request stores the current list of requests in ART_INT, empties REQUESTS itself, and sets ACTIVE to NIL. This is so that no further requests will be added by the monitor as it looks at succeeding words. The last request puts one request on the freshly cleared list. This request looks for the end of the phrase. PHRASE_BREAK is a predicate that becomes true when anything other than a noun or an adjective is seen. When this occurs, a number of actions are performed. First, the SENSE to be returned is built by applying the function EVAL_PHRASE to the list of words collected into HOLD. The changes to the values of PLACE and WORD are purely to keep the monitor from being confused and skipping over either the next word in the sentence (for which a dummy word is placed in front of the rest of the sentence) or neglecting to notice the newly constructed SENSE (for which WORD is set to NIL, which signals a change in SENSE to the monitor). The next action resets REQUESTS to its original value. The variable ACTIVE is reset to T. Finally the function CONSIDER is called which causes an immediate checking of REQUESTS to see if the new SENSE satisfies any expectations.

There is one fi[illegible]l request to mention. It is not attached to any word but is assumed by the monitor before the sentence begins. This request is:

```
((OR (NOT (EMPTY CONCEPT))
     (FEATURE SENSE (QUOTE PP))
     (NORMAL_MEANING SENSE))
  (COND ((NOT (EMPTY CONCEPT)) NIL)
        ((FEATURE SENSE (QUOTE PP)) (CHOOSE SUBJ SENSE))
         (T (REPLACE CONCEPT
                    (UTILIZE (NORMAL_MEANING SENSE) NIL))))
NIL))
```

The TEST of this request looks for three possible situations. The first one, which is true if CONCEPT has been given a value, performs the function of removing this request (by activating it to perform a null action) after the processing of the sentence is begun. The second situation is the normal one, where the sentence begins with a noun phrase referring to a simple object (PP in Conceptual Dependency terms). In this case, the noun phrase is saved as the SUBJ. The third situation is where the sentence begins with a noun phrase referring to a conceptualization, e.g., "a beating" or "the trip", in which case this conceptualization is placed directly into CONCEPT. Later requests provided by other words will tell what to do with this conceptualization.

The other set of information contained in a sentence is the set of features associated with the words of the sentence. One feature is common to the senses of all three nouns in this example. It is:

JOHN1, MARY1, BOOK1 substitute for X in:

((ACTOR (X)⟺(CONTYPE VAL (PP))))

That is, all three are things, conceptually.

Two features are in common between JOHN1 and MARY1. They are:

JOHN1, MARY1 substitute for X in:

((ACTOR (X)⟺(CLASS VAL (HUMAN)))) and

((ACTOR (X)⟺(WORDTYPE VAL (NAME))))

The first feature says that "John" and "Mary" have senses that refer to humans. The second feature says that "John" and "Mary" are names, and hence do not require articles preceding them. This could probably be better handled by having a request on "John" and "Mary" return for the value of SENSE the conceptual representation of "the person who is called John (Mary)", but a simple feature is good enough for our purposes.

Finally there is a feature for each noun that does not apply to the other two.

((ACTOR (JOHN1)⟺(SEX VAL (MALE))))

((ACTOR (MARY1)⟺(SEX VAL (FEMALE))))

((ACTOR (BOOK1)⟺(CLASS VAL (POBJ))))

The first two features would be necessary if the analyzer were doing pronominal reference, but are not of importance in any discussion that follows. The last feature is important, and says that a book is a physical object, as opposed to a human or an animal.

With these requests and features described, we can trace the flow of the analysis as the monitor reads the sentence. First the word "John" is seen. The word has no requests of its own to add nor does the sense JOHN1 which is attached to it. However, because JOHN1 has the feature PP it does satisfy the request to which REQUESTS was initialized, and so JOHN1 is chosen as the subject.

Next the word "gave" is read. The request attached to the word "gave" itself is added to the list. Because the TEST of this request is "T", it is executed immediately, causing the TIME of the conceptualization being built (which is empty at the moment) to be set in the past, before the time of utterance. The word "gave" itself satisfies no expectations. The sense GIVE1 also satisfiesno expectations. The requests it has are added to REQUESTS. Their evaluation causes one action to occur, because one request has "T" as a TEST. This triggered request sets the sense of "to" to TO1.

Next the word "Mary" is read. The word "Mary" itself has no features nor requests. The sense MARY1 also has no requests. However, there are features associated with MARY1 and these trigger one request. This is the request that is expecting SENSE to take on a value that has the feature HUMAN. MARY1 has this feature. The triggered request chooses MARY1 to be the RECIP. Because of the initial setting of pointers in CONCEPT this means that MARY1 will also fill in the "TO" slot in CONCEPT, if the ATRANS structure is built.

The next word read is "a". Neither it nor its sense, A, have any features to satisfy the set of expectations still in REQUESTS. Nor does the word "a" have any requests of its own to add. The sense A, however, has several requests, all of which have "T" as their TEST and so are executed immediately. The actions performed start the building of a noun phrase list, set ACTIVE to NIL, save the current set of requests in the variable ART_INT, and replace REQUESTS with one request that looks for the end of the phrase.

The next word read is "book". Because ACTIVE is NIL the only thing that happens is that "book" and BOOK1, its sense, are checked against the one request present on REQUESTS. Neither satisfies the expectation for an end to the phrase and so BOOK1 is placed on the holding list.

The next word read is PERIOD. This is a mark that says the sentence is finished. PERIOD does not have any requests or features of its own. However, it does satisfy the TEST of the request looking for a phrase end. The request is activated and builds the noun phrase "(BOOK1 REF (A))", which is put as the current value of SENSE. It also resets REQUESTS to the value it had before "a" was encountered, sets ACTIVE back to T, and calls for an immediate rechecking of the restored REQUESTS list. The value of SENSE satisfies the two requests looking for a SENSE referring to a physical object. The actions resulting choose "(BOOK1 REF (A))" to be the OBJ, make ATRANS the main act, and place SENSE as the OBJECT of this act. No other requests are activated, no more words are found, and the analysis is finished. The final result is:

```
    ((ACTOR (JOHN1)<=>(ATRANS) TO (MARY1) FROM (JOHN1)
                       OBJECT (BOOK1 REF (A))) TIME (TIMØ1)
                                               MODE (NIL))
```

where TIMØ1 has the Lisp-value "((BEFORE TIMØØ X))", and TIMØØ has the Lisp-value "((VAL T))". The basic graphic form of this is:

$$\begin{tikzcd}
 & & & & \text{MARY} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{BOOK} \arrow[l, "\text{o}"'] & {} \arrow[l, "\text{R}"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

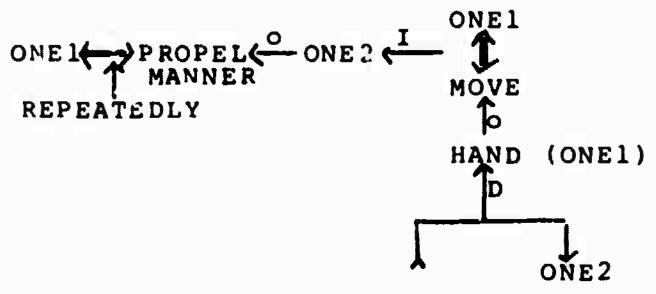
The second example is "John gave Mary a beating". The focus here is on the way a verb like "give" can pull together other elements of the sentence, while itself contributing no conceptual structures. The final result is the same as the result of analyzing "John beat Mary". Manipulating other structures is a very common job for "give" to do and there are other verbs that can function the same wav. For example, "John took a walk" means the same as "John went walking". "John got Mary a job" depends primarily on what "a job" means. "John stole a peek" and "John made a noise" are further examples of verb uses where a large part of the verb's "meaning" is the manipulation of other meanings.

We have already given all the information necessary for describing the analysis of this sentence, except for the definition of the word "beating". "Beating" is a noun, like "book", but it is not a physical object or a PP. "Beating" has only one feature associated with BEAT1, its sense. This is a conceptualization of the form:

```
((R1 (BEAT1) REL (NORMAL_MEANING)
  R2 ((ACTOR (ONE1)⟺(PROPEL) OBJECT (ONE2)
        INST ((ACTOR (≡ACTOR)⟺(MOVE) TO (≡OBJECT)
               OBJECT ((HAND) PART (≡ACTOR)))
               TIME (≡TIME)))
        TIME (NIL) MANNER (REPEATEDLY)) ))
```

The R1, R2 and REL are used to represent relationships in a general fashion, but this is only a tentative representation. REL is a link atom followed by the name of the relationship. R1 and R2 are the first and second arguments of the relationship, respectively. In this case the relationship is called NORMAL_MEANING. Its first argument is a sense name, and its second argument is a conceptual structure. Senses that refer to simple things are not treated this way but should be. That is, where now the analyzer uses the sense name BOOK1 in a graph structure, it should refer to a feature specifying the NORMAL_MEANING for BOOK1. However this lack of consistency is not crucial at this time.

The meaning of "beating" that this feature specifies is graphically:



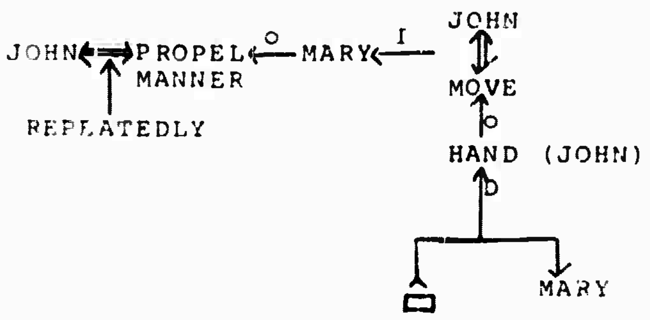
Note that the various occurrences of ONE1 and ONE2 are intended to be references to unique nodes, because of the ≡ constructs that appear in the linear form of this graph.

The analysis of "John gave Mary a beating" proceeds as it did in "John gave Mary a book" until the very last word. When "beating" is discovered, REQUESTS has been saved in ART_INT and reset to a request looking for the end of the noun phrase initiated by "a". The word "beating" is read and accepted as a noun, that is, it is not a noun phrase terminator. It is saved on the holding list and the next word is read. This is "PERIOD" and this does terminate the noun phrase. The noun phrase returned is thus "(BEAT1 REF (A))", where "BEAT1" is the sense of the noun "beating". With the phrase finished, REQUESTS is reset to the value saved on ART_INT, which still has two requests waiting. One is looking for a physical object and the other for the name of an action. In this example, the latter is satisfied. Hence the function INPLACE is called to overwrite any current structure in CONCEPT. CONCEPT is set to the structure that is the NORMAL_MEANING of BEAT1.

The function UTILIZE takes this structure and at the same time takes the current values of SUBJ (which is set to JOHN1) and RECIP (which is set to MARY1) and places them in the positions of ACTOR and OBJ, respectively. The TIME of the new conceptualization is set to that of the old unless otherwise specified. With this done, the sentence is ended, and the final analysis result is:

```
((ACTOR (JOHN1)<==>(PROPEL) OBJECT (MARY1)
   INST ((ACTOR (JOHN1)<==>(MOVE) TO (MARY1)
            OBJECT (HAND PART (JOHN1))) TIME (TIMØ1)))
            TIME (TIMØ1) MANNER (REPEATEDLY))
```

where TIMØ1 has the value "((BEFORE TIMØØ X))" and TIMØØ has the value "((VAL T))". Graphically this is:



A bit more rapidly, we can look at the analysis of a sentence like "John advised Mary to drink the wine." This example shows how the function NMBED works to shift the levels of manipulation. First, though, we must give the requests that are found for the words that appear in the sentence. There is one for each of the two verb forms:

advised:

```
(T (CHOOSE TIME  (BEFORE (NEW_TIME) (CHOICE TIME) X)) NIL)
```

drink:

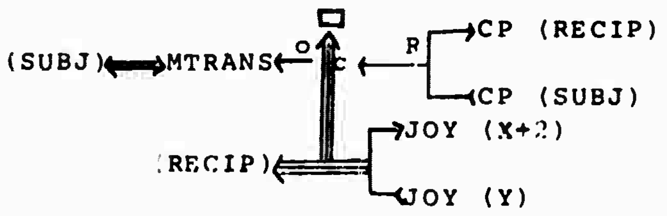
```
((NEED_TIME) (CHOOSE TIME (CHOICE TIME)) NIL)
```

The request for "advised" is the same "past-tense" request used with "gave". The request for "drink" has the predicate (NEED_TIME) instead of a "T" because "drink" may either be a present tense form or a non-tensed infinitive. Normally (NEED_TIME) returns the value T but certain other words have requests that set (NEED_TIME) to return NIL. One of the senses of "to", the one in fact that will be used in this example, has an instance of such a request. If (NEED_TIME) returns T then the action associated with it says that the TIME to use is the one currently set. This request is vacuous in the current system and is a remnant of an earlier version of the analyzer that did not include a default assumption of current time if nothing about TIME was specified.

"Advised" has the sense ADVISE1, which is the following set of requests:

```
ADVISE1:  (T (REPLACE CONCEPT
                 (QUOTE((ACTOR (#SUBJ)⟺(MTRANS)
                 TO (CP PART (#RECIP) REF (THE))
                 FROM (CP PART (#SUBJ) REF (THE)) MOBJECT
                 ((CON (NIL TIME (>) MODE (NIL))⇐C
                 ((ACTOR (#RECIP)⟺T (JOY)⟺F (JOY))
                 INC (2) TIME (≡MOBJECT CON TIME) MODE (NIL)))))
                 MODE (NIL) TIME (NIL)))) NIL)
          (T (DEFPROP TO TOØ CURRENT) NIL)
          (FEATURE SENSE (QUOTE HUMAN))
                 (CHOOSE RECIP SENSE) NIL)
```

The first request produces a conceptual form equivalent to:



This is, in English, expressible as SUBJ telling RECIP that if RECIP does something then RECIP will become happier. Now this is nothing more than a rough approximation to what "advise" means. It is actually a communication of a more general statement that if RECIP does something he will, on some unspecified scale, be better off than if he does not do this action. However the above approximation is good enough for our purposes.

The second request is like the request that gave "to" a meaning in GIVE1 but, as we shall see, TOØ is quite different from TO1. The last request is just like the request to fill RECIP that appeared with "GIVE1".

The verb "drink" also has a set of requests:

```
DRINK1:  (T (REPLACE CONCEPT
                          (QUOTE
                            ((ACTOR (#SUBJ)<=>(INGEST)
                              OBJECT (#OBJ)
                              TO (INSIDE PART (#SUBJ))
                              FROM (MOUTH PART (#SUBJ)))
                              MODE (NIL) TIME (NIL))))
         NIL)
         ((FEATURE SENSE (QUOTE POBJ))
         (CHOOSE OBJ SENSE) NIL)
```

The first request sets CONCEPT to a representation of "drinking". Graphically this is:

$$\begin{tikzcd}
 & & & & \text{INSIDE (SUBJ)} \\
\text{(SUBJ)} \arrow[r, Leftrightarrow] & \text{INGEST} & \text{(OBJ)} \arrow[l, "o"'] & \cdot \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{MOUTH (SUBJ)} \arrow[ul]
\end{tikzcd}$$

The second request looks for anything physical to use as the object of the ingesting. Now it is true that drinking wants an object that is a liquid, and this information should be made available to the deductive section of memory by the analyzer. Thus, a fuller graph for DRINK1 would have a predication on the object that the OBJECT is a liquid. However, while this information should be part of the output of the analyzer, it does not affect the analysis itself. That is, if the sentence was "John drank a chair", the fact that a chair is not a liquid does not prevent this sense of "drink" from being used nor does it change the result, which would be that John ingested a chair and the chair must have been liquid. A different matter however, for example, is the fact that "Mary" refers to a human, not just to a physical object. This feature affects the decision the analyzer makes when it reads "Mary was given..." as opposed to the decision it makes with "The book was given...".

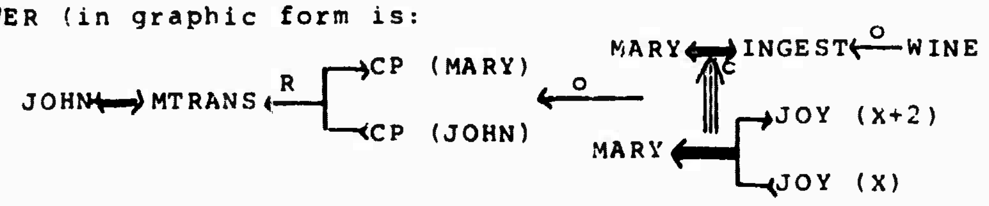
There is also a request for TOØ, the sense of "to" which has been set by a request of GIVE1.

TOØ has the form:

```
TOØ:  (T (PROG NIL (IMBED (MOBJECT CON)
           ((SUBJ CHOICE RECIP)
           (TIME AFTER (NEW_TIME) CHOICE TIME) X))
           ((BREAK_POINT) (RESET_ALL) NIL))
           (SETQ USE_TIME NIL)) NIL)
```

This request has two actions. The second, and simpler, is the setting of the variable USE_TIME to NIL. This variable is referred to by the function NEED_TIME, and the value NIL indicates that a TIME is not needed. The first action is an IMBED call. There are several subactions that IMBED can perform and this example uses them all. The first argument to IMBED gives a path which IMBED will follow. The value of CONCEPT will be saved and CONCEPT will be reset to the (possible empty) structure at the end of this path. The second argument specifies some new values for, in this case, SUBJ and TIME. IMBED first saves the former values of SUBJ, OBJ, and RECIP, then sets the three variables to NIL. Then the second argument, which is a list of pairs, tells IMBED that the first element of each pair should be set to the value of the second, where, in the second element, all references to variables are to their values before IMBED was called.

The analysis of the sentence "John advised Mary to drink the wine" proceeds simply enough. The monitor-initialized request looking for a subject (SUBJ) is satisfied by "John". "Advised" and ADVISE1 satisfy no requests but add their own to REQUESTS and further change CONCEPT (and hence ANSWER) to a conceptual skeleton of the MTRANS action. "Mary" has the sense MARY1 which satisfies the request looking for a recipient of the MTRANS. The word "to" has the sense TOØ. IMBED is called by TOØ and moves CONCEPT to head of the causal that makes up the conceptualization being MTRANSed. IMBED also resets REQUESTS. "John" has the sense JOHN1 which satisfies the request now being made for SUBJ. "Drink" has the sense DRINK1 which puts the conceptual skeleton for a drinking action into CONCEPT (which is still pointing to the head of the causal in the MOBJECT slot). "Wine" has the sense WINE1 which satisfies the request looking for an OBJ of the drinking. The end of the sentence causes REQUESTS and the syntactic cases and CONCEPT to be returned to the values they had before "to" was encountered. REQUESTS is checked again, nothing happens, and the analysis is over. The value of ANSWER (in graphic form is:



This says that John communicated to Mary that her ingesting wine could cause her to undergo a positive increase in the joy she feels.

The output in linear form is:

```
((ACTOR (JOHN1)<=>(MTRANS)
                   TO (CP PART (MARY1) REF (THE))
                   FROM (CP PART (JOHN1) REF (THE))
  MOBJECT ((CON ((ACTOR (MARY1)<=>(INGEST)
                    OBJECT (WINE REF (THE))
                    TO (INSIDE PART (MARY1))
                    FROM (MOUTH PART (MARY1))) MODE (NIL)
                                                TIME (TIMØ2))
             ⇐C ((ACTOR (MARY1)⟺T (JOY)⟺F (JOY))
                   INC (2) TIME (TIMØ2) MODE (NIL)))))
                    MODE (NIL) TIME (TIMØ1))
```

where TIMØ2 has the value ((AFTER TIMØ1)) and TIMØ1 has the value ((BEFORE TIMØØ X)) and TIMØØ has the value ((VAL T)).

The next example is "John killed Mary by choking Mary". It contrasts with the "gave a beating" example in the kind of manipulation that occurs. In "John gave Mary a beating", the meaning of "give" was a set of actions more than some conceptual piece. The actions built a conceptualization from a structure attached to other words in the sentence. In this example, "John killed Mary by choking Mary", the word "by" ties together two large conceptual pieces, "John did something that caused Mary to die" and "John grabbed Mary's neck so she couldn't breathe". "By" asks questions about conceptualizations rather than about words and differs from "give" in that way.

BY1, the sense assigned to this use of "by", has the following job to do. It has to tie together two conceptualizations, a main one and a secondary one, making the latter "instrumental" to the former. If the two actions are simple EVENTs, then the main action has the secondary action in its INSTRUMENTAL case. If the main action is a causal and the antecedent is unspecified (graphically there is a dummy "do" written for the act) then the secondary action is helping to specify the antecedent. If the secondary action is a simple act then it is directly placed in the antecedent slot. This happens in "John angered Mary by giving Bill the book," where the ATRANS action by John made Mary angry. If the secondary action is a causal itself, then the effect event of this secondary action is in turn the antecedent event of the main action. This happens in our example, as we shall see.

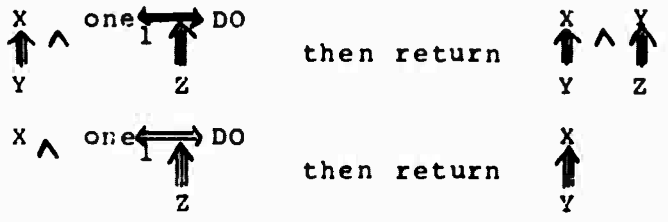
The requests for BY1 are as follows:

```
BY1:
     ((CHOICE ⇚)
      (PROG NIL
            (REPLACE CONCEPT
                     (QUOTE ((CON (NIL TIME
                                       (BEFORE (NEW_TIME)
                                       (CHOICE CON TIME) X)
                                       MODE (NIL))
                                ∧ CONCEPT))))
            (ADDREQ
              ((AND (CHOICE CON) (CAR (CHOICE CON)))
               (COND ((EQUAL (CHOICE ∧ CON ⇔)
                             (QUOTE (DO)))
                      (COND ((CHOICE CON ⇚)
                             (RPLACA  (SEARCH (QUOTE CON)
                                      (CAR (SEARCH (QUOTE ∧)
                                                   CONCEPT)))
                                      (CHOICE CON ⇚)))
                            ((CHOICE CON ⇔)
                             (REPLACE CONCEPT
                                      (QUOTE ((CON (#CON,
                                                   ⇚ (# ∧ ⇚)))
               NIL)))))))
              (IMBED CON ((SUBJ CHOICE SUBJ))
                         ((BREAK_POINT) (RESET_ALL) NIL)) )
      NIL)
```

```
((CHOICE⇐⇒) (IMBED INST ((SUBJ CHOICE SUBJ))
                          ((BREAK_POINT) (RESET_ALL) NIL))
 NIL)
```

This definition of BY1, as will be pointed out shortly is wrong and a better one would be simpler. However, this definition is a good example of how complex the actions that requests perform can be, if it is necessary. The operations above are more like memory routines in that conceptual structures are being manipulated according to features of other conceptual structures rather than according to linguistic factors. This capability of the analyzer, to go as deep conceptually as it needs, is an important aspect of this approach.

The above definition consists of two requests. Each one has an expectation about the structure of CONCEPT. The first one looks for CONCEPT to be a causal relationship, while the second looks for CONCEPT to be an actor. If the second succeeds, the action performed is an IMBED that will cause the clause following the "by" to be analyzed as a specification of the instrument, INST, of this action. The first request, if its expectation is satisfied, performs three subactions. The first action is to attach a new slot to CONCEPT and make the whole structure the new value of CONCEPT. The third subaction is an IMBED that causes the clause following the "by" to be interpreted as filling in this empty slot. The second subaction is concerned with the fact that if the clause preceding the "by", the main clause of the sentence, was interpreted as meaning the SUBJ did some unknown action that caused something else to happen, then the "by" clause is specifying the unknown action. This is done by having the second subaction add a request that waits for the empty slot to be filled. The action for this new request, if the main clause is a causal with an antecedent "DO" action, combines the "by" clause conceptualization with the main clause conceptualization by the following paradigm:



These are the requests, then, for "by". The requests for "killed" and "choking" are fortunately much simpler. The word "killed" has the same past tense request that "gave" and "advised" had. Nothing at the moment is done with "choking" except to recognize that it is a form of "choke". Some words have requests that test for an "-ing" form, such as forms of the word "be", but none occur in this example. The major requests in this sentence then are BY1, above and KILL1 and CHOKE1 below:

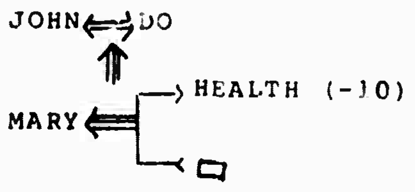
```
 KILL1:  (T
          (REPLACE CONCEPT
                   (QUOTE ((CON ((ACTOR (#SUPJ)⟺(DO))
                                 TIME (NIL) MODE (NIL))
                            ⟸((ACTOR (#OBJ)
                                 ⟺T (HEALTH VAL (-10)))
                                     MODE (NIL) TIME (NIL))))))
         NIL)
         ((FEATURE SENSE (QUOTE HUMAN)) (CHOOSE OBJ SENSE) NIL)
 CHOKE1: (T
         (REPLACE CONCEPT
                  (QUOTE ((CON ((ACTOR (#SUBJ)⟺(GRASP)
                                OBJECT (NECK PART (#OBJ)))
                                TIME (NIL))
```

```
                                  ⇐((ACTOR (≡CON OBJECT PART)
                                      ⟺(INGEST)
                                      OBJECT (AIR REF (A))
                                      FROM (MOUTH PART (≡⇐ACTOR))
                                        TO (INSIDE PART (≡⇐ ACTOR)))
                                      TIME (NIL) MODE ((CANNOT))))
                                                   )))    NIL)
                  ((FEATURE SENSE (QUOTE PP)) (CHOICE OBJ SENSE) NIL)
```

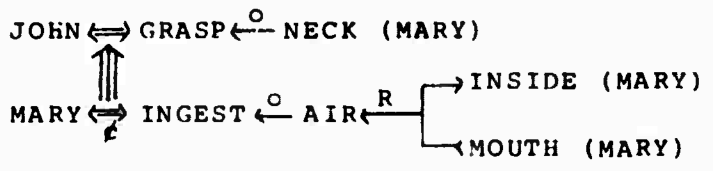
Both senses consist of two requests. In each case, the first request provides the conceptual structure and the second looks for a filler for the OBJ case. The two first requests provide conceptualizations whose main link is a causal.

The conceptual structure for KILL1 says that someone did something that caused someone to die. The conceptual structure for CHOKE1 says that someone grabbed someone's neck, causing that person to be unable to breathe. The KILL1 requests have been slightly simplified from a form that handles "the beating killed Bill" as well as "John killed Bill". And the check for HUMAN in the KILL1 request looking for an OBJ should be PP, as it is with CHOKE1. This is like the situation with "drink" where it might seem reasonable to check for "liquid" as a feature of the OBJ. Even though "John killed the desk" is indeed strange, it is strange not because of a peculiarity of "kill" in English but because the output of the analysis, that a desk dies, is strange conceptually.

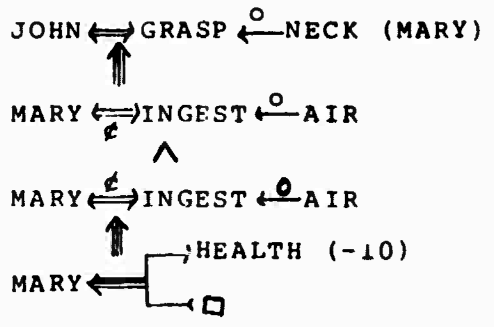
The analysis of "John killed Mary by choking Mary" is simple enough until "by" is reached. The value of CONCEPT at this point is, graphically:



BY1 then sets aside the requests still waiting, sets SUBJ to JOHN1, and starts building another conceptual structure from "choking Mary". When the end of the sentence is reached the substructure that has been built is, graphically:

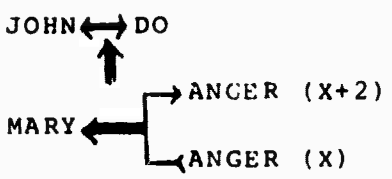


Because the end of the sentence has been reached, the RESET_ALL request that was provided by the IMBED call operates to bring back the original REQUESTS, including the one that looks to see if a conceptualization has been built. One has been, and the request, according to the paradigm given, takes the "kill" conceptualization and the "choke" conceptualization and forms the final answer, which graphically is:



This involves two conceptual forms. The first one says that John grasped Mary's neck causing her to be unable (marked by ¢ on the $\Longleftrightarrow$ link) to breathe. The second one says that this inability to breathe caused Mary to die, where death is the lowest point on a scale of health.

It was said earlier that this definition of BY1 was incorrect. In the above example it produces the correct answer, but consider the sentence "John annoyed Mary by choking Bill." "John annoyed Mary" is analyzed to produce the causal structure:



Thus the pattern for "annoy" is the same as for "kill", as far as BY1 is concerned. However it is neither necessarily true nor even probably true that Mary was annoyed by Bill being unable to breathe. Rather it was the whole event, of John causing Bill to be unable to breathe, that annoyed her. BY1 should more simply say that the conceptualization, assigned to the clause following "by", replaces the dummy causing action in the conceptualization for the main clause, if there is a dummy action. It is an inference from world knowledge that decides what aspect of the "by" conceptualization caused the final result. Thus with "John killed Mary by choking Mary", it is a fact about humans (and animals in general) that they die from being unable to breathe.

A simpler sentence than the previous one, but demonstrating that not everything in the analyzer must be complicated because it goes so deep, is the sentence "Did John give Mary a book?" The only element of this sentence that hasn't been described already is "did". The word "did" has a past tense request like "gave". The sense of "did" used is DO1, and it has two requests:

```
     DC1:  ((NULL (CAR (CHOICE SUBJ)))
             (RPLACA  (CHOICE MODE)
                      (CONS (QUOTE ?) (CAR (CHOICE MODE))))
           NIL)
           (T (SETQ USE_TIME NIL) NIL)
```

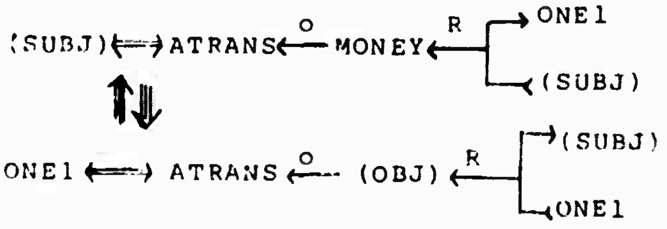
The second request says that the time specification has already been taken care of. Therefore USE_TIME is set to NIL so that (NEED_TIME) returns false and a later verb form will not affect the value of TIME. The first request is activated if no SUBJ has been chosen yet, that is, if the word form of DO1 is before any noun phrase in the sentence. If this is true then the MODE of the conceptualization to be built is set to "?" which means that a question is being asked about the truth of the conceptualization. This is all that "DO1" consists of.

The analysis of "Did John give Mary a book?" starts with the requests for "did" setting the TIME to before the time of utterance, setting USE_TIME to NIL, and setting MODE to "?". The rest of the sentence prodeeds exactly as before, except that the time setting request on "give", which is like the one for "drink", is not activated. The final result is thus:

```
((ACTOR (JOHN1)⟺(ATRANS) TO (MARY1)
  FROM (JOHN1) OBJECT (BOOK1 REF (A)))
  TIME (TIMØ1) MODE ((?)))
```

where TIMØ1 is before TIMØØ in value.

The next example, "John prevented Mary from buying the book by giving the book to Rita," is a reasonably complex sentence but turns out not to require much more than we've already described. The words with requests not previously given are "prevented", "from" and "buying". We won't take time to describe "buying" simply because it is not substantially different from previously given verbs. Basically it has a request that produces the following structure:



There is also a request with "buy" that changes the current sense of "from" so that if a human follows, that person fills the positions indicated by "ONE1" in the graph. There are the usual requests for SUBJ and OBJ. Finally there is a request that looks for a human who is neither the SUBJ nor the object of "from". If one occurs, then the whole buying action is embedded within a larger conceptualization that says that this buying was done in order to give the object bought to this other person. This request is for handling "John bought Mary a book." However, this request is not invoked in the current example.

The requests for "prevented" are the past tense request plus the requests that make up PREVENT1:

```
PREVENT1:
      (T (REPLACE CONCEPT
                       (QUOTE ((CON ((ACTOR (#SUBJ)⟺(DO))
                                         TIME (NIL))
                                   ⟸((ACTOR (#OBJ)⟺(DO))
                                         TIME (NIL)  MODE ((CANNOT))))
     )))NIL)
          (T (DEFPROP BY BY1 CURRENT) NIL)
          ((FEATURE SENSE (QUOTE HUMAN)) (CHOOSE OBJ SENSE' NIL)
          (T (DEFPROP FROM FROMØ CURREN' ) NIL)
```

The first request produces the following structure:

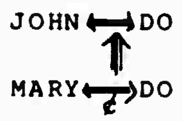
$$\begin{tikzcd}
(SUBJ) & \Leftrightarrow & DO \\
(OBJ) & \underset{c}{\Leftrightarrow} \arrow[u, Rightarrow] & DO
\end{tikzcd}$$

That is, "prevent" says that someone did something which caused someone else to be unable to do something. The second and fourth requests set senses for "from" and "by", and the third request looks for the person being prevented. The sense given to "by" is the same as was described before. The sense assigned to "from" has the request:

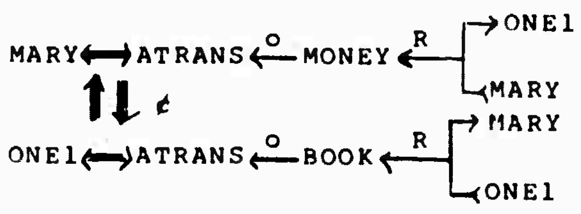
```
FROMØ:
     (T (IMBED⟸((MODE QUOTE ((CANNOT)))
                   (TIME CHOICE⟸TIME)
                   (SUBJ CHOICE OBJ))
               ((OR (EQ WORD (QUOTE BY))
                   (BREAK_POINT))
                (RESET_ALL) NIL))
     NIL)
```

There are two reasons why FROMØ is much simpler than BY1. First, FROMØ is specific to "prevent" and takes advantage of the knowledge that there is a dummy action present. BY1, being more general than is needed here, must first look to see if there is a dummy action to be filled. The other reason for the simplicity of FROMØ is that FROMØ does not worry, as BY1 incorrectly does, about breaking up the conceptualization assigned to the clause it precedes. Whatever conceptualization is built is placed, through IMBED, at the effect end of the causal link.

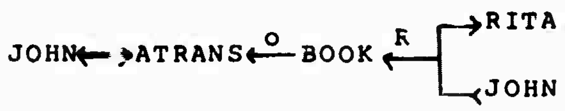
The analysis of the sentence proceeds simply. When the "from" is reached, the value of CONCEPT is, graphically:



The "from" requests set CONCEPT to where "MARY$\underset{\not c}{\Leftrightarrow}$DO" currently is, saves the information that MARY1 is the SUBJ and that the MODE is CANNOT. The analysis then produces the substructure:



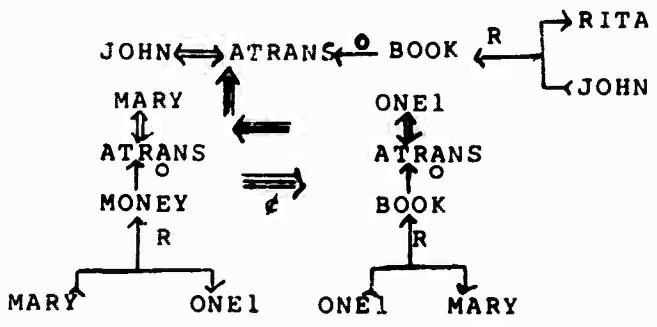
This is the "buy" structure with MARY1 replacing SUBJ and BOOK1 replacing OBJ and with a modification of the whole event by "$\not{c}$", i.e. CANNOT. This structure is now in the place where "MARY $\overset{\not{c}}{\Leftrightarrow}$ DO" used to be. The building of this structure ends with the encountering of the word "by". The request of FROMØ that is triggered by the discovery of the word "by" causes the "from" clause to be ended. This resets the various variables to the main clause level. Then the word "by" itself causes another embedding, this time to replace "JOHN $\Leftrightarrow$ DO" in the conceptualization. The information passed says that JOHN1 is the subject of this clause. The structure that is then built is:



The end of this clause is also the end of the sentence. The final result is:

```
     ((CON ((ACTOR (JOHN1)⟺(ATRANS) TO (RITA1) FROM (JOHN1)
              OBJECT (BOOK1 REF (THE))) MODE (NIL)
              TIME (TIMØ2))
          ⇐((CON ((ACTOR (MARY1)⟺(ATRANS) OBJECT (MONEY)
                   TO (NIL) FROM (MARY1)) TIME (TIMØ1))
            ⇐  ⇒((ACTOR (NIL)⟺ ATRANS)
                   OBJECT (BOOK1 REF (THE))
                   TO (MARY1) FROM (NIL)) TIME (TIMØ1)))
                   MODE ((CANNOT)))))
```

where TIMØ2 is before TIMØ1 which is before TIMØØ which has the value T. Graphically this is:



It would seem to be a fairly obvious assumption that the person, ONE1, who might have sold the book to Mary is John. However, this requires knowledge about how the transfer of an object can prevent its purchase. It also requires making a decision that the two occurrences of "the book" are referring to the same item. The analyzer does not make such decisions at this time. If it did, then it could deduce that ONE1 was JOHN1 because JOHN1 is the only person found who is capable of ATRANSing the book away from himself (as evidenced by the statement that he gave it to Rita).

The verb "want", in the sense of desiring something, is another example of the analyzer doing a little of the work that might be considered the job of memory. There are four requests for "want" that allow it to handle the following kinds of sentences:

John wants a book.
John wants Mary.
John wants to buy a book.
John wants Mary to buy a book.

The set of four requests, which has the name WANT1, is:

```
     WANT1:
((FEATURE SENSE (QUOTE PP))  (CHOOSE OBJ SENSE) NIL)
 (T (DEFPROP TO TO3 CURRENT) NIL)
 (T (REPLACE CONCEPT
              (QUOTE
               ((CON ((CON (NIL TIME (>) MODE NIL))
                       ⇐c((ACTOR (#SUBJ)⇔T (JOY)⇔F (JOY))
                            INC (2) TIME (>))))
                   ⇔(MLOC VAL (LTM PART (#SUBJ)
                                       REF (THE))))
                MODE (NIL) TIME (NIL))))
  NIL)
```

```
(T (ADD_BP
     (COND ((SETQ TEMP2 (WALK (QUOTE (CON CON)) CONCEPT))
            (COND ((CAR TEMP2) NIL)
                  ((FEATURE (CAR (CHOICE OBJ)) (QUOTE HUMAN))
                   (REPLACE TEMP2
                            (QUOTE ((ACTOR (#OBJ)<=>(PTRANS)
                                     OBJECT (#OBJ) TO (#SUBJ)
                                     FROM (NIL)) TIME (>))))))
                  ((FEATURE (CAR (CHOICE OBJ)) (QUOTE POBJ))
                   (REPLACE TEMP2
                            (QUOTE ((ACTOR (ONE1)<=>(ATRANS)
                                     OBJECT (#OBJ) TO (#SUBJ)
                                     FROM (≡ACTOR))
                                    TIME (>)))))))))
 NIL)
```

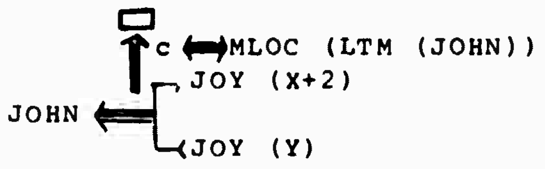
The first request looks for any conceptual thing to serve as the OBJ of "want". There may not be an OBJ, of course. The second request sets the sense of "to" to TO3 which will be shown shortly. TO3 functions basically to introduce a clause that describes what is wanted. The thrid request is the basic frame for wanting, that is, that, in the long term memory of the SUBJ, there is the belief that some particular thing will cause him to be happier. The fourth request insures that what is wanted is always some event. It uses the function ADD_BP, which is like ADDREQ except that the request added has BREAK_POINT as its TEST. Thus the argument to ADD_BP will be evalutate when the clause containing "want" ends. The ACTION looks at the conceptualization which has been built, when the end of the clause is reached describing what the SUBJ wants. The function WALK that is used is just a generalized form of CHOICE, and the arguments (CON CON) and CONCEPT point to the conceptual object of the wanting. If a conceptualization has been built there, nothing more is done. However if nothing has yet been constructed, this request fills in the gap. It does this in one of two ways, depending on whether the object involved in the desire is a person or a thing. If OBJ is a human, then the event that would please the SUBJ is for the OBJ to come to the SUBJ. If the OBJ is a thing, then SUBJ wishes to get, somehow, this thing.

The request for TO3 is:

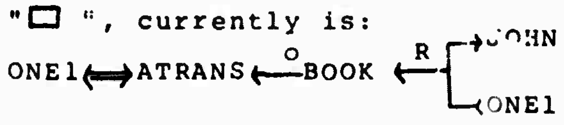
```
TO3:
(T (PROG NIL
         (IMBED (CON CON)
                ((SUBJ COND ((NULL (CAR (CHOICE OBJ)))
                             (CHOICE SUBJ))
                            (T (CHOICE OBJ)))
                 (TIME AFTER (NEW_TIME)
                             ((CHOICE TIME) X))
                ((BREAK_POINT) (RESET_ALL) NIL))
          (SETQ USE_TIME NIL))
NIL)
```

This request shifts the level of manipulations, after "to" is read, to the point where what SUBJ believes would give him pleasure is described. The SUBJ of the new clause is either the OBJ of the dominating clause, if there is an OBJ, or it is the SUBJ of that clause.

These requests are all that are needed to analyze the sentences given. In the first one, "John wants a book" and the second one, "John wants Mary," no "to" clause occurs before the end of the sentence. Therefore the analyzer finds that it has built no conceptualization for what John wants when the sentence is finished. The request that was added by ADD_BP is evaluated, with the OBJ set to "(BOOK1 REF (A))" in the first case and "(MARY1)" in the second, and with CONCEPT equal to:

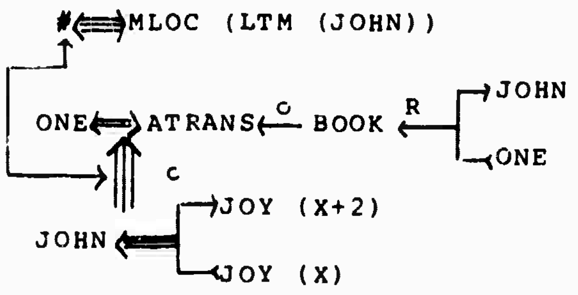


If the ADD_BP request sees OBJ equal to "(BOOK1 REF (A))", then it produces the following structure, and puts it where the gap, "□ ", currently is:



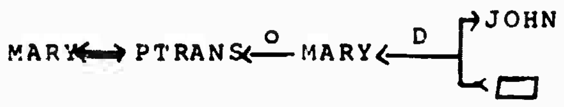
The linear form output and the graphical equivalent, are:

```
JOHN WANTS A BOOK
TIMØØ: ((VAL T))
TIMØ1: ((AFTER TIMØØ X))
TIMØ2: ((AFTER TIMØØ X))
((CON ((CON ((ACTOR (ONE1)⟺(ATRANS)
OBJECT (BOOK1 REF (A)) TO (JOHN1)
FROM (ONE1))TIME (TIMØ2))
⟸c((ACTOR (JOHN1)⟺T (JOY)⟺F (JOY))
INC (2) TIME (TIMØ1))))
⟺(MLOC VAL (LTM PART (JOHN1) REF (THE))))
MODE (NIL) TIME (TIMØØ))
```



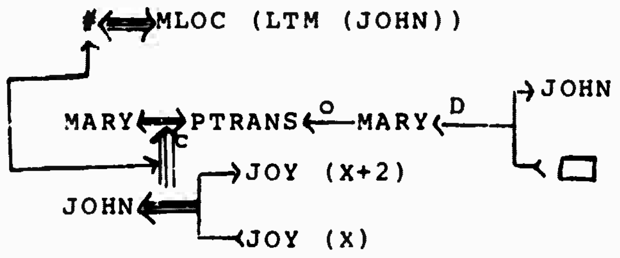
That is, John believes that someone giving (ATRANS) him a book will cause his joy to increase.

If the ADD_BP request sees OBJ equal to "(MARY1)", then it produces the following structure for what is wanted:



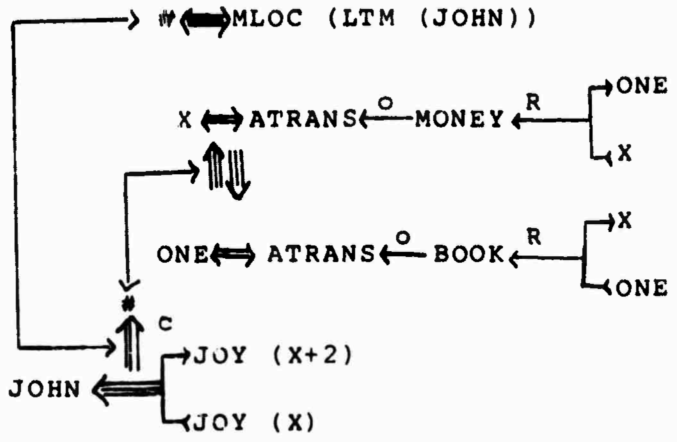
The linear form output and the graphic equivalent are:

```
JOHN WANTS MARY
TIMØØ:   ((VAL T))
TIMØ1:   ((AFTER TIMØØ X))
TIMØ2:   ((AFTER TIMØØ X))
((CON ((CON ((ACTOR (MARY1)⟺(PTRANS)
OBJECT (MARY1) TO (JOHN1) FROM (NIL)) TIME (TIMØ2))
⟸c((ACTOR (JOHN1)⟺T (JOY)⟺F (JOY))
INC (2) TIME (TIMØ1))))
⟺(MLOC VAL (LTM PART (JOHN) REF (THE))))
MODE (NIL) TIME (TIMØØ))
```



That is, John believes that if Mary comes to him his joy will increase.

In the third and fourth sentences, "John wants to buy a book" and "John wants Mary to buy a book," the TO3 sense of "to" is used. In the third sentence, this request starts the building of the subclause with "(JOHN1)" as the SUBJ. In the fourth sentence, this request starts the building of the subclause with "(MARY1)" as the SUBJ. In both cases the same structure for "buy" is built, but in the first case, of course, John is buying a book and in the second Mary is doing so. The final result is the following structure where "X" is "(JOHN1)" or "(MARY1)", depending on the sentence:



In both of these cases, the ADD_2P request that looks, when the sentence is finished, to see if a conceptualization describing what is wanted has been built, will find one. Hence it will not perform the actions necessary to fill a gap.

CHAPTER 8

MULTI-SENTENCE ANALYSIS

8.1 INTRODUCTION

The last few examples to be given involve the analysis of more than one sentence at a time. These examples are far from adequately treated, as will be seen. However they do show how the basic mechanisms that apply to simple sentences are appropriate for text as well. These examples touch lightly on some aspects of contextual effects. Certain contextual effects can be viewed as predictions made at one point in a text about what will be seen at a later point in the text. Compare this with the basic scheme of analysis which is the prediction at one point in a sentence about what will be seen at a later point in the sentence. Contextual predictions are not just analogously similar to sentential predictions, but are, with some extension, built from the same functional tools. One inadequacy in what is to be described arises from the fact that the two sets of predictions are not handled as one. Another inadequacy is that sometimes the interaction of requests occurs too late in the flow of the analysis. This leads to an artificial form of backup being needed, which is done but in an ad hoc non-generalizable manner. Both these problems are treated in a more uniform manner through the introduction of another basic mechanism, which is described in Part II. Simply expressed, the mechanism associates a request with the need it is filling. This allows requests to come and go independently and allows requests to easily recognize other requests. However what shall be described here is the analyzer system prior to this extension. Despite the clumsiness of some of the implementation, hopefully it will be clear how requests can carry information from the analysis of one sentence to the next.

The examples to be described are involved with changes in word sense choice. The effects and examples treated are:

1) Contextual lexicons - where certain domains of concepts have their own jargon associated with them. The program is affected by jargon in the text: "John and Mary were racing. John beat Mary" - where the second sentence is treated differently by the analyzer when appearing in isolation.

2) Conceptual predictions - where an expectation of a certain kind of conceptualization affects the analysis of a sentence. Two texts where conceptual predictions are handled by the program are:

   a) "John hated Mary. John gave Mary a sock" - where the second sentence is treated differently by the analyzer when appearing in isolation.

   b) "John was hunting. John shot a buck" - where, again, the analysis of the second sentence is different in isolation.

8.2 MODIFICATIONS TO THE ANALYZER

In order to do the above examples, the format of the request was extended slightly, an extra monitor variable was added, and several new functions were written. The format change merely involved the adding of a field to each request specifying the word or sense that was the source of that request. The monitor did this automatically. The extra variable was IM_REQS. IM_REQS is a list of requests like REQUESTS. The distinction is that REQUESTS is re-initialized to a starting set of requests at the beginning of each sentence. IM_REQS, however, is unchanged by the occurrence of sentence boundaries, except of course, insofar as a request it contains may be triggered by a sentence boundary.

The new functions are of two sorts. Some are intended as mechanisms for passing information. Others are for performing certain manipulations on the basis of this information. The variable IM_REQS holds the information passing functions, while the extra field in the requests is needed for some of the manipulations.

One of the information passing functions is CONSIDER_IM, which is like the function CONSIDER. Where CONSIDER checked the list REQUESTS, CONSIDER_IM checks the list IM_REQS. Another function is CONDICT. CONDICT is one of a proposed set of functions that takes a list of conceptualizations containing some special forms, and interprets these forms to provide links that tie these conceptualizations together. The particular job of CONDICT is to interpret certain special forms that tie pieces of a conceptual cluster to related lexical items. The form used is ($$$ (X Y)) where X is a word, like "beat", and Y is a sense of that word, like BEAT2.

This form is not a conceptualization because it does not relate a concept with a concept, but rather a word of the language with a list of programs, i.e. a sense. CONDICT is the basi function used to implement a conceptual lexicon. The IM_REQs list, containing requests passed from sentence to sentence, is used to implement conceptual predictions.

A new function that manipulates the flow of analysis is BACKUP. It calls two functions, REMREQ and UNREQ, and relies on information about the analysis saved by LAST_SEEN. BACKUP redoes the analysis of a sentence from some word on. In general, where this reanalysis should begin should be determined by the type of conflict that occurred. However in this program it is assumed that analysis should go back just one word, which is done by the function LAST_SEEN. Undoing the analysis involves several actions. The requests that were added by the no longer desired word sense must be removed. This is done by REMREQ, using the extended request format to decide which requests should be deleted. Also, the requests that the previous word sense triggered, and hence have had their FLAGs set to T, need to be restored. This is the job of UNREQ, but UNREQ takes advantage of the fact that analysis is backing up only one word and only in examples that have a conflict at the end of the sentence. The function UNREQ reactivates all the requests that the word sense might have triggered, whether or not it actually did. Obviously this could cause problems in general, but UNREQ was sufficient for the task of the moment. Another action, which might be necessary, is to erase the structures built by the previously triggered requests. However it is sufficient here just to have the reanalysis overwrite what was previously built.

There are three pairs of pro and con that I can say about the above:

1) Con - Backup does not seem to occur for me on the word "sock" in example (2.a) and I am more interested in avoiding it.

   Pro - Some people do hesitate before choosing the right sense of "sock" and for them backup would seem a reasonable model.

2) Con - The functions for doing backup are very limited in their applicability.

   Pro - The functions given, however, indicate how the various actions involved in backing up could be distributed.

3) Con - Given the intention that requests represent the predictions made from and about the recognition of some situation, it seems wrong to have predictions that mistakes will be made.
   This is particularly true in the example (2.a), which, as implemented, seems to say that a general concept of not liking someone predicts the need for backup in later sentences.

   Pro - As was mentioned, there has been further extension done on the analyzer. This extension has included a generalization of the idea of request, along with a generalized form of control over requests. The general concept of a request includes a more integrated view of backup. It is the source of such requests that differs from the way the implementation was done here.

8.3 EXAMPLES OF TEXT ANALYSIS

Consider, now, the analysis of the multi-sentence text, "John and Mary were racing. John beat Mary," versus the isolated sentence "John beat Mary." If the analyzer understands "beat" in the same way in both cases then the multi-sentence example would require no information to be passed from the first sentence to the second. However, if the analyzer understands "John beat Mary" in isolation as meaning John hit Mary repeatedly, then the analyzer must be capable of changing in context. Therefore, for demonstration purposes, the analyzer was put into this second situation, where "John beat Mary" in isolation was not interpreted the same as when it followed "John and Mary were racing."

The verb "beat" is assumed to have two senses, BEAT1 and BEAT2. Both senses are simple, consisting of two requests, one providing the conceptual frame and the other looking for an OBJ.

```
         BEAT1:
((FEATURE SENSE (QUOTE PP)) (CHOOSE OBJ SENSE) NIL)
(T (REPLACE CONCEPT
            (QUOTE ((ACTOR (#SUBJ)⟷(PROPEL)
                    OBJECT (#OBJ))
                    TIME (NIL)))) NIL)
         BEAT2:
((FEATURE SENSE (QUOTE PP)) (CHOOSE OBJ SENSE) NIL)
 (T (REPLACE CONCEPT
              (QUOTE
              ((CON
                ((CON ((ACTOR (#SUBJ)⟺(LOC VAL (ONE)))
                       TIME (ONE))
                    ∧ ((ACTOR (#OBJ)⟺(LOC VAL (ONE)))
                       TIME (ONE)) ))
                    ∧
                     ((CON ((ACTOR (≡CON CON TIME)
                               R1 (≡ CON TIME)))⟺(<<)))))))
NIL)
```

The instrument of the conceptualization in BEAT1 has been removed for simplicity. The conceptual frame in BEAT2 is meant to be suggestive of the idea of one person being in a location before, in time, another person gets there. This is not a regular CD construct, nor is it complete enough to represent the ideas involved, nor should BEAT2 be this specific to the concept of winning in a race. However these objections are irrelevant. The only thing that is important for our purposes here is the existence of two distinct senses of "beat".

The analysis of this sentence involves two words besides "racing" not described before and which shall be treated only briefly. ne is "and". The requests for "and" are valid only for a restricted class of sentences. Basically the request saves the noun phrase following the "and" and adds a request that waits for the end of the sentence. The rest of the sentence is analyzed as if only the noun phrase preceding the "and" had been read. When the end occurs, this request attaches to the structure that has been built a copy of that structure, with the sense of the saved noun phrase appearing everywhere in that structure that the sense of the first noun phrase did. So the result of analyzing "John and Mary were racing" is the same as the result for "John was racing and Mary was racing." It would have to be inferred that John and Mary were probably racing against each other.

The "were" in the sentence, the other new word, has a past tense request of its own and has the sense BE1. BE1 has a number of requests, which look at the item to come to see if it is a verb or a noun or an adjective. For example, there is a request such that if there is a verb in the past participle form, then actions are taken treating the subject of the verb as if it had appeared following the verb. This is the request that handles passive sentences. The simplest request in BE1 is the one we need, which says that if the word is a progressive verb form, i.e. ending in "ing", then do nothing except remove the other requests BE1 had set up and treat the progressive verb form as the main verb in the sentence.

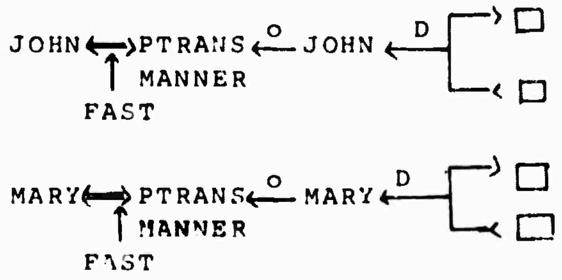
"Racing" is not complicated, but it is the word that introduces the contextual lexicon that affects "beat". "Racing" has just one request which provides an <u>ad hoc</u> conceptual frame. This conceptual frame request is:

```
         RACE1:
(T (REPLACE CONCEPT
            (QUOTE
            ((ACTOR (#SUBJ)⟺(PTRANS)
              OBJECT (=ACTOR) TO (NIL) FROM (NIL))
              TIME (NIL) MANNER (FAST)
              MODE (NIL)⟷($ RACECON))))
NIL)
```

There is no request looking for an object simply because the example text only used "race" intransitively.

"Racing" then is analyzed to the same structure as "running". However it differs from "running" because of the form "($ RACECON)". This points to a cluster of conceptualizations. RACECON is a Lisp atom whose value is a list of conceptualizations and special forms. The call on REPLACE in turn calls FIXUP when interpreting a Lisp form, and FIXUP calls CONDICT when it sees "($ RACECON)". CONDICT ties the conceptualizations in the list together. Briefly, one of them says that there is a group of people, another says that each person wants to do better than any-one else in the group, and another says that doing better than someone means getting to some location before that someone does. Further, within the first conceptualization there is a pointer to the form "($$$ (RACER RACER1))" which says that the contextual lexicon entry for a member of this group is "racer". Within the second conceptualization there is a pointer to the form "($$$ (BEAT BEAT2))" which says that the lexicon entry for doing better is "beat". CON-DICT makes BEAT2 the CURRENT sense of "beat". Thus, when the sentence "John and Mary were racing" is finished by the analyzer, the output is:

```
((CON ((ACTOR (JOHN1)<==>(PTRANS) OBJECT (JOHN1)
         TO (NIL) FROM (NIL)) TIME (TIMØ1) MANNER (FAST)
         MODE (NIL)<-->(NIL))
   ∧  ((ACTOR (MARY1)<==>(PTRANS) OBJECT (MARY1)
         TO (NIL) FROM (NIL)) TIME (TIMØ1) MANNER (FAST)
         MODE (NIL)<-->(NIL))))
```



and furthermore the CURRENT sense of "beat" is BEAT2. There-fore, when the next sentence, "John beat Mary," is analyzed, the result is the conceptualization of "John finished the race before Mary":

```
((CON ((CON ((ACTOR (JOHN1)⟺(LOC VAL (ONE)))
                 TIME (TIMØ1))
             ∧ ((ACTOR (MARY1)⟺(LOC VAL (ONE)))
                 TIME (TIMØ2)) ))
   ∧  ((CON ((ACTOR (TIMØ1) RI (TIMØ2)))
             ⟺ ((<)))))
```

$$\begin{tikzcd} \text{JOHN} \arrow[r, Leftrightarrow, "T_1"] & \text{LOC (ONE)} \wedge \text{MARY} \arrow[r, Leftrightarrow, "T_2"] & \text{LOC (ONE)} \\ & \wedge & \\ & T_1 < T_2 & \end{tikzcd}$$

However, had the sentence "John beat Mary" been analyzed prior to the encountering of "racing", then the result of the analysis would have been the same as for "John gave Mary a beating," because the analyzer assumes the normal sense for "beat" is BEAT1.

The next multi-sentence text example is "John hated Mary. John gave Mary a sock." The analyzer is set so that when analyzing "John gave Mary a sock" in isolation, the result is a simple ATRANSing:

```
((ACTOR (JOHN1)⟺(ATRANS) TO (MARY1) FROM (JOHN1)
  OBJECT (SOCK1 REF (A)) TIME (TIMØ1))
```

This is "a sock" interpreted as a simple physical object, like "a book". However, in this text situation, the desired result is assumed to be that "sock" is an action, like "beating" in "John gave Mary a beating," and the analyzer output for "John gave Mary a sock" should be:

```
((ACTOR (JOHN1)⟺(PROPEL) OBJECT (MARY1))
  TIME (TIMØ1))
```

where we have simplified by leaving out the instrumental case. The problem here is to specify what "John hated Mary" could do that would have this effect. Unlike a race or a contes where we can envision an associated jargon, a contextual lexicon, here it would be very unlikely that any direct association exists between "hate" and "sock".

The analyzer sets up a conceptual prediction when it sees "hate". This conceptual prediction is a request that is placed in IM_REQS. The TEST of this request is a pattern match on later constructions in CONCEPT. The pattern matches any structure about the person who is hating doing something. The ACTION taken if this pattern is found is to ask memory if the person who is hating is doing something bad to the person he hates. If so, nothing happens. If not, the function BACKUP is called to try a different sense of the last word seen. If a new CONCEPT is built then this is checked for the same property and so on, till either an acceptable CONCEPT is found or BACKUP fails to produce a different value of CONCEPT.

When the analyzer runs on this specific example text, then the sentence "John hated Mary" is analyzed as:

```
((ACTOR (JOHN1)<=>(MFEEL) OBJECT (ANGER)
  TO (MARY1)) TIME (TIMØ1) MODE (NIL))
```

$$\begin{tikzcd} & & & & \text{MARY} \\ \text{JOHN} \arrow[r, Leftrightarrow] & \text{MFEEL} & \text{ANGER} \arrow[l, "O"'] & {} \arrow[l, "R"'] \arrow[ur] & \\ & & & & \square \arrow[ul] \end{tikzcd}$$

This is only an ad hoc representation of "hate". More important is the following request that now appears in IM_REQS:

```
((FITS CONCEPT
  (QUOTE ((ACTOR (JOHN1)⟺ONE1)))
           NIL)
  (PROG NIL
        (PRINTSKIP (CAT CONCEPT (QUOTE "OK?")))
        (COND ((NULL (READ))
               (BACKUP)) (T NIL)))
NIL)
```

The TEST is almost as we described it. The ACTION substitutes a real memory call with a message to the console. When "John gave Mary a sock" is analyzed, the first message to the console is:

```
((ACTOR (JOHN1)⟺(ATRANS) TO (MARY1) FROM (JOHN1)
  OBJECT (SOCK1 REF (A))) TIME (TIMØ1)) OK?
```

We respond with a NIL because ATRANSing Mary a sock will not cause her to undergo a negative state change. Hence BACKUP is called and another sense of "sock" is chosen. This time the choice is SOCK2, which refers to the act of hitting someone. The message to the console is:

```
((ACTOR (JOHN1)⟺(PROPEL) OBJECT (MARY1))
  TIME (TIMØ1)) OK?
```

Since this would cause a negative state change in Mary, we reply T. BACKUP is not called again, and the above choice for CONCEPT is accepted as the final result.

The problem with the way this example is handled is that it was necessary to construct the whole conceptualization before rejecting the sense of "sock" chosen. It would seem preferable, if not in this case then certainly in others, to be able to contextually affect requests directly rather than acting after the requests have produced results. The two sentence text example, "John was hunting. John shot a buck," is related to this kind of approach. As with "sock" in the last example the key element is the ambiguity of the word "buck". We have two meanings of "buck" available in the dictionary One is BUCK1 which is an animal.The other is DOLLAR1, i.e. money. The analyzer when presented with the sentence "John shot a buck" in isolation produces the output:

```
((ACTOR (JOHN1)⟺(ATRANS) OBJECT (DOLLAR1 REF (A))
FROM (JOHN1)) TIME (TIMØ1) MODE (NIL))
```

The above structure is graphically:

$$\begin{tikzcd}
JOHN \arrow[r, Leftrightarrow] & ATRANS & DOLLAR \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[r] \arrow[dr] & \square \\
 & & & & JOHN
\end{tikzcd}$$

That is, John spent a dollar.

This results not because "shot" is assumed to mean spend. In fact "John shot..." is initially analyzed as:

```
((ACTOR (JOHN1)⟺(PROPEL) OBJECT (BULLET)
  TO (NIL) FROM (JOHN1)) TIME (TIMØ1))
```

SHOOT1, the sense of "shot" involved, has three requests, and one is capable of overwriting the effects of the others. The three requests are:

```
         SHOOT1:
(T (REPLACE CONCEPT
            (QUOTE ((ACTOR (#SUBJ)⟺(PROPEL)
                    OBJECT (BULLET) TO (#OBJ)
                    FROM (=ACTOR)) TIME (NIL))))
NIL)
```

```
((FEATURE SENSE (QUOTE ANIMAL))
 (CHOOSE OBJ SENSE) NIL)

((FEATURE SENSE (QUOTE MONEY))
 (PROG NIL (CHOOSE OBJ SENSE)
      (REPLACE CONCEPT
                 (QUOTE ((ACTOR (#SUBJ)<=>(ATRANS)
                          OBJECT (#OBJ) FROM (≡ACTOR))
                          TIME (NIL) MODE (NIL)))))
NIL)
```

The first request builds the shooting of bullets frame. The second says that a noun phrase referring to an animal should be assumed to be the OBJ. The third says that if a noun phrase referring to money is seen, then the action is really not shooting bullets, but rather it is spending money.

"Buck" is given a CURRENT sense of DOLLAR1 so that the third request is activated when "John shot a buck" is analyzed. Hence the PROPELing of bullets is overwritten to be the ATRANSing of money. The problem then is how "John was hunting" can change the sense of "buck" from money to animal. It is possible that this is a contextual lexicon effect. However it is also possible to handle it by a conceptual prediction in such a way that, unlike with "hated" and "sock", only one conceptualization is ever constructed. That is, the analysis program does not have to build and reject the conceptualization saying that "John spent a dollar" before it gets the one it wants.

The analysis of "John was hunting" produces two results. One is the following conceptualization, which is an oversimplification of the idea of "hunting":

```
((CON ((ACTOR (JOHN1)<=>(DO)) TIME (TIMØ1))
   <=E ((ACTOR (JOHN1)<=>(PROPEL) OBJECT (BULLET)
         FROM (JOHN1) TO (NIL)) TIME (TIMØ1))))
```

The above is graphically:

$$\begin{tikzcd}
JOHN \arrow[r, Leftrightarrow] & DO & & & \\
JOHN \arrow[r, Leftrightarrow] \arrow[u, Rightarrow, "E"] & PROPEL & BULLET \arrow[l, "O"'] & \cdot \arrow[l, "D"'] \arrow[r] \arrow[d] & \boxed{} \\
 & & & JOHN &
\end{tikzcd}$$

The other result is that the following request is built and added to IM_REQS:

```
((FITS CONCEPT
        (QUOTE ((ACTOR  (JOHN1)<=>(PROPEL)  OBJECT (BULLET))))
        NIL)
  (ADDREQ
    ((NOT (FEATURE OBJ (QUOTE ANIMAL)))
     (BACKUP) NIL))
NIL)
```

This means that when the sentence "John shot a buck" is analyzed, the following actions occur. "Shot" is interpreted, as always, as referring to PROPELing bullets. This triggers the IM_REQS request and thus a request is built that objects to choosing an OBJ that is not an animal. "A buck" is read and initially interpreted as referring to money. Immediately, the newly added request rejects this interpretation of "a buck", BACKUP is called, and another sense of "buck" is found, which FEATURE does find to be an animal. Therefore the final result is:

```
((ACTOR (JOHN1)<=>(PROPEL) OBJECT (BULLET)
  FROM (JOHN1) TO (BUCK1 REF (A))) TIME (TIMØ1) MODE (NIL))
```

Graphically this is:

$$\begin{tikzcd}
 & & & & \text{BUCK} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{PROPEL} & \text{BULLET} \arrow[l, "O"'] & {} \arrow[l, "D"'] \arrow[ur] \arrow[dr, leftarrow] & \\
 & & & & \text{JOHN}
\end{tikzcd}$$

There are of course unsatisfactory aspects about this solution. Certainly the request should not be dependent on the OBJ of the verb, since there may be a verb that means shooting but uses a preposition to signal the object being shot. A more general way of treating both this and the "hate" text is described in Part II. However, this example is only meant to show. as are the others, some of the ways by which contextual effects in text analysis can be implemented.

CHAPTER 9

REVIEW OF THE ANALYZER

The analyzer has been described by means of a number of examples presented in some detail. The word-oriented nature of the analyzer makes this kind of description necessary. Analysis occurs through the execution of programs, i.e. requests, that originally spring from individual words. The meaning of a word in this system is therefore a very dynamic thing, best illustrated in action. Further, during analysis there will be many of these programs. Examples are necessary to show how these programs interact with each other.

With the presentation of these examples, hopefully several points have been made clearer. For one, there is a close relationship between the basic assumptions, which were listed before the description of the analyzer functions, and the shape of the analysis program, between the theory and the implementation. For another, the word has been given a central role in the process of analysis but, at the same time, weakened as an isolatable element. That is, a word is important for the actions it performs. There may or may not be a conceptual structure closely associated with the word. Further, when work is done through words and not syntactic patterns, there is both a capability for and a bias towards producing conceptual interpretations directly. It is not necessary to first produce a syntactic description, as an intermediate goal, in order to interpret a sentence. Further, conceptual structures can be given to a deductive memory system. Therefore the intermediate results, which are conceptual structures, can also be given to memory. Hence memory can be used during, not just after, the analysis.

Chapter 8 on "Multi-Sentence Analysis" demonstrated that the expectation mechanism could be used to implement certain contextual effects. But it also demonstrated the need for a more systematic approach to the manipulation of expectations. Chapter 8, in other words, provided the motivation for the work in Part II.

I would like to have spent more time describing how the analysis of various situations proceeds. It is important to accept the claim that the expectation mechanism can handle the analysis of any reasonable sentence. Part II is based on the assumption that the analysis program, as given, is essentially correct. The sufficiency of the expectation mechanism cannot be proved, of course, but it can be made believable by applying expectations successfully to a large number of situations.

The analysis program is not a large one. It has a vocabulary of about sixty verbs, two dozen nouns, half a dozen prepositions, and a few adjectives. However the definitions of these words, particularly of the verbs and prepositions , are often quite complex and can handle a number of different constructions. Thus the analysis program, besides the examples that have already been given, can do the following:

1) "Is" constructions - The requests associated with "is" allow the analysis of predicate adjectives ("John is sick"), progressives ("John is going"), passives ("John was hit") and yes-no questions ("Is John coming?").

2) "Get" constructions - The verb "get" is like "give" in the range of types of objects it can appear with. The analyzer is capable of handling constructions like "get someone a sock", "get someone a job", "get a beating", and "get mad".

3) Verb-prepositions - Although there are only a few prepositions, there are many cases where the function of a preposition is totally determined by the main verb. Thus there are many "meanings" possible for these few prepositions. The analyzer has verb-preposition constructions for "agree-to", "agree-with", "agree-that", "help-to", "help-with", "remind-to", "remind-that", "swap-for", "swap-with", and others.

4) Causation - The analyzer is capable of handling verbs with implicit causation such as "kill", "hurt", "aggravate" and "bother". It can also handle an explicit statement of causation, using the word "because", as in "John went because Mary came." Further, the verb "cause" can be analyzed in sentences like "John caused Mary to drink the wine," "The beating caused Mary to be hurt," and "The book caused Mary to decide." Notice that in the first sentence, John does some action that affects Mary, whereas in the last sentence Mary does something that affects herself.

5) Unannounced clauses - The analyzer can handle cases where a new clause begins without an introductory word like "that" or "which". For example, the analyzer can treat "John told Mary the dog was sick" and "John promised Bill Rita could come."

6) Miscellaneous - Simple active interrogatives like "Who is coming?" can be handled.

- The construction "is going to", as in "John is going to go", is recognized as indicating a future time.

- The complex tense relationships that "has" and its forms can produce have not been treated, but the analysis program does handle "have a book", "have a headache", and "have a job" constructions.

- Simple adjectives in noun phrases are handled. The analyzer can do "the green book" or "a sick man", using the same senses of "green" and "sick" that are used when these words appear as predicate adjectives. The function FEATURE knows about modifiers. It can tell that "a green book" is a physical object by the definition of "book", and that it is green, by the explicit modifier.

Most of the definitions and programming for these constructions was done by Paul Martin, based on the definitions and analyses that were described in Chapter 7. Very few conflicts occurred when his new definitions and mine were combined. An advantage of writing programs for specific words rather than for syntactic word-classes is this modularity. Independent efforts can be done, extending the analyzer's capabilities, without interferring with each other, even though programming styles may differ quite a bit.

The time required to analyze sentences with constructions like the above depended more on the complexity of the conceptual manipulations involved than on the length of the sentence. For example "John gave Mary a book" was analyzed more rapidly than "John bought Mary a book." The latter involves building and attaching a second conceptualization to the buying structure because "Mary" appears where it does. But the time for analysis was rarely over five seconds of computer time, when the Lisp program was compiled, even for complex sentences.

The analyzer therefore is not solely an implementation of a theory of analysis. It is also a practical approach to computational language analysis. It seems particularly useful for those cases where the results of the analysis are quite different from the elements of the input.

In Part II this analyzer is extended. This extension is not the simple growth that results from more and more definitions. In fact, no new words are defined in Part II. Instead, definitions that have been described become the objects of other processes and manipulations. Far from being replaced by Part II, the analysis program of Part I becomes valuable in a new way.

PART II

GUIDE TO PART II

Part II is less technical than Part I because Part II has not (at the time of writing) resulted in a program. However, it uses the devices developed in Part I to express ideas specifically enough so that programming is not far away.

In Part II the general topic is the analysis of texts. It assumes that the way the analyzer handles sentences is essentially correct. The task then is to extend the analyzer in a consistent manner so that texts can be handled as well. To do this, two things have to be studied until structures for each can be seen.

One of these things is the nature of a text. What effects occur during the comprehension of a text that must be accounted for? This is the topic of Chapter 11. What general mechanisms produce these effects? This is the topic of Chapter 13. Chapter 13 starts with information stored in static forms. In particular, it is concerned with what pictures are present that say where the discourse is going. Then the chapter continues with a description of the process by which this static information comes to actively affect analysis.

To describe this conversion, we have to know what it is that is being affected. The second thing which must be studied, and for which a structure must be found, is the analyzer. In Part I, the analyzer was expanded to meet the needs of various examples. There was no predefined concept of what the analyzer should look like, except for the general one imposed by the goal of modelling human comprehension. But once there was enough of the analyzer to work with, it became both possible and necessary to look for implicit structures.

Chapter 12 describes the structure that resulted from studying the analyzer. The later sections of Chapter 13 show how this structure is used in a theory of text comprehension. Basically, a way is described of linking expectations to each other and to points in static forms. When everything is tied together in a large structure, information diffuses easily throughout the system. An important result of this is that useless forms and expectations do not accumulate. When something is removed, those elements that were tied to it and only it are also removed. Hence, at any point in the analysis, everything present is tied to and justified by the presence of something else. Section 12.2 presents a simple flow table, similar to that given in Part I. Like the previous flow table, this one should provide a helpful overview for the reader.

Chapter 14 wraps things up. It looks at what has happened and what is yet to come and maybe what it all means.

CHAPTER 10

INTRODUCTION TO EXTENDED ANALYSIS

The term "Extended Analysis" refers to two different kinds of extension to the previous work. First, there is the extension of the domain of analysis from isolated sentences to short texts of several sentences Second, there is the extension of the analyzer itself to satisfy certain needs of text analysis and to correct some of the deficiencies of the original analyzer. (The terms "text" and "text analysis", as used in the remainder of this description, refer only to the kinds of sequences of connected sentences and the analysis of them that the examples that will be given indicate. The terms do not refer to the body of work called "text analysis".)

In extending the domain to texts, there must be a theory about what a text is, about what makes a sequence of sentences form one unit. When the analysis of single sentences was both developed and described, it was implicitly assumed that the sequences of words analyzed did indeed form coherent units. In developing an approach to text analysis, it became clear that the same assumption must hold, that the analyzer must assume that each sentence it sees should, if possible, be tied in with the previous sentences.

A sequence of words is bound into one coherent sentence by the predictions that the analyzer makes when it sees these words. Words are defined in terms of the predictions that should be made when these words are seen. Word definitions emphasize how a word interacts with other words, rather than what the word in isolation might mean.

The theory of text analysis also stresses binding by predictions. To assume that a word is appearing in a sentence, is to associate with it predictive (hence interactive) information. To assume that a sentence is appearing in a text is also to associate with it predictive-interactive information. For a word this predictive information is in the definition the word has for the analyzer. For a sentence the predictive information is in the context cluster which the analyzer associates with that sentence. A context cluster is basically the bundle of predictions and structures, knowledge that can bind a text into a unit. The cluster has much the same theoretical role in the description of the analysis of the text, as the predictive word definitions had in the description of the analysis of sentences.

Besides the addition of the context cluster, the analyzer itself is extended. In extending the mechanisms of the analyzer itself, however, the original scheme of analysis remains. To the original analyzer have been added several mechanisms that extend the manipulative power of requests. One of the crucial directions extension takes is to allow requests to manipulate easily not only conceptual structures, as before, but other requests as well. Manipulation involves three actions: creation, modification, and deletion. As far as requests were concerned, the analysis of single sentences was concerned primarily with creation, with calling into play relevant predictions. With text analysis, the focus shifts to the mechanisms necessary for modifying and deleting predictions when they become no longer relevant. To do this requires not only various devices for modifying and deleting but also a definition of what relevance means.

Because the manipulation of requests implies the capability of recognizing what various requests do, much of the developmental work is on characterizing requests. The goal is to make it easy for one request to recognize another request. First, of course, it had to be determined exactly what information about a request was crucial for recognition, and what was not. As much as possible, I wanted to have a request be a black box in the eye of the analyzer, recognizable and manipulable through a small set of links that tied one request with another. The alternative to this would be to give the analyzer the capability of reading and writing the Lisp programs that the requests were written in. This would require an analyzer very advanced in the domain of automatic program writing. Further, it would contradict the intention of modelling human comprehension, unless I was postulating that people know how their own thought processes are encoded.

There are therefore three things that have to be described. One is the context cluster, the initial source of the information that is used to organize the analysis of texts. The second thing to describe is how requests in the analyzer can be characterized and manipulated. Finally, we can talk about how the information from a context cluster, through the manipulation of requests, leads to the analysis of a text.

Describing what context clusters look like means describing the kinds of predictions and the sources of these predictions that cause the analyzer to treat a sequence of sentences as a coherent text. These predictions alter the flow of decisions in the analysis of sentences. The alterations are intended to make the interpretations of the sentences of a text consistent with each other, in the same way that a verb in a sentence alters the interpretation of prepositions so that a consistent whole may be formed.

The next two sections are concerned with the kinds of alterations that occur when sentences are analyzed in context.

CHAPTER 11

CONTEXT

11.1 THE WORD "CONTEXT"

One dictionary definition of context is: "the parts of a discourse or writing which precede or follow, and are directly connected with, a given passage or word." (The American College Dictionary. Random House). This can be generalized to "the situation in which a given passage or word occurs" and thus include the non-linguistic elements that relate to a passage.

Context in this sense is a descriptive idea. We say that in different contexts one sentence can have different meanings, and that in different contexts there are different restrictions on the types of sentences that can occur with them. Context, used this way, is not an explanative concept. By that I mean that the specification of a context in which a sentence appears does not include any specification of what pieces of information from the context affect the understanding of that sentence. We say that a sentence has a certain interpretation in a certain context, but this doesn't tell us why it has that interpretation in that context. For example:

1) John and I were fishing. He caught one small trout. I caught one too.

2) John and I were very cold and wet that day. He got a bad cold. I caught one too.

3) John and I decided to stay. Hello.

The sentence "I caught one too" means something different in the first context than it does in the second. In (1) I take it to say that "I caught a trout also." In (2) I take it to mean "I caught a cold also." In the third example the phrase "Hello" seems out of place in the context and would probably indicate a break in the monologue. Notice that these statements do not say why the effects occur, but only that they do. The word "context" therefore commonly refers to the surroundings in which an utterance appears. Here, however, "context" will refer to the underlying elements that cause the contextual effects and restrictions, like the ones given above. A result of studying language comprehension rather than language patterns is this stress on underlying mechanisms.

However, before we can talk about what mechanisms underlie various contextual effects, we need to have a better idea of what kinds of effects can occur. The next section presents a number of examples of different kinds of effects.

11.2 CONTEXTUAL EFFECTS

There is no problem finding examples of contextual effects. In fact, it is much more difficult to find cases where such effects do not occur. Few sentences occur isolated from other sentences, and none occur apart from a non-linguistic environment.

One contextual effect is change of meaning. For example:

1) John and Mary were racing. They were afraid of being beaten.

2) John and Mary were running. They were afraid of being beaten.

Chapter 8 implies that the meaning for "beat" chosen by the analyzer was not the same for texts (1) and (2). In (1) John and Mary were afraid of losing the race, while in (2) they were afraid of being physically struck. It is not crucial theoretically if someone disagrees with this assumption as long as he believes there are cases of meaning change like (1) and (2). Changes of word sense is the focus of extended analysis, although it is more concerned with those that can't be explained by word-associations, as (1) and (2) might be.

Another contextual effect is change in the significance or intent of an utterance. For example:

3) It takes me 20 minutes to bicycle home. Can you go faster?

4) Driver, I'm in a hurry to get home. Can you go faster?

Even though the analysis of "Can you go faster?" should give the same conceptual structure in both (3) and (4), the intent of that sentence is different for each case. In

(3) I take it to intend "Are you capable of going faster than I do?" In (4) I take it to intend "Please go faster." For this reason it is satisfactory to respond with only a "yes" to (3) but not to (4).

Another contextual effect is anaphoric reference. For example:

5) John gave Bill a book. He returned it later that day.

The referents for "he" and "it" in the second sentence of (5) are presumably Bill and the book respectively.

Related to this is the use of the definite article "the" versus the indefinite "a". For example:

6) John gave Bill a book. The book was War and Peace.

7) John gave Bill a book. The spine was broken.

In (6) and (7) "a" is used in the first sentence to introduce the book. In (6) "the" is used in the second to refer to it. In (7) "the" is used with the "spine" to indicate that the spine of the book of the previous sentence is meant.

Ellipsis in utterances involves another contextual effect. A sentence can have a gap that should be filled with pieces of the preceding context. For example:

8) Whose house was on fire?--John's.

9) That house is on fire.--John's?

10) Did George say what kind of ice cream Mary likes?--George thinks vanilla.

Example (8) is a common form of ellipsis following a question that means "Tell me an X such that Y." The reply can be just a specification of the X without repeating the Y. Example (9) is an elliptical question given in response to a statement. Example (10) is one used by generative semanticists to argue against the traditional transformational view that grammaticality can be considered out of context.

There are obviously a wide variety of contextual effects. It is doubtful that any one simple mechanism can account for them all. Some of them are syntactic effects and some are conceptual. Examples of syntactic effects are the elliptical answers of (8) and (10), where surface structure is abbreviated by assuming the repetition of part of the structure of the question. Conceptual effects occur in (1) and (2) where the meanings of words change. We shall be concerned here with conceptual effects.

Furthermore, there is a basic contextual effect that motivates the rest. I am referring to the fact that the sentences in a text do not disappear, like the Cheshire cat, leaving only a smile to affect later analyses. Rather, in comprehending a text a single structure is built, a kind of super-sentence, to which each new sentence is added as it is understood. Part of the intuitive notion of understanding a sentence is this ability to see how the sentence relates to what has gone before and what is to come.

We will be concerned with contextual effects arising from information that is associated with the elements that appear in the text. This information is organized in what is called a _context cluster_, and the general nature of such clusters is the topic of the next section.

11.3 THE CONTEXT CLUSTER

The basic mec .nism used by the analyzer for handling context is the context cluster. A context cluster is a coherent group of interrelated facts and expectations oriented about a common theme. Theme is the unifying element (or elements) of a context cluster. However, outside of using words to label them, a particular theme probably can be steictly defined only circularly in terms of the concepts that are organized about it. An example of a theme is a "contest", which is the unifying element for concepts involving a group of people, each of whom is trying to do better on some fairly well-defined scale than anyone else in the group. "Contest" appears nowhere in this set of concepts but is instead a label for the theme of this cluster. These conceptualizations that express the theme of the cluster form the core of the cluster. While providing the "meaning" of a cluster, however, this core is a passive data structure. Hence it is not enough for dealing with effects upon a process like language analysis.

Associated with the core, and part of the total context cluster, are specifications of actions relevant to the context. For our purposes only those actions related to word meanings in particular and language processing in general will be important, but it is also in this area that we might place the response that a person has to being in a race, .e. instructions on how to run fast, and rules for deciding when the goal is no longer worth the effort.

For example, the meanings that are chosen for words like beat", "win", and "cheat" are affected by the context cluster of a "contest" when it is active. A context cluster has actions that change the meanings of words to make them more relevant to the theme of the cluster. In terms of a network model of memory the effect involved could be seen as lighting up or activating nodes in a graph.

This kind of effect was implemented and described in section 8.1. Every word has two pointers to possible senses of that word. One, called CURRENT, points to at most one element. It is a sense of the word that has been specifically predicted by the preceding context. The other pointer, called COMMON points to a list of senses for that word. These are senses that are usually recoverable after a moment's thought about the word out of context. When no sense has been predicted for CURRENT, an arbitrary element from this list is picked. It is not necessary, by the way, that CURRENT point to a sense that appears in the COMMON list. And it should be obvious that the list of common senses is a convenience and not a model of the organization or words in memory. The ordering of that list is also not of immediate concern. What is important is the presence of one sense predicted as being the current one, the one that should be considered before others.

Another action specification that can appear in a cluster is a request. That is, as part of the knowledge in a cluster, there can be a datum that says "if you see an X then do Y." This can be viewed as a generalization of the contextual lexicon which says "if you see the word X, then assume it means Y."

In the programmed implementation of text analysis described in section 8.1, requests attached to a context cluster were responsible for the conceptual predictions that the cluster led to. In extended analysis conceptual predictions are made by the monitor program based on the conceptualizations in the static core. However, there are still certain requests that belong to the action part of a cluster. These are called traps and are described in section 12.4. Monitor-generated predictions are of more interest to us here than the trap-generated ones. This is because the monitor makes predictions in order to achieve some goal. That is, there is a motivation for a monitor prediction. But a trap prediction says that certain situations happen to be handled in a particular way when this context cluster is involved. A trap prediction is not motivated by some overall purpose. For example, the monitor may predict that a human being will be referred to by some future noun phrase because the conceptualization being built needs a human. A trap prediction might say that "a rat" refers to a human in a given context, but this prediction is not made in response to any particular need for a human.

The context cluster is a device whose function is to provide the analyzer with world knowledge in usefully-sized chunks. Clusters are an hypothesis about one aspect of the cognitive process of finding relevant information. The hypothesis is that world knowledge is organized, in the long term memory, into clusters. When part of a cluster is perceived in some situation, then all of the information in that cluster is assumed to be relevant to understanding the situation. That is, this information is used to predict what will come next, and how everything ties together. A word usage that is peculiar to a cluster, or some concept that is central to a cluster, are reasons for activating that cluster.

The context cluster is not one of the main topics of this thesis. It is necessary to develop some idea of how world knowledge is presented to the analysis processes, but of more concern here is how that information is then used. For this reason only a sketch of context clusters is given and many major issues are left untreated.

For example, it seems reasonable that context clusters are highly organized internally. For our purposes the only organization is that of the story-pattern, where conceptualizations are arranged in narrative sequences. Presumably however these story-patterns themselves appear in larger structures, with some patterns serving as subparts of others.

Another issue is how to recognize when a cluster should be activated and wher de-activated. For our purposes, the use of associative links, from word senses and concepts to clusters, is sufficient for the activation of relevant clusters. And just as it was not necessary to worry about removing requests when only single sentences were being an-anlyzed, so too it is not necessary to remove context clusters when only short texts are being considered. However both of the processes, activation and de-activation are clearly important issues in the area of general cognitive mechanisms.

Another issue is how many context clusters there are, and, related to this, how they are. This, of course, is something that only experience will tell. The answers depend in part on what internal structures clusters are assumed to have, and on what, if any, relationships exist between clusters.

In summary, the context cluster is part of a theory of memory organization. As such it is not the main focus of this study of language analysis processes. Our interest here on processes leaves us less time to spend on data organization. Hence the context cluster as a mechanism is developed just enough to allow the description of extended analysis to continue.

In the section 13.2 "Contexts and Stories" more will be said about the conceptualizations that appear in the core of a context cluster. Also, this section begins the description of how these passive structures are converted to active language analysis effects. First, however, the mechanisms which extend the analyzer and which are used in this conversion must be described.

CHAPTER 12

EXTENDED ANALYSIS MECHANISMS

12.1 OVERVIEW OF EXTENDED ANALYSIS

Extended Analysis applies to the analysis of texts, as opposed to sentences. It starts with the prediction of conceptual patterns caused by the creation of some initial conceptualization during analysis. The prediction is done with the mechanism of a story-pattern. Story-patterns are sequences of conceptualizations stored in memory. They are part of a person's world knowledge. The recognition of one element in a sequence leads to the prediction of adjacent elements in the sequence.

Story-patterns appear in context clusters. A context cluster consists of related conceptualizations, a contextual lexicon which is a jargon associated with the topic matter of that cluster, and various instructions about what actions to do if certain situations relevant to that cluster are encountered. Subsets of the conceptualizations in a cluster form story patterns.

Conceptual patterns predicted from story patterns are compared against new conceptualizations as they are built during the analysis. When a conceptual pattern partially matches a conceptualization being built, it is predicted that the remaining sub-patterns to be filled in the conceptualization will match the corresponding ones in the pattern.

The real task, however, is to convert these predictions of conceptualizations into ones about language constructions. The mechanism for doing this is the chain. At any point in the analysis of a sentence, there is a set of requests waiting to be triggered. Each of these requests specifies a situation and an action which will be executed if that situation becomes true. Some situations depend directly upon the appearance of some word or word sense. Others, however, become true only after one or more other requests are triggered first. That is, the situation of one request depends on the action of a second one. The situation of the second one may in turn depend on the action of a third, and so on. This string of dependencies, from situation to action and back, is called a chain. Eventually, during the analysis, every request that could be triggered must belong to a chain ending in a dependency on a word or word sense situation. Otherwise no language event could ever trigger that request.

The analyzer starts therefore with predictions of conceptual patterns or sub-patterns. These lead to preferences for any requests whose execution produce those patterns. When a request is preferred the situation that triggers that request is preferred. But a situation may depend on the actions of other requests. Therefore these requests are themselves preferred. In other words, the initial conceptual prediction leads to a preference traveling down the chain of dependencies that exist between requests. At each stage those requests are preferred that would trigger the request already known to be preferred. Ultimately preferences travel to the level of word and word sense situations. Word sense preferences are the basis of the kinds of contextual effects that extended analysis deals with.

For example, suppose a prediction of the following conceptualization has been made:

$$\begin{tikzcd}
 & & & & \square \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{ATRANS} & \square \arrow[l, "o"'] & \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

Then suppose the sentence "John gave Mary a sock" is analyzed. When "gave" is encountered several requests are added to the set of those waiting to be triggered. One of these says that if a noun phrase referring to a physical object is found, then an ATRANS conceptual pattern will be built. Because this pattern is the preferred one, this request is preferred. Therefore the situation that would trigger this request is preferred. The situation that triggers this request is given by the TEST portion of the request. Therefore there is a preference for a noun phrase referring to a physical object. This preference is at the language level. This preference means that when a noun phrase is built, while this preference is in effect, if the main noun can refer to a physical object, then that sense of the noun will be chosen. Thus, in "John gave Mary a sock," "a sock" can and hence will be interpreted as an "item of footwear". This causes the ATRANS building request to be triggered and the initial conceptual prediction to be satisfied.

Central, therefore, to extended analysis is the ability to find these chains of dependencies. This is accomplished by isolating and explicitly specifying two pieces of information for each request. One is called the _need_ of a request, and the other is called the _focus_.

Both the need and the focus of a request are given by a role, i.e. some position in a structure, conceptual or linguistic. Every request looks at the substructure in some position in a structure, and, if the request is triggered, puts a new substructure in some position in a structure. The particular need that a request has is that role which the execution of the request causes to be filled. The focus of a request is that role which the request looks at. It is possible that a request may depend on a conjunction of situations, in which case the focus may be a list of roles.

Needs and foci are used in the conversion of preference from the concept level to the word level. The need gives information about the action portion of a request. The focus gives information about the test portion of a request. Together these two provide the information needed to find the chains of request dependencies.

The key rule is this: a request depends on those requests whose need equals its own focus. The focus of a request is that role upon whose value the triggering of the request depends. Any other request with a need equal to that focus can change the value of that role, by the definition of need. Hence this other request can affect the triggering of the first request. Therefore the first request depends upon the other.

For example, suppose there is a request whose action is to give a value to CONCEPT. That means the need for this request is CONCEPT. Suppose further that this request is triggered when OBJ has a certain feature. That means the focus of this request is OBJ. Hence this request depends on each request that gives a value to OBJ, i.e. whose need is OBJ. For most verbs, requests filling OBJ depend on some feature of SENSE. That means that the focus for such a request is SENSE. So we now have chains of dependency that go from the CONCEPT request to each OBJ request, and from each of those to SENSE, i.e. the language level.

Need and focus have independent justification for their existence, beyond their use in chaining. In particular a need specifies the purpose of a request, and when that purpose is no longer viable the request can be removed. A focus specifies what a request is affected by, and only when the focus changes in value is it necessary to see if the test of the request has become true.

The need mechanism is used to regulate what requests are active at a given point in an analysis. This regulation is based on the gaps that exist in those structures that the analyzer is building. The absence of information in some structure can be taken care of only by those requests that affect the value of that role. As long as the gap remains, these requests are kept waiting, for the purpose of filling it. And as soon as that gap is filled, the reason for keeping these requests disappears and so, therefore, do the requests, even though the requests may never have been triggered. The need of a request is used, therefore, to link a request with the gap in a structure that is the motivation for retaining that request. The next section gives an example of how this works.

The focus does not have so drastic an effect but it does allow the monitor to handle the testing of requests in a more efficient manner. When a role is given some value during analysis, it is not necessary to check the entire set of requests to see which ones have been activated. Instead, only those requests whose focus equals the role which was just given a value need to be considered, because only these requests are affected by that role. Further, since executing a request affects only the role specified by the need of that request, only those requests whose focus equals the need of a triggered request need to be looked at next. In other words, the checking and execution of requests proceeds up the chain of dependencies down which preferences were passed.

Exactly how the request chains are formed, and how preferences are passed along them, as well as details about the nature of the need and focus of a request form the content of the discussion of extended analysis that follows.

12.2 A FLOW TABLE FOR EXTENDED ANALYSIS

As a guide to the reader, this section presents a table of the flow of operations in extended analysis. The table is similar to the one given in section 2.4 ("Overview of the Analysis Process"), but several notational changes were required. Several new columns were added and these necessitated the use of letters and numbers to refer to predictions and requests. What these symbols stand for is explained in the comments after the table.

The new columns are "Predictions Active" and "Needs Active". The predictions that are in effect at each step in the analysis flow are listed under "Predictions Active". Listed under "Needs Active" are the gaps waiting to be filled at each step in the analysis, followed by the numbers identifying the requests that have been suggested to fill this need. Also listed here are traps, i.e. requests not directly attached to any particular need. Traps will be discussed in section 12.4.

The text analyzed in the table is "John hated Mary. John gave Mary a sock." Only the second sentence is traced because the conceptual predictions A and B are made after the analysis of the first sentence.

The definition of "give" is modified in a minor way to demonstrate the need mechanism better. Rather than having a request that changes the sense of "to" to TO1, there is instead a request that looks for "to" and performs the same actions that TO1 would.

STEP	WORD READ	PREDICTIONS ACTIVE	NEEDS ACTIVE	REQUESTS TRIGGERED	ACTIONS TAKEN
Ø	none	A,B	SUBJ-1 CONCEPT-	none	none
1	John	A,B	SUBJ-1 CONCEPT-	1	Choose SUBJ to be John Remove need for SUBJ
2	gave	A,B	CONCEPT- 2,3 RECIP- 4,5	none	Remove B Add C based on A Add D bas[illegible] on C
3	Mary	A,C,D	CONCEPT- 2,3 RECIP- 4,5	4	Choose RECIP Remove need for RECIP
4	a	A,C,D	CONCEPT- 2,3 trap I	I	Set aside needs and the predictions A and C Add trap II
5	sock	D	trap II	none	Choose the action sense of "sock"
6	period	D	trap II	II	Build NP "a sock" Reset needs and predictions Remove D
7	none	A,C	CONCEPT- 2,3	3	Build CONCEPT to be John hit [illegible]y Remo[illegible] C
8	none	A	none	none	Remove A

Comments:

Predictions:

A-predict CONCEPT is John will hurt Mary
B-predict CONCEPT is Mary had hurt J[illegible]hn
C-predict request 3 will be used
D-predict a NP will refer to an event

Requests:

1-if there is an NP, choose SUBJ to be the NP
2-if an NP is a POBJ build CONCEPT to be giving the NP
3-if an NP is an event, build CONCEPT to be that event
4-if an NP is human, choose RECIP to be the NP
5-if there is a "to", choose RECIP to be the NP following

Traps:

I-if true then start building an NP
II-if WORD ends NP, then build NP and reset everything

Traps will be discussed in section 12.4. Basically they are requests generated to start new structures rather than fill the needs caused by an existing structure.

In Step Ø we see the initial state of the analysis. There are two needs, one for SUBJ and one for CONCEPT. A suggestion has been made on how to fill SUBJ, but none exist for CONCEPT.

Also there are two predictions about the future value of CONCEPT. Prediction A says that CONCEPT will be about John hurting Mary. Prediction B says that CONCEPT will be about Mary having hurt John. These two predictions are made on the basis of the analysis of "John hated Mary." The understanding of this sentence causes two predictions to be made in order to fill in two different gaps in a larger structure. The larger structure, to which the conceptualization underlying "John hated Mary" belongs, is a story-pattern about how people behave. One gap in this pattern is a reason for why John hates Mary. B predicts that this gap will be filled. The other gap in the pattern is what follows from John hating Mary. A predicts that this gap will be filled.

In Step 1 nothing happens that didn't happen in the original analyzer. "John" is chosen as the SUBJ and so the request for SUBJ disappears. The reas n this request disappears is slightly different, though. In the original analyzer it was removed because it had been triggered. Here it is removed because there is no longer a need for a SUBJ filling action.

In Step 2 the verb "gave" is associated with two suggestions for filling CONCEPT. These are added to the REQUESTS list. Also a need for RECIP is added, plus two suggestions on how to fill it. Further, because "John gave" means that any value of CONCEPT to be built will have John as the actor, prediction B can no longer apply and it is removed. Prediction A is still possible if request 3 is executed. So the monitor makes prediction C which prefers request 3. In order for request 3 to be executed, a NP must occur referring to an event. Therefore the monitor makes prediction D, which prefers a NP referring to an event.

In Step 3 the RECIP need is filled by request 4. Note that this means that both request 4 and request 5 are removed, even though 5 is never triggered.

In Step 4 a trap is encountered. This means that "a" does not itself fill any needs, but rather starts some separate actions going. In this case the building of a noun phrase is begun. Everything is set aside except prediction D. Prediction D is kept because it applies to noun phrases.

In Step 5 prediction D causes the hitting sense of "sock" to be chosen.

In Step 6 the trap that completes the noun phrase is triggered and "a sock" is built. Prediction D is removed because it has been used successfully on this noun phrase. The rest of the predictions and needs are restored.

In Step 7 request 3 is activated, thanks to the success of prediction D in picking a sense for "sock". This fills the need for CONCEPT so both requests 2 and 3 are removed. Also prediction C is satisfied and removed.

In Step 8 prediction A is satisfied by the final value of CONCEPT and so it is removed. The importance of this step is that if another sentence were to follow, prediction B could still be tried, but not prediction A.

We have of course left out numerous details in this chart. The building of the structure for CONCEPT involves the use of needs and suggestions, none of which were listed above. The basic flow of analysis however is as above. There are two major changes from the flow given before in section 2.4. One is the use of needs to control the set of requests active. The other is the maintaining of a set of predictions that affects which requests are executed.

The way needs, predictions, and requests interact should be kept in mind as each of the elements is described. Although each element can be justified separately, of greater interest is the fact that they work together so closely in the total process of comprehension.

12.3 NEEDS AND EXPECTATIONS

A basic concept to be added to the analyzer is called the need. Every expectation is associated with some need. When the action associated with an expectation is performed, the need disappears. At the same time, when a need disappears, all the expectations (and their associated actions) associated with that need desappear as well. A need, in other words, is the reason for the existence of an expectation. The fact that an expectation causes itself to disappear once the situation it predicted is encountered is a special case of why expectations disappear in general. What is important about the concept of a need is the fact that triggering an expectation is no longer the only way in which an expectation might be removed.

As an example of what is meant by a need, consider the following conceptual structure:

$$\begin{tikzcd}
 & & & & \text{*} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{BOOK} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

The "*" marks a gap in the structure, a place where there is a need for information. The above structure is created during the analysis of the sentence "John gave a book to Mary" after the phrase "a book" has been comprehended. The need for a recipient has two requests associated with it. One is on the list of requests for "give":

```
((FEATURE SENSE (QUOTE HUMAN))
 (CHOOSE RECIP SENSE) NIL)
```

The other request associated with this need is on the request list for the current sense of "to":

```
(T (CHOOSE RECIP SENSE) NIL)
```

So far nothing has changed from the description of the analysis given before. The same structure has been built and the same requests are active. When the phrase "to Mary" is comprehended, however, something new happens. The request for "to" fills the gap in the conceptual recipient case. The request for "to", as before, thereby disappears (in the sense that its TEST will not be looked at again). But now the request from "give", the one with the TEST "(FEATURE SENSE (QUOTE HUMAN))" also disappears, because its need is fulfilled. Previously this request would still be present. In the sentence "John gave a book to Mary" this is not harmful because the sentence ends immediately, and all the requests disappear. However, there are cases where the ability to remove expectations when they are no longer necessary is important. The texts discussed in section 8.1 are examples of this, and they shall be discussed again later.

The need for a concept to fill a gap is one of several types of needs. Some of the sources of needs are:

1) A word construction needs a word, with a particular feature, to fill a gap;
2) A conceptual frameneeds a concept, with a particular feature, to fill a gap;
3) A sequence of conceptualizations needs a conceptualization, with a particular feature, to fill a gap.

Section 12.4 ("Internal versus External Needs") will expand this list of sources.

To incorporate this new mechanism, the structure of a request in the analyzer is modified. No longer does the variable REQUESTS refer to a simple list. Instead, it is a list of lists. Each of these lists is associated with one need. These lists specify the requests that are relevant to that need. This has a two-fold effect on the analysis:

1) Requests come and go in groups, according to the needs they are relevant to. Hence it is possible for a request to disappear without being activated;
2) Requests are separated and classified according to the needs they satisfy. This allows some requests to recognize others and thereby increases the manipulability of requests.

12.4 INTERNAL VERSUS EXTERNAL NEEDS

When the analyzer was being described, it was noted that words within noun phrases were not handled as they were in the rest of the sentences. Normally the monitor read in a word, checked the request list, loaded the requests that made up the sense of that word, and finally rechecked the request list. Within a noun phrase, however, the requests were not loaded from the word sense. A noun phrase was built using only the passive features associated with the senses of the words involved. To do this, noun phrase introducers had a request that changed the value of ACTIVE and this changed the way the monitor program ran. It was not possible to get the necessary effect solely with requests.

There were two disadvantages to this solution. One was that it was ad hoc and unaesthetic. The other was that there was a failure to keep all the analyzer's knowledge about language in the dictionary. The information about how noun phrases were built was hidden within the monitor control structure. One justification for the request formalism was that it allowed almost all that the programmer intended to say about analysis to be written in a uniform and highly visible way.

With the concept of a need, a motivation for the way in which noun phrases are handled can be given. Central to the explanation is the distinction between internally generated needs and externally generated ones. By an internally generated need, I mean one that is generated to fill a gap. An example of an internally generated need is one generated by a gap in the TO position of an ATRANS structure. Externally generated needs are those generated in response to the discovery, by the analyzer, of something in the input. An example of an externally generated need is one that calls a backup routine when the features of an OBJ of a verb contradict what that verb requires.

The needs listed in section 12.3 ("Needs and Expectations"), are all internally generated needs. Some structure, either of concepts or of words, in order to be completed, requires more information, and this requirement is an internally generated need. Consider, however, the sentence (suggested to me by J. P. Paillet), "John drinks his coffee cold." There is nothing in either the syntactic structure built from "John drinks his coffee" nor the conceptual structure that is the interpretation of "John drinks his coffee" that requires the information provided by "cold". The adjective is not expected, and there is no internally generated need it fulfills. Hence there is no request activated by the discovery of "cold". There is, however, an overall desire to understand the input, and hence there is a need to tie this word in with the rest of the sentence. The failure to trigger requests, in other words, leads to a need to use this input that no one wanted. At this point the analyzer looks to a list of suggestions that it has that are concerned with externally generated needs. This list of suggestions is part of the analyzer's general knowledge about language processes, tied not to structures, but rather to the need for structures. These background suggestions are called _traps_.

"Cold" is not the only word in the above example that is not prepared for by the requests of the analyzer as it stands. Both "drinks" and "his" are also unlooked for. That is, no requests are triggered when these words are encountered. Hence a need arises to use each one. This need to use a word is taken care of by loading the requests of the sense associated with the word. These requests provide the action necessary for making use of the words involved. Thus the requests for "drinks" provide the basic information for connecting "drinks" to the main clause. The requests for "his" provide the information that "his" is starting a noun phrase. In both cases the requests were loaded because of a need to make use of the words seen. One of the background suggestions therefore says that the sense of an unexpected input word may provide the actions necessary for tying this input to the rest of the sentence.

In contrast to "drinks", "his" and "cold", the two words "John" and "coffee" are already taken care of by internally generated needs. "John" is looked for by the initial SUBJ request. "Coffee" is looked for by the noun request generated by the noun phrase structure begun by the requests associated with "his". Because "John" and "coffee" have uses already prepared for them, the monitor does not need to load any requests they might have.

This, then, is why the analyzer treats words within a noun phrase in a different fashion from those outside a noun phrase. Words outside a noun phrase are external events and something (perhaps the words themselves) must provide the information necessary for using these words. Words within a noun phrase have roles already prepared for them, and thus no needs to incorporate these words are generated.

12.5 TRAPS VERSUS NEEDS

The term trap is used for those routines called by an unexpected event. Examples of traps are the routines described in the last section, that try to find a use for a word that is encountered which satisfies no expectations. Backup routines, called when a contradiction is encountered, are also traps.

Traps can be loosely classified as routines that start with pieces and attempt to make a whole from them. The routines associated with needs, on the other hand, start with an incomplete whole and attempt to find pieces which can be used to complete it.

Needs have the property that it is easy to regulate their coming and going. Given a particular structure, and a set of syntactic rules for such structures, it can be determined what, if any, gaps exist. When a gap is found, a need to fill that gap is generated and when the gap is filled, the need disappears. For this reason, needs are helpful in controlling what routines or requests are active at any one time.

Unfortunately, traps do not begin with well-defined structures but with smaller elements, for which a structure must be found. How then should traps be regulated?

One possibility is that they shouldn't be, that the set of traps should be the same at all times. This has several disadvantages, however. One is that it means that some traps will probably be irrelevant but active in some situations. That is, there may be a routine for handling an unexpected event that succeeds in some cases but could not possiblysucceed in others. With this solution the programmer cannot use information about the current situation to regulate what traps should be activated. Further, this solution leads to a fairly restricted set of traps, to routines for handling fairly general problems. These are traps concerned with the language itself, such as the fact that an adjective can occur free in "John drinks his coffee cold." Hence, traps are not readily available to the programmer for dealing with specific contexts. For example, when listening to commentary on a baseball game, there is a trap for sentences like "Ball two" that immediately knows it must tie this in with a structure of the state of the game, not with the text of the commentary, which may be about the home life of the catcher.

Another possibility, therefore, is to associate traps with the recognition of particular situations. In other words, some traps are part of a context cluster, as described in section 11.3 ("The Context Cluster"). Other traps might be associated with situations like "currently in a noun phrase" or "currently in a clause" that describe the syntactic state of the analyses. Traps, therefore, would come and go along with the needs associated with these contexts and situations. The difference between traps and needs would remain, however. A need routine is attempting to fill out a structure, but a trap routine is either building a new one or modifying an existing one. Further, a need can disappear while the cluster that generated it is still present. A rap, however, remains until the cluster is removed.

However, the focus of study here will be needs, hence the first assumption about traps, that the set of them is constant, will be made. Only enough traps to allow a description of analysis with needs will be used, and a more det led study of them will not be made here.

12.6 THE FOCUS OF A TEST

The need is the crucial extension to the request format. It isolates that aspect of the action of a request that allows the analyzer to decide whether the request should continue to exist as is, or should be modified or deleted.

However, the action is only one of the two parts a request specifies. The other part is the test. It turns out that it is also necessary to isolate for the analyzer a piece of information about the test as well. Suppose the analyzer has decided that it would prefer a certain request to be executed. It found this request by having a preference about how some role should be filled and looking at those requests whose action affected this role. The need of a request specified what role its action affected. Knowing that it would like this request to be executed, the monitor then wants to know what situation would cause the request to be triggered. The triggering situation is specified by the test of the request. Hence if the monitor can tell what role the test predicate looks at, then the monitor can shift its focus to those requests that fill that role.

For example, suppose the monitor would prefer that the value of CONCEPT be set to an ATRANS action. Looking at the requests for the need CONCEPT, it finds one whose action creates an ATRANS structure. Suppose the test of this request says that the OBJ, of the verb of the sentence being analyzed, must be some kind of money. Knowing that the predicate is about OBJ, the monitor can then go to those requests that fill OBJ and see if there is one that creates a filler that is a kind of money. If such a request exists, it then prefers this request to be executed. Now the monitor is concerned with what triggers this request and the cycle repeats. In what ways this chain can end will be described later.

The information that needs to be extracted from the test portion of a request, called its focus, is the role (or perhaps the list of roles if the test is a conjunction of predicates) which the test predicate depends on. For a test like "(FEATURE (CHOICE SUBJ) (QUOTE HUMAN))", the focus is SUBJ, because it is the value of SUBJ which determines whether the test is true or not.

It is not necessary to classify requests by the foci, as we classified them by their needs. This is because a request is first accessed for manipulation through its need. Having the request may lead to using its focus to specify the need of the next request to be looked at. There is a situation where collecting all requests with a certain focus would be useful. A SUBJ is filled, for example, and only those requests with SUBJ as their focus need to be tested to see if their predicates have become true. One could use classification by focus then to avoid evaluating those predicates that have nothing to do with SUBJ. This, however, is only a useful side-effect of specifying foci of tests. The fact that chains of requests can be formed with them is of more importance.

The next section gives the formal structure used for specifying needs, suggestions, traps and foci.

12.7 FORMS FOR NEEDS AND REQUESTS

Having spent some time discussing some of the properties of needs, we can now see how requests look that include this information.

The list of requests attached to a word and the list REQUESTS, which is kept by the monitor, have an expanded form. Formerly both had the structure (R_1 R_2...R_N). R_i was a request of the form (T_i A_i F_i), where T_i was the test predicate, A_i was the action and F_i was the flag. The expanded form is (S_1 S_2...S_N) where S_i is a need plus a list of requests. That is, S_i=(N_i R_{i1} R_{i2}...R_{iN}) where R_{ij} is a request and N_i is a need. A need is a Lisp predicate. A request has the form (V T A) where V is the variable that T focusses on, and T and A are the test and action as before. The flag is no longer necessary for removing a request after its execution because the filling of the need by the request's action automatically causes the disappearance of that request.

The list S_i has a different interpretation when attached to a word than it has when appearing in REQUESTS. In the former case, S_i is read as "If the need N_i exists, then (R_{i1} R_{i2}...R_{iN}) is a list of requests whose actions are relevant to filling this need." When S_i appears in REQUESTS, however, it is read as "The need N_i does exist, and (R_{i1} R_{i2}...R_{iN}) is a list of requests, whose actions may fill this need, that have been suggested so far." An S_i, when attached to a word, will be called a <u>suggestion</u> of ways to fill a particular need.

The monitor, during analysis, maintains the list REQUESTS. This list contains those needs currently active plus requests relevant to those needs which have been found so far. When a new list of suggestions is encountered, such as the sense of a new word, the need portion of each suggestion is checked against the list of needs active. If the need portion appears in this list, then the list of requests that follow is appended to the list of requests that have previously been suggested for that need. If the need portion of a suggestion does not appear in the list of actual needs, the suggestion is ignored. That is, suggestions for non-existent needs have no effect.

The needs are generated either by the initialization routines of the monitor or by the action portions of triggered requests. Thus, for example, there is initially a need for a SUBJ and a suggestion that says "If you need a SUBJ, take the first noun phrase." When a request builds a structure in CONCEPT, it also adds a list of needs and requests for completing that structure.

At this point, we must clarify what it means to say that a need portion appears in the list of actual needs. The list of actual needs contains specifications of gaps that are relevant to the particular structure being built at the moment. A suggestion made by the sense of a word will, however, be intended for handling a general class of structures, of which the one being built is a particular instance. For example, the monitor may have the need "($\Leftarrow$ ACTOR)", that is, the actor of an event caused by another event is not filled. A word may have a suggestion for filling a need for an object that is directly affected by some action, which could mean either the OBJECT of the act or the ACTOR of the event caused by the act. We shall see specific uses of this kind of general need specification in section 12.8 ("Example Definition Using Needs").

Because the need portion of a suggestion is apt to be more general than the needs actually existing on REQUESTS, a need portion is said to appear in REQUESTS when there is a need which is a special case of that need portion. Situations where more than one existing need is an instance of a need portion have not arisen in the examples studied. The best way of resolving such occurrences is therefore left open until more is known about them.

Three more comments about the analyzer extension remain. First, the list of expanded request forms is treated like the former list REQUESTS with respect to functions like IMBED. The list of new forms, also called REQUESTS, may be temporarily set aside in favor of another. Second, just as there was a function ADD_REQ that allowed requests to add more requests, so there is a function ADD_SUGG that allows requests to add new needs, and requests for those needs. ADD_SUGG is the main function used for generating new needs during the analysis. Third, senses can still have requests that should be executed immediately, regardless of the presence of a need. The word DO in the need field of a suggestion tells the monitor to evaluate the action following immediately.

12.8 EXAMPLE DEFINITION USING NEEDS

We now present a revised version of the first sense of "give":

GIVE1:

```
(CONCEPT      (this signals a need for a value for CONCEPT
  (SENSE      (this is the focus of the first suggestion)
   (FEATURE SENSE (QUOTE POBJ))      (the test and action of
   (REPLACE CONCEPT                   the first request are
            (QUOTE                    as before)
             ((ACTOR (NIL)      (ATRANS) TO (   RECIP)
               FROM (    ACTOR) OBJECT (   OBJ))
              MODE (NIL) TIME (NIL))))))

  (SENSE      (this is the focus of the second suggestion)
   (FEATURE SENSE (QUOTE EVENT))    (the test is the same)
   (PROG NIL (but the actions are different from before)
         (REPLACE CONCEPT (UTILIZE (NORMAL_MEANING SENSE)
                                    NIL))
         (ADD_SUGG  (this adds a suggestion to REQUESTS)
           (HAPP_TO  (this is a new need - HAPP_TO see below)
             (RECIP  (this is the focus of the only suggestion)
               (FEATURE (CHOICE RECIP) (QUOTE PP))
               (CHOOSE HAPP_TO (CHOICE RECIP))))) )))

(DO (ADD_SUGG   (this adds a need and suggestion to REQUESTS)
      (RESP_FOR (this is the added need - RESP_FOR see below)
        (SUBJ (FEATURE (CHOICE SUBJ) (QUOTE HUMAN)) (this is the
              (CHOOSE RESP_FOR (CHOICE SUBJ)))))) suggestion)

(DO (ADD_SUGG   (this adds a need and suggestion to REQUESTS)
      (RECIP    (this is the added need)
        (SENSE (FEATURE SENSE (QUOTE HUMAN)) (this is the new
               (CHOOSE RECIP SENSE)))))       suggestion)
```

The last three requests use the form (DO (ADD_SUGG X)) where X is in need format. This forces monitor to add that need (and the suggestion) to REQUESTS. If only X were used then the suggestion would have been added to REQUESTS only if the need referred to by X already existed.

The first of these need-adding requests, the one for

RESP_FOR, is interesting because, in a normal active declarative sentence using "give", this request causes a RESP_FOR to be filled before CONCEPT is given a value. RESP_FOR (which will be exactly defined shortly) points to either the ACTOR of CONCEPT if CONCEPT is a simple action or to ACTOR of the first action if CONCEPT is a causal. The fact that RESP_FOR is filled as soon as "give" appears means that no matter what value CONCEPT takes on, there must be a position for RESP_FOR. If there isn't, then a trap routine is called that modifies CONCEPT in a simple way. Namely the old value of CONCEPT is replaced by:

$$\begin{tikzcd} \text{ONE} \arrow[rr, Leftrightarrow] & & \text{DO} \\ & \text{CONCEPT} \arrow[u, Rightarrow] & \end{tikzcd}$$

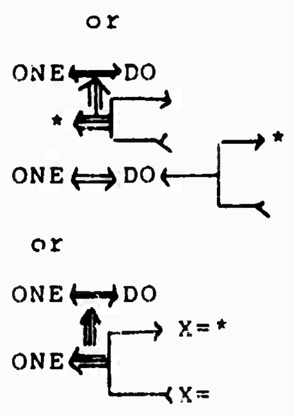
The trap routine may also have to modify CONCEPT to fit the rules of conceptual syntax. Basically, if CONCEPT is a state, then it cannot appear as the effect of a causal. Instead a state change, whose final state is the original value of CONCEPT, is used. Using this approach it is no longer necessary to have three requests to handle "John gave Mary a push," "John gave Mary a scare," and "John gave Mary a headache," even though the first is a simple action (PROPEL), the second is a causal (doing something causing fear), and the last is a physical body-state. Now, with general role specifiers, like RESP_FOR, and observing conceptual syntax rules, the few requests shown are all GIVE1 needs.

The first request under GIVE1 has two suggestions for filling CONCEPT. The main difference between these suggestions and the original definition is that each suggestion fills only the gap for CONCEPT and perhaps some of the gaps in the structure built for this filling. Therefore the first request does not fill OBJ at the same time as it fills CONCEPT. This is done by the third request under the need OBJ. The first request also does not fill in the ACTOR of the ATRANS, because the RESP_FOR request is already prepared to do this.

RESP_FOR, HAPP_TO, and RECIPIENT (not used in this example) are three general (as opposed to particular) need descriptions. Each of these is a match for several possible conceptual structures, all of which are considered equivalent by the suggestions using them. RESP_FOR is short for "responsible for". In a conceptualization it refers to the actor of an act, if the conceptualization is a simple action, or to the actor of the act that caused some other event, if the conceptualization is a causal structure. HAPP_TO is short for "happens to". In a conceptualization it refers to the object of an act, if the conceptualization is a simple act, or to the actor of the state change that is caused by an act, if the conceptualization is that kind of causal structure. RECIPIENT refers to the recipient of an act, if the conceptualization is a simple act, or to a terminal state in a state change caused by an act, if the conceptualization is that kind of causal structure. Graphically these definitions are:

DESCRIPTOR	GRAPHIC POSSIBILITIES
RESP_FOR	$* \Leftrightarrow \text{DO}$
	or
	$\begin{array}{c} * \Leftrightarrow \text{DO} \\ \Uparrow \\ \text{X} \end{array}$
HAPP_TO	$\text{ONE} \Leftrightarrow \text{DO} \xleftarrow{o} *$

RECIPIENT



The "*" in each graph is the point specified by the descriptor.

The advantage of a general descriptor can be seen in the second suggestion for the need CONCEPT. There HAPP_TO is used to find a place for RECIP in the conceptual structure provided by SENSE. SENSE might be either a simple action or a causal but HAPP_TO allows us to simply specify where RECIP should go in either case. The same advantage is true for the use of RESP_FOR in the second request.

Several items were left out of this description. First, there should be another DO request to, as before, set the preposition "to" to initially specifying the RECIP. Also, it must be remembered that the monitor will initialize REQUESTS to several needs, notably CONCEPT (and attendantly a TIME for the conceptualization) and SUBJ. In talking about request manipulation during text analysis, these items will not be important.

CHAPTER 13

THE EFFECT OF KNOWLEDGE ON ANALYSIS

13.1 TEXTS AND NEEDS

None of the above is meant to reduce the power that the analyzer had using simple requests. Instead more information is being added to the requests to increase their power. But the power of this extension will not be seen in definitions like GIVE1 as used in single sentence analysis, but from a consideration of the problems of handling texts of more than one sentence. How then does the mechanism of need apply to the analysis of text?

First, the need mechanism means that text analysis does not require two separate request lists. In the flow of analysis described in Chapter 8, one list disappeared at the end of a sentence and the other did not. With the need mechanism, only one list needs to be kept. When a sentence ends, many of the needs disappear and take their requests with them. This is because when the end of a sentence occurs, trap routines are triggered which fill in the various remaining gaps of the CD structure that is the interpretation of the sentence. With these gaps filled, the associated needs disappear. There are, however, as shall be described in this and the next sections, needs which are not concerned with gaps inside structures built by the analysis of the sentence. These therefore remain, except for those filled as a result of the analysis of the sentence.

Initialization of requests at the start of a sentence can now be viewed as a trap, triggered by the occurrence of a word with absolutely no structure to tie it to. That is, everything from the previous sentence has been closed and when a new word is seen, a trap routine, trying to find a use for the word, decides it is starting a new sentence. Therefore it creates the CONCEPT and SUBJ gaps and suggests that SUBJ be filled by the first noun phrase. There are two advantages to handling initialization this way. First, it moves more language data from the monitor to external data. Second, it makes the initialization more accessible for modification by previous analysis. This is not a crucial point at this stage, however.

Needs also provide the motivating element in the analysis of texts. A text, I claim, has certain needs, generated by:

a) the occurrence in the text of certain sentences and their interpretations;

b) the desire to unite these sentences and structures into one coherent whole.

That is, a text has the same relationship to its sentences that sentences do to their words. In the latter case the sentence was seen as being the result of actions and predictions, initiated by some words and concepts, that tied together the other words and concepts. The next section begins the description of the needs and actions that occur at the supra-sentential level that perform the same role of forming a coherent whole.

13.2 <u>CONTEXTS AND STORIES</u>

A context cluster is a coherent set of thoughts about some topic. Some of these thoughts are passive beliefs and are represented with CD graphs. Others are active instructions for behavior. These instructions are of two types. One type involves the activation of contextual lexicons, of jargons for the topic involved. The other type involves the making of predictions about things that might be seen next and how to treat them. These instructions are the trap routines associated with that cluster.

In the extended view of analysis, there is a close relationship between passive conceptualizations that appear in a cluster and active conceptual predictions. This is accomplished through the use of needs and <u>story-patterns</u>.

A story-pattern is a sequence of conceptualizations. These conceptualizations are descriptions of events, and these events are tied together by relationships of causation, instrumentality, and simple chronological sequencing. A story-pattern, then, is a subpart of a context cluster. Clusters may involve a number of such patterns. For example, the context cluster for hunting involves stories about travelling, stalking, killing, returning, and so on.

The use of the term "story-pattern" is meant to give an intuitive feeling to the kinds of structures being postulated. The setting of boundaries between sequences of conceptualizations and the labelling of them as stories is not a phenomenon that is assumed to occur in the mind of a human. What is being claimed is that certain conceptualizations, when recognized, lead to predictions that certain other conceptualizations will be encountered soon. The intuitive idea of a story emphasizes a situation where such predictions are fairly strong, that is, where sentences are not treated as isolated from each other but as being intimately related. Our treatment of sentences implied the same motivation in assuming that "John beat Mary" was not a simple sequence of words like "ball stick band" but a coherent whole.

But the prediction of conceptual structures from other conceptual structures is not enough. These predictions are of static conceptual patterns. For analysis to make practical use of these patterns they must be converted to active predictions about the input that can guide the course of language analysis.

Needs and foci are used to convert the predictions of a static story-pattern in a context cluster to active effects on text comprehension. When a text, T, is following a story-pattern, S, we mean that, so far, if the story-pattern S has the sequence of conceptual structures:

$$A \Leftarrow B \Leftarrow C \Leftarrow D$$

then the interpretation of T has generated some subsequence, such as:

$$* \Leftarrow B' \Leftarrow C' \Leftarrow *$$

where B' and C' are considered to be instances of B and C. Thus, for example, the single sentence text, "John hated Mary," can be said to be following the story-pattern:

Person-1 do something bad to Person-2 *causes*
Person-2 not like Person-1 *causes*
Person-2 do something bad to Person-1.

The interpretation of "John hated Mary" is an instance of the second line of the story-pattern.

If a text T is a subsequence of a story-pattern S, then there are needs to extend T to include elements of S that are not yet present. In particular, if T and S have the abstract form from above, then there is a need for each of the *'s. The left * is a gap with a need for an instance of A and the right * is a gap with a need for an instance of D. Notice that this is not a restriction on the order of story presentation in the text. All it predicts is that the next line of the text might satisfy one of these two needs. In the single sentence text, "John hated Mary," there are two needs generated after the analysis of that text. One is for a specification of why John hated Mary (first line of the story-pattern) and the other is for a specification of what John will do in retaliation (third line of the story-pattern).

Further, these needs have suggestions for filling them, based on the static information that the story-pattern presents. In the abstract case of T and S, these two needs for the left and right *'s are:

```
        need reason for B' (i.e. ? ⇐ B')
             suggest instance of A
        need result of C' (i.e. C' ⇐ ?)
             suggest instance of D
```

For the text "John hated Mary" this means:

```
        need ? ⇐ John hate Mary
             suggest Mary do something bad to John
        need John hate Mary ⇐ ?
             suggest John do something bad to Mary
```

The suggestions are in the form of conceptual patterns. If a sentence is interpreted as a structure that fits one of these patterns, then immediately the analyzer can decide what role this structure plays in a coherent interpretation of the text. More importantly, for explaining contextual effects on the course of analysis, the results of partial analysis can be used to reduce the set of possible patterns. And fewer predictions means that the predictions that remain have a greater effect on the flow of analysis. The end of the sentence will be seen from a very narrow viewpoint, even though that end might be highly ambiguous. This will be illustrated in the sections that follow.

An important point to make about the use of story-patterns with texts is that a text has only a limited view of the pattern. As far as prediction goes, the analysis of a text leads to the expectation only of those conceptualizations in the immediate vicinity of the equivalent story-pattern element. Further, as the text moves along a pattern, earlier lines in the pattern drop out of view. Thus, for example, if the text is "John hated Mary. He gave her a sock," I believe that the set of predictions at the end of the analysis of the second sentence, based on a story-pattern, is basically that either she is hurt or she will do something back. The previously made prediction, that there would be a specification of why John hated Mary, is lost from view as the text has moved along the story-pattern. Indeed, the sentence "She hit him" has different relationships in the two texts:

1) John hated Mary.
   She hit him. (This is why he hated her.)
2) John hated Mary
   He gave her a sock.
   She hit him. (This is her response to being hit by John.)

Rules for telling exactly how limited a view a text has of a story-pattern cannot be given here. This is partly because of lack of experience. But it is also hard to answer because it involves a question of pragmatics. With a wider view, more predictions are made. With more predictions, the greater is the chance that an immediate use can be made of a sentence in a text. But at the same time, with more predictions there is more bookkeeping and a greater chance that conflicts between predictions will arise and have to be resolved. The example of story-pattern prediction that will be given uses only the two adjacent lines in the story-pattern, a very restricted view, but no final answer to the question is being given here.

The predictions based on story-patterns are also limited to those involving links like the causal. That is, the reasons and results of an action are predicted by the static story-patterns. A more dynamic and hence more flexible approach to making predictions about reasons and results is developed in Rieger (1974). The two approaches are not contradictory but rather they emphasize different aspects of the problem. Rieger's work focusses on predictions, based on general beliefs, involving a fair amount of deductive manipulation. The predictions here are based on idiosyncratic, highly specific beliefs, such as what goes on during a hunt. What is described here is really just one facet of the general inference task discussed by Rieger.

Story-patterns do not predict sentences modifying a previous sentence. For example, "John made Mary mad. She had never been so furious," has a second sentence modifying a state change described by the first. The content of the second sentence, while consistent with the first, would not be predicted. The second sentence occupies a role in the text much like the role an adverb plays in a sentence. In neither case is it predicted, but in neither case is it inconsistent.

At this stage in the development of extended analysis, it cannot be said how many predictions it may be necessary to carry along at one time. Those predictions arising from story-patterns should not be too numerous. Story-patterns do not predict all the possible consequences and antecedents of an event. Instead they predict only those that are most likely to be mentioned. The predictions are very oriented towards the task of language comprehension.

It may be that story-pattern predictions and contextual lexicons, both of which are brought into working memory as part of context clusters, are not enough to model all the assumptions about meaning that people can make when understanding texts. However the analyzer will not fail to know what to do with a sentence if no predictions exist. The analyzer always has something to work with, based on the requests associated with the words of the sentence. Any predictions, in fact, require the presence of requests to have an effect. If some prediction is not made that should be, the analyzer still constructs some interpretation of the sentence for that context. There is at least one other source of prediction, based on stylistics. This is mentioned briefly in section 14.1

It should be noted that just because a certain sense of a word is the best choice in some context, it needn't be the case that this sense is predicted from the context. It may be that initially one sense of the word is chosen but the final interpretation of the sentence that results fits badly with the conceptualizations of the previous text. Therefore reanalyses are tried until a satisfactory interpretation is found. These reanalyses may choose a different sense of the word in question, and it is indeed true that context has determined the final sense of that word.

This kind of effect is not the kind we're trying to explain here. Rather we are concerned with those cases where the choice of a word sense seems clearly determined the moment it is encountered. The predictions from story-patterns seem to provide this kind of determination.

13.3 PATTERNS TO PREDICTIONS

The basic method of generating predictions from patterns is this: suppose at the start of the analysis of a sentence three patterns have been predicted as possible results of that analysis. Pattern A is X_1 Y_1 Z_1, pattern B is X_1 Y_1 Z_2 and pattern C is X_1 Y_2 Z_3. Assume these are the only patterns predicted. Then there is a prediction that the analysis will produce a structure beginning with X_1. If X_1 is found, then a prediction can be made that either Y_1 or Y_2 will follow. If Y_1 is found then there is a prediction that Z_1 or Z_2 will follow. If Y_2 is found then there is a prediction that Z_3 will follow.

The phrase "will follow" is talking about the final result of the analysis, not about the order in which the items are discovered. For example, we would also predict from --Z_3 that X_1 Y_2 will be found.

To take a concrete example, suppose the story-pattern for the text "John hated Mary" generated two conceptual predictions that said:

John will do something bad to Mary.

Mary did something bad to John.

It the next sentence is analyzed to the point of "John$\Leftrightarrow$do", then a prediction should be made that Mary is the HAPP_TO of the "do" and that this is bad for her.

This conversion of pattern to prediction is only the beginning, of course. The prediction created is about conceptualizations. The analyzer needs predictions about language events, based on these conceptual predictions. In the previous description of the analyzer this conversion was crudely done. If the analysis produced an unexpected conceptualization, then the analyzer was told to try again. The next sections are concerned with a more refined approach to the conversion.

13.4 CONCEPTUAL REQUESTS TO LANGUAGE REQUESTS

Consider the text example, "John hated Mary. He gave her a sock." The analysis of the first sentence, we have claimed, results in a number of conceptual predictions. One of them is that John may do something to hurt Mary. Such a prediction is independent of a particular language. As such, it says nothing about what might be seen or heard next in a particular language. Conversion is necessary.

As mentioned before, one means of conversion is to wait until the language processing is done, that is, wait until a conceptual structure is built. When such a structure is built, it can be compared with the conceptual predictions fairly simply and be accepted or rejected on the basis of this comparison.

There are a number of flaws with this approach. A major one is that it depends on reanalysis to eventually produce the desired result. But it is very difficult to direct reanalysis towards a better interpretation. Just eliminating one path is not very helpful. Many wrong paths may be taken before a good one is found, or before it is decided that the sentence does not follow the predictions and the original interpretation should be used. And if backup routines are able to affect the flow of the reanalysis, then it should have been possible for the analyzer to have caused such effects when the analysis was first done.

The alternative approach to conversion is to affect the steps in the analysis directly. This means changing the set of requests active by recognizing what a request does. But requests are difficult to recognize for two reasons. One is that requests till now have been grouped together under the situations that trigger them. One situation, as looked for by the test portion of a request triggers a number of different actions. Some of the actions perform conceptual duties and some syntactic, and there is no guarantee that the actions a request performs share a common purpose.

The second reason for difficulty in recognizing requests comes from the simple fact that the information about what an action does is not explicitly stated but only implicit in the functions and arguments used. An action relevant to CONCEPT might appear either as a call on REPLACE or INPLACE, with the first argument being CONCEPT. A particular position in CONCEPT might be filled either by CHOOSE, IMBED, or INPLACE, and the request doing such might be hidden in the setting of a sense of a preposition.

The idea of a need is relevant to both these difficulties. It is directly related to the second because a need is an explicit statement of the purpose of a request. No matter what functions are used by the request, if the request appears under the need CONCEPT, then it is placing a value in CONCEPT. And because requests are organized under needs and not situations, the first difficulty is also taken care of. If requests are grouped by needs, then there is no case of one request performing several unrelated actions.

In Chapter 8, on multi-sentence analysis, there were two texts involving conceptual predictions. One was "John hated Mary. John gave Mary a sock," and the other was "John was hunting. John shot a buck." The first will be redone in detail later, to illustrate text analysis using needs and story-patterns, but a brief word can be said now about the way both can be analyzed.

In the text "John hated Mary. John gave Mary a sock," the need for a value for CONCEPT is utilized. Analyzing the first sentence sets up a prediction, among others, that the topic of the second sentence is John doing something bad to Mary. Initially, then, there is a preference that CONCEPT be an action by John detrimental to Mary. When the word "give" is encountered in the second sentence, there are two requests for GIVE1 grouped under the need CONCEPT. These were given in section 12.8. One of them offers an ATRANS framework for CONCEPT and the other says the framework is specified by the OBJ of the verb. At this point the analyzer can prefer the second request over the first because it assumes that ATRANSing someone an object is not normally detrimental to them. That is, the first request does not match the prediction and is downgraded. If the analyzer prefers the second request, then it prefers the situation that triggers the second request, which is that the OBJ be the name of an event. Eventually, as we shall see, this leads to the preference that "a sock" refers to striking someone and not to an item of footwear.

The other text, "John was hunting. John shot a buck," involves the need TO which is the conceptual goal of a motion act. The first sentence sets up the conceptual prediction that John is shooting bullets at animals. Referring to the definition (unmodified of course) of "shoot" in section 8.3, we see that there are two requests filling CONCEPT. One involves the PROPELing of bullets (shooting) and the other involves the spending of money. The analyzer doesn't know that the pattern "John shoot bullets at animals" really applies until the value of CONCEPT is set, by "shoot", to "John shoot bullets at *." At this point the pattern match with the conceptual prediction leads to the prediction that the TO should be an animal. Under the need TO is a request triggered if the OBJ is an animal. Therefore the preferred situation is that OBJ refer to an animal and this leads to the interpretation of "a buck" as a deer and not a dollar.

From these two sketches, we see that the path of conversion is from context cluster to story-pattern to conceptualization to preference. Preferring certain results, coupled with the information provided by needs that identify what situations lead to these results, is the method by which language predictions are finally generated. The idea of preference, and what it means for the analyzer, is the topic of the next four sections.

13.5 PREFERENCE: SENSE CHOICE

In the analyzer described in Part I, the sense of a word was picked from a list of possibilities associated with that word (or with a root form of that word). The monitor program did this using a function called FIND_SENSE. Little was said about the function because it had little to do. It took the one sense that appeared under the property CURRENT, if there was such a sense, or took the first sense on the list under the property COMMON, if CURRENT was empty. The real work of disambiguation of words like prepositions was done ahead of time by the requests of other words. These requests gave particular values to CURRENT, which FIND_SENSE returned when called.

In the extended analyzer, FIND_SENSE is slightly more complicated, due to the addition of preference. When no preference is present FIND_SENSE behaves as before. When, however, a preference does exist, FIND_SENSE tries to satisfy it. For example, if there is a preference that the word that FIND_SENSE is working with refer to an animal, then FIND_SENSE will not simply return the CURRENT sense, but instead will search CURRENT and COMMON for a sense that has the feature of being an animal. If there is none, then FIND_SENSE behaves as if there is no preference. If there is such a sense, it is the value that FIND_SENSE returns, rather than whatever CURRENT may be. At the same time this sense should probably be made the new value of CURRENT, but that will not concern us here.

The preferences that FIND_SENSE must apply are those that were passed down the chains of request dependencies to the role variable SENSE. Now, most of those requests that refer to SENSE expect it to be the sense of a noun phrase. For example, a request that fills RECIP with the value of SENSE, if the value of SENSE refers to a human, is expecting the sense of a noun phrase, like "a man", not just "man". A noun phrase introducer, like "a", temporarily sets aside all other requests. When a noun phrase has been built, these requests are returned and SENSE is set to the sense of the newly built noun phrase.

FIND_SENSE, however, is needed while the noun phrase is being built. This is because the end of a noun phrase is recognized by encountering some word that cannot belong to the noun phrase, such as a verb. But to know that a word does not belong, it is necessary to know the sense of that word. Therefore FIND_SENSE is called during the building of noun phrases.

Now, it is possible that FIND_SENSE will have to apply two separate preferences to the words in the noun phrase being built. This happens with verbs that have both the RECIP and the OBJ syntactic roles, such as "give" does. Each of these roles contributes information to the final conceptual structure that is the analysis result. Therefore a conceptual prediction about that result may lead to preferences about the natures of both RECIP and OBJ. But when a noun phrase is being built it is not known which role it is going to be playing. Hence it is necessary to apply to a noun phrase being built the preferences for both OBJ and RECIP.

For example, consider these three two-sentence texts:

A) John was feeding the deer at the zoo.
He gave a buck some peanuts.

B) John was feeding the deer at the zoo.
He gave a buck to the peanut vendor.

C) John saw a beggar on the street.
He gave a buck to him.

I assume that "a buck" refers to a deer in text (A) but to a dollar in texts (B) and (C). Looking at the analysis of these three texts we can see the application of preferences for both RECIP and OBJ. The (B) example also gives us a chance to mention backup, because it is clear that the right sense for "a buck" cannot be picked on the first pass in both (A) and (B).

In text (A) the conceptual prediction is that "John will give food to a deer." Therefore, once John and giving are recognized there are two preferences to apply to the noun phrase "a buck". One prefers that "a buck" refer to a deer and the other prefers that "a buck" refer to food normally given to deer. Only one of these preferences succeeds and this is, of course, the deer sense of "buck". Once the deer sense of "a buck" is chosen, it becomes the RECIP because it is animate.

In text (B) "a buck"is also taken as referring to a deer. But now the phrase "to the peanut vendor" supplies another RECIP, because of the "to". This conflicts and causes backup to occur. The actions that I think occur at this point in the analysis are beyond programming at this time. Basically, "giving to the peanut vendor" is assumed to be the ATRANSing of something to him. There are gaps to be filled for the OBJ and OBJECT of the "gave" and "ATRANS" respectively. A peanut vendor (or any kind of salesman) is a person from whom one normally buys things, by definition of "vendor". Conceptually, then, there is an ATRANS of money to the vendor. Therefore, there is, by pattern matching, a prediction that the object of the current giving be money. Further, since RECIP is filled, the analyzer knows that the other noun phrase must be the

OBJ. This other noun phrase is "a buck". The preference that OBJ refer to money can be applied to this noun phrase, and thereby the money sense of "a buck" is chosen.

Notice that this description of the analysis would not choose the money sense of "a buck" in the following text, even though backup occurs, because no preference for money would arise:

D) John was feeding the deer at the zoo.
He gave a buck to a little boy.

I think that choosing the animal sense of "a buck" in (D) is a reasonable interpretation.

The analysis of text (C) involves a conceptual prediction based on the f ct that a beggar is someone who normally asks people to give him food or money. The conceptual pattern predicted following the first sentence is "John may give money to the beggar." There are two preferences, then, for the noun phrase following "gave". "A buck" should either refer to money or to the beggar. In this case it can refer only to money and this sense is chosen. This means simultaneously that the noun phrase "a buck" is chosen as the OBJ. The prepositional phrase "to the beggar" does not conflict, and the analysis proceeds without problem.

The text examples (A) through (D) illustrate the action of preference on the choosing of senses for words. These preferences can be used effectively in converting conceptual predictions to language analysis effects.

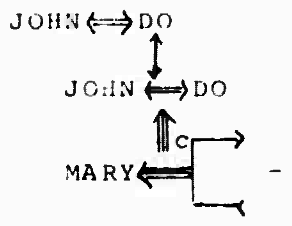
A point made was the way in which the analyzer applied preferences when it knew that a particular item must fill a certain role. For example, in text (C), there was a prediction, after backup occurred, that "John would give money to the peanut vendor." But because the analyzer knew that the conceptual TO was already filled (by "the peanut vendor"), it knew that the only preference to apply to "a buck" was the preference on OBJECT for a reference to money.

This section began by mentioning FIND_SENSE because the focus of this section was on preference of senses for individual words. In the next section we back up a bit and focus on how preferences about sense choices are obtained.

13.6 PREFERENCE: EXAMPLE ANALYSIS

Eventually the conceptual predictions that are made from the conceptual patterns need to have an effect on the choice of word senses. The text, "John hated Mary. John gave Mary a sock," will be the model for describing the manipulations the analyzer must perform when converting conceptual patterns to preferences of word senses. The description that follows is an expansion of the sketch presented in section 13.4 ("Conceptual Requests to Language Requests").

When the sentence, "John hated Mary," is analyzed, the following pattern is predicted (among others) as a possible topic of the next sentence:



That is, John will do something that could cause Mary to undergo a negative state change.

When the second sentence is analyzed, the requests for "gave", i.e. GIVE1, are added. These are detailed in section 12.8. The monitor has a predicted pattern for CONCEPT. Therefore it looks at those requests that have been suggested whose need is CONCEPT.

Using needs the monitor picks a subset of the requests waiting, knowing that each of these requests, if executed, fills the gap in CONCEPT, but not other gaps that are waiting to be filled. Therefore the monitor can see what each request does by first saving the current value of CONCEPT (presumably empty although some information, like TIME, may be already specified), evaluating the action of the request and looking at the resultant value of CONCEPT. It must also assume that, for the moment, SENSE is NIL since, as in GIVE1, some actions use SENSE s a building element. Afterwards the original value of CONCEPT is replaced. The monitor must also save temporar[illegible] the value of REQUESTS since the building of a structure may entail the addition of suggestions for completing it.

While the hypothetical value of CONCEPT is present the monitor also triggers those requests that fill needs this assumed structure generates. In this hypothetical mode, however, these requests are not deleted once they are used since, of course, they have not really been used. It should be noted that the requests triggered by an assumption like this are the special "must be used" requests. For example, the RESP_FOR created by "John gave..." is such a request. The RESP_FOR has been filled and is waiting to be used, not to be filled.

In this example the monitor evaluates the action of the first suggested request for filling CONCEPT. This creates the structure:

$$\begin{tikzcd}
 & & & & \text{(RECIP)} \\
\boxed{} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{(OBJ)} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \boxed{} \arrow[ul]
\end{tikzcd}$$

There are, since neither OBJ nor RECIP are known, gaps in this structure for everything except the act. The gap for the ACTOR is filled however, by the RESP_FOR request and this produces:

$$\begin{tikzcd}
 & & & & \text{(RECIP)} \\
\text{JOHN} \arrow[r, Leftrightarrow] & \text{ATRANS} & \text{(OBJ)} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & \text{JOHN} \arrow[ul]
\end{tikzcd}$$

At this point all the monitor can build with this assumption is done. Now it knows that it has John doing some action. It asks memory, therefore, if John can hurt Mary by giving away something, i.e. it asks about the structure:

$$\begin{tikzcd}
 & & & & (RECIP) \\
JOHN \arrow[r, Leftrightarrow] & ATRANS \arrow[d, phantom] & (OBJ) \arrow[l, "O"'] & {} \arrow[l, "R"'] \arrow[ur] & \\
 & & & & JOHN \arrow[ul] \\
MARY & {} \arrow[l, Leftarrow] \arrow[uu, Rightarrow, "C"] \arrow[r, "?"] \arrow[dr] & - & & \\
 & & {} & &
\end{tikzcd}$$

In some contexts the answer to this would be "yes". For example, if John had something that Mary wanted and, further, which she hoped to get some day, then he could hurt her by giving the object to someone else. But assuming that such a situation is not known to memory, the answer would be "no". That is, giving things is not a normal way of hurting people. Therefore the monitor demotes this request. That is, the monitor has a negative preference towards situations that would make the test predicate of this request become true.

This negative preference has an effect only if there are other requests which are not negatively preferred. In this example the monitor has yet to look at the second suggestion for CONCEPT. When it activates this one, no structure is produced. This is because the structure depends on what the value of SENSE is, and SENSE is assumed to be NIL for the moment. Because there is no structure, the monitor neither prefers nor rejects the second suggestion. Therefore this second suggestion is preferable to the negatively preferred first suggestion. This means that the situations that would make the test predicate of the second suggestion true are preferred.

The focus of the test predicate is SENSE. This means that the test predicate is passed to FIND_SENSE and when FIND_SENSE is applied to a word, it will look for a sense that makes this predicate true. If none exists, it returns the CURRENT or first COMMON sense, as usual.

In this example the test predicate is "(FEATURE SENSE (QUOTE EVENT))". When the analysis reaches "sock" in "John gave Mary a sock," this predicate is true only for the sense of "sock" that means "to hit someone". Thus the final interpretation will be that John hit Mary.

The original analyzer of Part I, with the addition of IM_REQS and such, was able to do the same example and achieve the same final result. However, the steps in the two analyses differ in the way the final choice of a meaning for "a sock" is made. In the original analyzer, a choice for "sock" was made, a total structure was built with this assumption for the OBJ of "gave", and then, if memory didn't like the result, the work was undone and a different sense of "sock" was used. In the extended approach, the point at which memory is asked about the possible consequences of an action by John occurs before "sock" is seen. Having "gave", it turns out to be possible to prefer one request over another. With this knowledge, the final sense of "sock" can be found using only the fact that an event, not a physical object, is desired.

This effect, that decisions are made as soon as enough information becomes available, was one of the goals of the extension of the analyzer. Even the separation of the RESP_FOR request, which is not important for this example, is intended to further this effect. By separating, the monitor can tell that no matter what request is used, the SUBJ, if a person, is the RESP_FOR in the final result.

One example of course cannot include all the possible problems that occur. The next section looks at the general mechamism by which predictions are passed along chains.

13.7 PREFERENCE: FOLLOWING A CHAIN

In section 12.6 the following of a chain from need to focus and back to need again was referred to. As such a chain is followed, predictions are made and passed along. The example in the previous section had some simple instances of creating new predictions by using the test predicate that the monitor wished would be triggered eventually. This is a special case of the general method of chain following.

The general case consists of having a preference P, and a request with the form:

(NEED (FOCUS TEST ACTION))

where ACTION involves giving a value to the NEED role. So we could write the request as:

(NEED (FOCUS TEST (NEED$\leftarrow$FORM)))

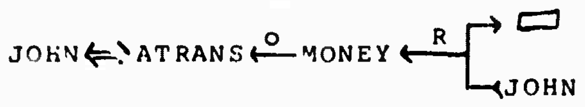
Often NEED is something like OBJ. Then usually both FOCUS and FORM are equal to SENSE. When NEED is CONCEPT, FORM may contain the FOCUS as a part. E.b. FOCUS may be SENSE and FORM will be equal to:

$$\begin{tikzcd} ONE \arrow[rr, Leftrightarrow] & & DO \\ & SENSE \arrow[u, Rightarrow] & \end{tikzcd}$$

When the monitor applies P to some request, P is actually applied to the FORM. If FORM satisfies P then TEST is the new preference. It is applied to the requests filling FOCUS. If FORM contains variables, then to make it satisfy the preference P, some conditions, i.e. preferences, may be placed on these variables as well.

If FORM is just one variable, then P is passed directly to it. If FORM is more complicated, it then becomes a matter of finding those conditions which would make FORM satisfy P. There will be conditions about the variables in FORM.

For example, suppose FORM were:



and the preference was that John does something good for Mary. Now a condition in FORM that would make this true is that the recipient of the ATRANS be Mary. Therefore this is a preference that would be applied to the requests trying to satisfy the need for TO.

If the preference on TO is not blocked at some point, i.e. the chain does not end with a set of requests all of which are negatively preferred because all contradict the preference, then the monitor can accept the original request. Then it can use the TEST to generate a preference on FOCUS, as well.

Now if FOCUS and FORM are the same, there will be two preferences waiting at the same time. For example, with the request:

```
        (OBJ (SENSE (FEATURE SENSE (QUOTE POBJ))
                     (CHOOSE OBJ SENSE)))
```

and the preference "(FEATURE OBJ 'QUOTE MONEY))", monitor would first follow the FORM and pass the preference on to SENSE, then it would follow FOCUS and put the test predicate on SENSE as well. The final preference on SENSE would thus be:

```
        AND (FEATURE SENSE (QUOTE POBJ))
            (FEATURE SENSE (QUOTE MONEY)))
```

Effectively this is equivalent to the simple preference for money, assuming that money objects are always physical. This is why informally one can talk about simply passing a very specific preference through an action like CHOOSE.

It can now be seen that the description of the analysis of "John hated Mary. John gave Mary a sock" omitted the passing of preferences on FORM. For if the preference that "John hurt Mary" is applied to the FORM in the second suggestion for CONCEPT, then there is a preference that SENSE not only be an event, but that if John is RESP_FOR this event, then Mary would be hurt by it. This extra preference does not affect the example analyzed. What it might affect is a case where the object of "gave" could either be a physical object or some event that benefits the RECIP of "gave". In the simple description of text analysis, the non-physical sense would still be preferred because there is no mention that the sense should be negative. In the description just given, the sense choice would depend on whether thwarting a prediction at this point should cause the entire chain of preference to be undone, and hence perhaps prefer the ATRANS request, or the chain, once built, should be maintained and the non-physical sense used despite the fact that it contradicts one of the preferences. It is not clear at this time which reaction to thwarted preferences models human behavior best.

It should also be pointed out that for the monitor to pass preferences along FORM as well as FOCUS, it still does not need the capability to read Lisp programs. What it needs to know is when and where FORM involves roles like SENSE or SUBJ. The functions CHOOSE, REPLACE, and INPLACE did not, even in the original analyzer, pass just the values of the roles. They passed the pointers these roles had to their values. In the extended analyzer, they must also keep track of the role names that these pointers belong to. With this, monitor can execute a gap-filling action and then, looking at the result, can see what roles were used in what structures.

The next obvious topic would be how the conditions are created that must be true if FORM is to satisfy a preference. At this point however the discussion will stop. We have seen how the analysis system can construct a conceptualization whose truth-value affects the course of further analysis. And we have seen how purely conceptual answers, as returned by general memory processes, are used by the analysis system to make predictions about future language events.

The distinction between memory and analysis processes should not be understood as a claim that there is a real difference between them. In fact, one of the major arguments for the analysis system described here is that its devices are reasonable general cognitive processing devices. In a simple way expectations are tied together by the roles which they depend upon and affect. Likewise in a simple way conceptual information is diffused through this structure of expectations, so that new knowledge is available to all expectations in appropriate forms.

The distinction of processes that begin with or end with language elements, from those that do not, serves one purpose. It offers a reasonable point at which this description of an analysis system can stop.

We can summarize the general preference passing algorithm. It starts with a preference, that is, a predicate about the value of some role. The requests whose NEED equals that role are examined. Examination consists of executing the ACTION of the request and looking at the structure, if any, that is produced. Within this structure will appear the names of those roles whose value at the time of execution is not yet known. These are the variables of the structure.

The analyzer, using memory, asks if the structure produced satisfies the preference predicate. If the answer is "no" the preference passing fails. If the answer is "yes", there may be conditions on the values of the variables in the structure which must be satisfied. These conditions are passed as preferences to the requests whose NEEDs equal the role names/variables. If all of these preference chains succeed, then this request is preferred. This means that the TEST predicate is applied as a preference to those requests whose NEEDs equal the FOCUS of the preferred request.

A preference chain terminates successfully when the focus of a preferred request is SENSE. At this point everything that is known about the current state of the analysis has been used. Now the analyzer must wait for more input.

CHAPTER 14

REVIEW

14.1 OBSERVATIONS

At various places in Parts I and II, general statements were made about the analysis system. These were basic principles that were intended to explain why things were done the way they were. Some of these principles were about the analyzer itself and some were principles behind the principles. All of these statements were reasons for what was being done, although these reasons were of different levels of abstraction.

At this point we can make a few more statements about the analysis system. These are more in the nature of observations, looking back at what the assumed principles led to.

The Need as Organizer

The development of the mechanism of the need had a number of ramifications on the analysis system. It was originally devised as the most reasonable way to regulate the coming and going of requests. It was in particular an answer to the first difficulty that the analysis of text raised, the difficulty of getting rid of some but not all requests. The need not only showed how to get rid of requests that were never executed but gave an explanation for why requests that were executed disappeared afterwards. That a request should only be executed once had seemed both necessary and reasonable, but now it could be seen as part of the general phenomenon of requests being removed when their needs were filled.

It was then found that the need was important for the process of converting conceptual predictions to language predictions. Originally there was the general idea of a routine that would start with a conceptual prediction, look at each request, and prefer those situations that activated those requests that were most compatible with the conceptual prediction. The crucial step seemed to be how to recognize what a request was doing. The need turned out to specify exactly what was wanted, when the conversion routine was translated from a general idea into a specific algorithm.

The concept of need also provided a motivation for the stacking, or setting aside, of requests. This motivation was based on the bond formed between the set of requests and the set of gaps in the current structures being built. When embedding occurred, the change was a shift in structures. This change in structure meant a change in the set of gaps to be filled. This change in the set of gaps meant a change in the set of needs, and change in the set of needs meant a change in the set of requests. Unstacking, or resetting, the requests was then a matter of returning to a previous structure, which meant returning to the needs of that structure.

The question of how deeply stacking can occur becomes one of how many times shifts in structure can happen before the analyzer is unable to remember where to return to. And a new question is raised about how recoverable the suggested requests are. That is, when a previous structure is returned, inspection of it can determine the set of needs. But it is not clear that the set of suggestions for these needs is still recoverable. It can be, in the analyzer at present, but consider the following two sentences:

John told Mary Bill was sick.

John told Mary, who was upset, Bill was sick.

There is a suggestion with "tell" that says that, to fill the need for the object of the communication, look for an unintroduced clause following the specification of the recipient. Thus in the first sentence, "Bill was sick" is taken as being the content of what John told Mary. In the second sentence an embedding occurs following Mary. To me the second sentence requires a "that" before "Bill" to be natural. That is, the clause should not be unintroduced in the second sentence. It seems as though the suggestion described has been lost due to the embedding. "That", which had its meaning altered, is required to bring back the necessary request.

To return to the role that the need came to play, I would also refer the reader to section 12.4 in Part II, "Internal Versus External Needs". There, needs turned out to provide a motivation for the way noun phrases were being handled.

The need, in other words, is really the star of Part II, both as a device for the processes of extended analysis and as a concept about which much of the analyzer had implicitly been organized.

Top-down and Bottom-up

To some extent, it could be claimed that the approach to analysis in Part I is bottom-up, while that in Part II is top-down. That is, in Part I the analysis is based on the discovery of an element which leads to a set of hypotheses about what function this element might be playing. The element in this case is a word, and the hypothesized functions are the requests of that word. The interaction of the hypotheses from the various elements results in the final interpretation.

In Part II the analysis is based on making a prediction about the function of future elements and, when an element is discovered, seeing if it can fulfill this function. In this case the functions are preferred word senses and the elements, again, are words. The final result is the product of the successful predictions.

These two approaches do not contradict each other. In fact each depends upon the other. The Part II approach requires the Part I because, in Part II, the top-down predictions are not complete, as they are in a top-down parsing strategy for a grammar. That is, there are situations not predicted by the extended analysis routines. When such situations occur, the analysis system depends on the hypothesizing of requests from Part I. At the same time, the analyzer in Part I depends on the operations of Part II because the analyzer is not a pure bottom-up approach. That is, the analyzer in Part I does not keep all possible hypotheses until they are definitely ruled out. Instead it tries to choose one as soon as possible as the most likely hypothesis. To do this in a text it requires the information passed by the mechanisms of Part II.

The two approaches, then, form one integrated system. The two approaches do not take turns but rather operate simultaneously, acting on different levels. The final result of an analysis is therefore the product of successful predictions that have modified and been modified by the sets of hypotheses, and their interactions, generated from the input elements.

The Monitor

The monitor program in Part I was basically very simple. It took an input element, chose a word sense for it, looked through a list of requests, evaluated those that said they applied, and removed from the list those requests that had been executed. This loop was repeated for each input element. Except for noun phrases, all the activity was by the instructions themselves.

In Part II the monitor became more complex. Instead of keeping a list of requests, it keeps a list of needs, each of which has a list of requests attached. These needs are related to gaps in structures that are being built. The input loop is almost the same, but instead of removing requests directly needs are removed as they are filled by the execution of requests. The monitor also has a new job. This is to keep track of conceptual predictions and, through the chaining algorithm, eventually modify the function that chooses word senses.

I would claim however that, if anything, the monitor is less language-specific in Part II, despite the increase in its complexity. This is because the handling of noun phrases is removed from the monitor. The loading of suggestions from word senses, and hence the inhibiting of that loading, is handled instead by trap routines that are, at least in spirit, separate from the control functions of the monitor. This separation decreases the language-specificness of the monitor.

The major additions, on the other hand, are the building of chains of request dependencies and the passing of predictions down these chains. Both of these processes are independent of the particular language the analyzer is concerned with. These mechanisms for handling contextual effects are based on the most fundamental aspects of requests. The requests for handling any language will consist of TESTs and ACTIONs. The TEST predicate will depend upon information coming either from the input or from some role in a structure. Therefore a focus for a request can always be specified. The ACTION function will be placing the structure it is building in some other structure. Therefore a need for a request can always be specified. These elements are all that are needed to implement the algorithms for building chains and for passing predictions.

The monitor therefore is as universal as the request structure. The processes that form the monitor therefore are independent of particular languages. It is only the objects of these processes, the actual predicates, functions and structures needed, that are peculiar to a given tongue.

14.2 FUTURE WORK

In this section, some of the many open problems related to this work are mentioned. Many of them have been referred to briefly in the descriptions in Parts I and II. A few of them are specific perhaps to the structure of the approach used, but most are, I think, the kinds of problems any analysis system will have to answer.

Assigning Suggestions

One of the quite specific problems that was mentioned is that of resolving the situation where one suggestion is applicable to two needs. This arose because suggestions were allowed to be of the form "if you need something of type X, then in situation Z do Y." It might happen that several different needs would be present, all of type X, when the suggestion is made. One question is whether this situation occurs, or whether it should always be assumed that when it does, either the suggestion or the needs were incorrectly specified in the dictionary. If it can occur, is the solution to assign the suggestion to both needs, modified in each case so that the filling of X goes to the right need, or only to one? If the latter approach is chosen, what are the criteria for deciding? Alternatively, should the suggestion be held until it is triggered, and then a decision made, based on the nature of the structure the suggestion produces, as to which need should be filled?

Failure of Preferences

Another problem involves the chains of preferences. Given an initial conceptual prediction the chains of request dependencies are followed until eventually there is a preference at the language level. The problem arises when this language preference is contradicted. That is, suppose a certain feature is preferred but none of the word senses available satisfy this preference. Should the preference chain be forgotten and a "most common" sense of the word picked? Or should the preference chain be re-examined and modified in some way until a preference is generated that can be satisfied?

<u>Preferences and Noun Phrases</u>

Another problem stems from the fact that noun phrase boundaries may be recognized only by looking at the next word not in the phrase. The problem is that when the analyzer is building a noun phrase it is applying to the function choosing the word senses a set of preferences. Once the noun phrase is finished, various requests will be triggered, and the set of preferences is likely to change. If, however, the analyzer looked at the first word after the noun phrase to determine that the noun phrase had ended, then it applied the same preference set to that word that it did to the noun phrase elements. When all the actions caused by the "digesting" of the noun phrase have finished and this word is looked at for its own sake, should the sense previously picked be kept, or should a sense be rechosen based on whatever the new preference set is?

<u>Inferences and Chains</u>

The rest of the problems to be discussed are more concerned with the language-memory interface. They are basically the same problems that people working on models of human deductive systems have encountered, but here they are seen from the viewpoint of language comprehension.

For example, the problem of depth of inference is important when building preference chains. In the text example "John hated Mary. John gave Mary a sock," at one point the analyzer asked the memory if ATRANSing something could hurt someone other than the ATRANSer. I said that in normal circumstances the answer to this should be "no", but that there are situations where it could be "yes". With a simple "no" the analyzer can build the preference chain described.

Suppose though that the memory instead discovers a possible situation where the answer is "Yes". For example, the memory might respond "Yes, with the condition that Mary wants the object being ATRANSed, she expected to get it, and she is not the recipient of the ATRANS." Even though this particular hypothesis would be removed when the second sentence is analyzed to the point "John gave Mary...", it still seems unlikely that this hypothesis should have been kept at all. And, of course, the memory might have been even more imaginative and come up with situations where Mary is the recipient and is still hurt by an ATRANS to her.

How then can we restrict how imaginative the memory should be, when answering questions from the analyzer?

<u>Story-Pattern Predictions</u>

There is a large body of work to be done on determining the conceptual structure of story-patterns. Part of this is specifying what conceptual relationships should be used to build story-patterns in context clusters. Besides the causal links it seems reasonable to assume that some patterns depend on a proximity in time relationship between events. Before trying to categorize such relationships though, there needs to be a respectable collection of story-patterns that have proved useful for binding texts together. And one decision involved in the gathering of this collection is choosing between general patterns that cover many cases, but require more deduction when used to build preference chains, and specific patterns, which make more exact predictions but which cover only limited situations.

Stylistic Predictions

There is another kind of context prediction which is based only partially on the particular content of a text. These are stylistic predictions, predictions based on knowledge about how stories are written. For instance there is a prediction, I believe, arising in the following text that stems from an idea that stories should be told "symmetrically":

> It was a beautiful day. John looked out his window. To the left he saw the trees in bright colors.

A prediction that seems reasonable is that there will soon be a sentence beginning "To the right...".

Work on this type of prediction is much harder, I feel, than on the other form of context prediction. An approach could be based on very syntactic-like rules about composition, but, as with the syntax of sentences, problems will crop up due to the difficulty of letting the conceptual content of the text interact with these rules. An approach more compatible with the analysis system presented would require an understanding of what thoughts and motiviations are communicated by various types of story-telling devices. Then predictions about what kind of story is being told could be converted from these thoughts and motivations back into text constructions.

Backup

Intelligent backup routines are a very important and immediate task for future work. Of particular immediacy are the backup routines applicable primarily within one sentence.

In section 13.5, "Preference: Sense Choice", I sketched a possible flow of backup and reanalysis for the text "John was feeding the deer at the zoo. He gave a buck to the peanut vendor." It was crucial to the process that the backup routine made use of all the information available to the analyzer, including the nature of the problem that initiated the backup. With this information the backup predicted what the reanalysis should result in. In other words, the information collected for doing the reanalysis became the context of the new analysis and made a "better" prediction than the original context had made.

The making of a better prediction, while non-trivial, at least involves the same kinds of processes that are needed for forward analysis. But to do backup requires deciding where to go to begin the reanalysis. An answer that might work on single sentences, namely start over at the beginning, will be clearly inadequate for texts.

A good source of texts which require backup are jokes. The comprehension (but not the appreciation) of jokes was an early goal of this work. Jokes stress several aspects of text comprehension which have been the focal points of the discussion of extended analysis. One point is the need for making predictions, for "setting yourself up". The following joke uses the story of a hunting expedition to cause an incorrect prediction to be made:

> I was on a hunting expedition in Africa. What a time! I shot two bucks. It was all I had.

Another point is the integrity of a text, that texts can be as tightly bound together as sentences.

A third point stressed by jokes is that the backup routine predicts the result of reanalysis. In this joke it uses "all I had" to predict the idea of losing money, which then leads to a correct analysis of "I shot two bucks."

A fourth point is that the backup routines seem to know where to try again. In the above case it is the previous sentence, not the initial one, but the opposite is true in:

> My grandfather was an old Indian fighter. He did it for twenty years. Then he had to quit. There weren't any more old Indians.

This is probably due to the presence of structures in texts and sentences which set up kinds of dependency relationships. These dependencies would form the paths along which backup routines travel, somewhat independent of the actual order of text presentation.

At any rate, while backup routines for sentences seem feasible, routines capable of handling texts seem very far away as yet.

Finding Referents

The reason referential elements like pronouns are ignored in the analyzer is that, although they are a very big problem, they can be bypassed and other work can still be done. Further, the nature of the problem's solution seems to lie more in deductive rather then in comprehension processes. To do "The city councilmen refused the women a parade permit because they advocated violence" and also "The city councilmen refused the women a parade permit because they feared violence" requires very little from the language analysis but an awful lot from the deductive memory.

A partial solution, such as that described by Wilks (1973b) and sketched in Chapter 3 ("Previous Work"), could be incorporated in this system. The basic idea is to generate a conceptual pattern (from CSIRS for Wilks, from a general deductive memory here). If this matches an incomplete pattern the analyzer has produced for a sentence with referential elements, and the restrictions set up by the referential elements (e.g. female) are met, then the generated pattern provides the missing information.

The real problem with this is finding the pattern that has the answer. It is unfortunate that the common text binding predictions from story-patterns do not produce, in general, the kinds of patterns needed. This is because the conceptual predictions from story-patterns are very broad, specifying classes of possibilities. These predictions or preferences can be applied through chaining to words which have finite sets of senses, each of which produces definite conceptual structures. The prediction selects one of these senses.

Referential elements, on the other hand, are themselves broad classes of possibilities. What is needed are predictions of specific conceptual structures from which a choice can be made. The work of Charniak (1972) is the most relevant in this area. This is so not only because it considers the many kinds of information needed, but also because the basic demon mechanism he uses has much in common with the request. Hence many of his problems and solutions can be phrased in the terminology of the analyzer.

<u>Other Languages</u>

The analysis system described has been offered as a general model of language comprehension. Although specific functions and syntactic relationships are peculiar to English, the basic structures of requests and needs, of conceptualizations and context clusters, etc., are oriented towards language processing in general. There is no way to prove this claim rigorously, but there are two ways to support it.

One way is to show how the processes can be viewed as memory mechanisms. This has been the thrust of the general comments that have been made about this system. The other way is to construct similar analysis systems for other languages.

Work has just begun at the Istituto per gli studi semantici e cognitivi, in Switzerland, on a German analyzer. While German bears many similarities to English, it has one feature that makes the design of an analyzer for it of great interest: the verb comes last. That is, the word that often specifies the main conceptual frame for the sentence is frequently the last word read. Since the English analyzer depended on the verb a great deal (the verb is not given the special status of being central, but most of them turn out to be so), this aspect of German would seem to be a problem.

However, it should be pointed out that in the analysis of English in Part II, the verb was no longer the only source of predictions about the content of the sentence. There were also conceptual predictions originating from the analysis of previous text. In German, I claim, conceptual predictions play a much stronger role, especially in isolated sentences. For example, the analysis of the "Germanized" English sentence "John had with a club Bill on the head hit" would involve a prediction that John was hitting Bill before "hit" was actually read. This prediction, in isolation, would be based on what someone is likely to do with a club to someone else. In the context of a description of a fight this prediction would be even stronger. Even in English one can construct sentences where predictions about conceptual relationships are based on the objects involved, such as "With a freshly sharpened knife and a long fork, John carved the turkey." The difference on this point between German and English is that German makes greater use of these conceptual predictions.

For this reason, the work on the German analyzer offers an interesting challenge. It requires, at an early stage, the mechanisms described in Part II, and uses them constantly. At the same time, it advances our knowledge of English analysis by focussing on effects that occur in English but are somewhat masked by the way English sentences are constructed.

14.3 SUMMARY

The various major elements of the analysis system have been introduced and summarized several times. Therefore this shall be only a brief recapitulation of the total effort.

What has been described is a system for the comprehension of natural language, in particular of sentences in English. The system has one aspect of completeness in that it involves manipulations both on the conceptual and on the surface levels, and a non-ad hoc means of communication between the two levels. This is a sort of completeness of depth. The system is not complete in terms of breadth, for it does not include all, or even most, of the manipulations needed at either level.

The comprehension processes are written in terms of requests. Requests are pairs of predicates and functions. If the predicate of a request becomes true, then the function of the request is executed. The basic flow of analysis consists of maintaining a set of requests, reading words from the input in a left-to-right direction, executing the functions of those requests whose predicates have become true, and modifying the set of remaining requests. Requests initially come from the dictionary entries of the words that appear in the sentence. The execution of a request may also introduce new requests.

The final goal of the analyzer is to build a conceptual structure that represents its interpretation of the sentence being analyzed. Along the way the analysis will produce a number of partial structures, both conceptual and syntactic. These structures lack information in various places. These gaps generate needs for structures to fill them. Requests are grouped according to the needs they would satisfy if they were executed. When a request fills a need by being executed, that need disappears and with it all the requests grouped under that need.

One structure that generates needs is the story-pattern structure of a text. A text is interpreted as a sequence of conceptualizations. A story-pattern is a proto-typical sequence of conceptualization-types that has been stored in memory. When a text is recognized as following some particular story-pattern, needs are generated to fill those parts of the pattern which have not yet appeared in the text. Associated with these needs are general conceptual patterns which the story-pattern says should appear in the still unfilled positions. It is predicted that these patterns will be found later in the text.

When these predictions are applied to the sets of requests maintained during analysis, they lead to preferences about word meanings, i.e. about what dictionary entries for words of the input should be used. The preferences, when successful, cause sentences of the text to be interpreted so as to fill out the missing elements of the story-pattern.

The analysis system adheres to the following assumptions:

1) Its primary task is to comprehend a sentence, not to assign a syntactic structure to it;
2) Pieces of the interpretation are assumed as soon as possible, while the sentence is still being read;
3) There are predictions at various times during the analysis about what will come and how input elements should be looked at first;

4) The words of a sentence, through the dictionary, provide the information base upon which all the processes depend;

The requests which originate in the dictionary are language-specific. The conceptual predictions which arise from the story structures are culture-specific. The processes that communicate between these two are universal in the sense that they do not depend on particular predicates and functions but only on the form of requests in general.

14.4 CONCLUSIONS

The development of this analysis system is still continuing. But, from what has been accomplished already, some conclusions can be drawn now.

One conclusion is that goals are a very important factor in determining the nature of the system produced. The modelling of human language comprehension was the primary reason for the development of this system. I claimed that this goal set this work apart form previous efforts. Hopefully this point is clearer now that the analysis system has been described, both in terms of where it is and of where it will be going.

Another conclusion is that the expectation has been verified as a useful mechanism for describing analysis processes. It was shown to be feasible for programming in Part I, and it was shown to be easy to extend in Part II. In Part I, the advantages derived from the fact that expectations didn't require a separation of the analysis process into a sequence of stages. Hence it was fairly easy to take an intuitive hypothesis about the flow of decisions that occurred in the comprehension of some sentence, and program that flow as a sequence of triggered expectations. In Part II, the advantages derived from the fact that expectations were small units and could be characterized and manipulated easily. Hence it was possible to tie expectations together with predictions of conceptualizations, converting these predictions into a direct effect on the flow of analysis.

Another conclusion is that generative grammars are not a prerequisite for computational linguistic progress.

Linguistic theories about such grammars are concerned with the notion of the "structure of language". By contrast I take the concern of computational linguistics to be a search for mechanisms for obtaining information from language constructions. One way to do this is to incorporate elements of generative grammars. Woods and Winograd both did this. But one can also attack a computational linguistic problem directly, creating new devices that seem most appropriate for the job at hand. The analysis system described here is a result of a direct approach to the problem of modelling human comprehension. It does so without recourse to the notion of a generative grammar.

A final conclusion of this work is the feasibility of treating language analysis as a memory process. The devices used, i.e. the expectation, the need, the context cluster and so on, and the problems of concern, i.e. adding, deleting and diffusing information, are proper to the creation of a general model of memory processes. There is a tendency, I think, to associate work on memory models with the unproductive construction of formalisms, where simple mechanisms are postulated but no content for testing these mechanisms is provided. Here, however, both content and mechanisms have developed together. An analysis system, like this one, that is consistent with a general memory model, can contribute not only to the domain of computational linguistics, but to artificial intelligence as a whole.

REFERENCES

Charniak, E. "Towards a Model of Children's Story Comprehension", AI TR-226, MIT, December 1972.

Enea, H. and Colby K. "Idiolectic Language-Analysis for Understanding Doctor-Patient Dialogues", IJCAI3.

Goldman, N. "Computer Generation of Natural Language from a Deep Conceptual Base", Ph.D. thesis, SU, 1974.

Goldman, N. and Riesbeck, C. "A Conceptually Based Sentence Paraphraser", AIM-196, SU, May 1973.

Halliday, M. "Functional Diversity in Language as Seen from a Consideration of Modality and Mood in English", Foundations of Language, 6, 1970.

Quam, L. and Diffie, W. "Stanford Lisp 1.6 Manual", Stanford AI Lab Operating Note 28.7, SU.

Quillian, R. "Semantic Memory" in Minsky, M. Ed. Semantic Information Processing, The MIT Press, Cambridge, 1968.

Rieger, C. "Conceptual Memory", Ph.D. thesis, SU, 1974.

Riesbeck, C. "Expectation as a Basic Mechanism of Language Comprehension", presented at the International Conference on Computational Linguistics, Pisa, Italy, September 1973.

Russell, S. "Semantic Categories of Nominals for Conceptual Dependency Analysis of Natural Language", AIM-172, SU, July 1972.

Schank, R. "Conceptual Dependency: A Theory of Natural Language Understanding", Cognitive Psychology, 3, 4, October 1972.

Schank, R., Goldman, N., Rieger, C., and Riesbeck, C. "Primitive Concepts Underlying Verbs of Thought", AIM-162, SU, February 1972.

Schank, R. and Tesler, L. "A Conceptual Dependency Parser for Natural Language", Statistical Methods in Linguistics 6, 1970.

Smith, D. "Mlisp", AIM-135, SU, October 1970.

Tesler, L., Enea, H., and Smith, D. "The Lisp 70 Pattern Matching System", IJCAI3.

Wilks, Y. "An Artificial Intelligence Approach to Machine Translation", in Schank, R. and Colby, K. Eds. Computer Models of Thought and Language, W. H. Freeman and Co., San Francisco, 1973.

Wilks, Y. "Natural Language Inference", AIM-211, SU, August 1973.

Winograd, T. "Procedures as Representation for Data in a Computer Program for Understanding Natural Language", MAC TR-84, MIT, February 1971.

Woods, W. "Transition Network Grammars for Natural Language Analysis", Communications of the ACM, 13, 10, October 1970.

Abbreviations:

IJCAI3-proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, California, 1973.

MIT -Massachusetts Institute of Technology, Cambridge, Massachusetts.

SU -Computer Science Department, Stanford University, Stanford, California.