AD-A062 638 YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE F/G 5/7
INTEGRATED PARTIAL PARSING.(U)
NOV 78 R C SCHANK, M LEBOWITZ, L A BIRNBAUM N00014-75-C-1111
UNCLASSIFIED RR-143 NL

| OF |
ADA
062638

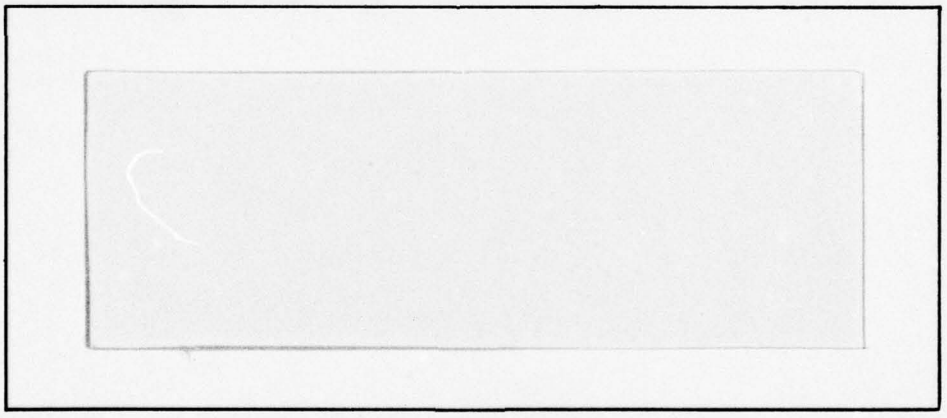
END
DATE
FILMED
3-79
DDC

AD A062638

DDC FILE COPY

LEVEL II

DDC
DEC 29 1978



This document has been approved for public release and sale; its distribution is unlimited.

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

78 12 27 018

The research described here was done at the Yale Artificial Intelligence Project and is funded in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

Integrated Partial Parsing

by

Roger C. Schank, Michael Lebowitz,
Lawrence A. Birnbaum

Research Report #143

This document has been approved for public release and sale; its distribution is unlimited.

December 1978

78 12 27 018

A

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(14) -RR-

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 143	2. GOVT ACCESSION NO. (9) Research rept.
3. RECIPIENT'S CATALOG NUMBER	
(6) 4. TITLE (and Subtitle) Integrated Partial Parsing	5. TYPE OF REPORT & PERIOD COVERED Technical
	6. PERFORMING ORG. REPORT NUMBER
(10) 7. AUTHOR(s) Roger C. Schank, Michael Lebowitz, Lawrence A. Birnbaum	8. CONTRACT OR GRANT NUMBER(s) (15) N00014-75-C-1111
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University - Dept. of Computer Science, 10 Hillhouse Avenue, New Haven, Connecticut 06520	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209	(11) 12. REPORT DATE Nov[illegible]78
	13. NUMBER OF PAGES 61
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research, Information Systems Program, Arlington, Virginia 22217 (12) 67p.	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

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

18. SUPPLEMENTARY NOTES

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

conceptual dependency
inference
interestingness
internal representation
natural language understanding
parsing
scripts
story understanding

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

A new type of natural language parser is presented. The idea behind this parser is to map input sentences into the deepest form of the representation of their meaning and inferences, as is appropriate. The parser is not distinct from an entire understanding system.

A B

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407051

A B

-- OFFICIAL DISTIRUBTION LIST --

Defense Documentation Center — 12 copies
Cameron Station
Alexandria, Virginia 22314

Office of Naval Research — 2 copies
Information Systems Program
Code 437
Arlington, Virginia 22217

Office of Naval Research — 6 copies
Code 102IP
Arlington, Virginia 22217

Advanced Research Projects Agency — 3 copies
Cybernetics Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209

Office of Naval Research — 1 copy
Branch Office - Boston
495 Summer Street
Boston, Massachusetts 02210

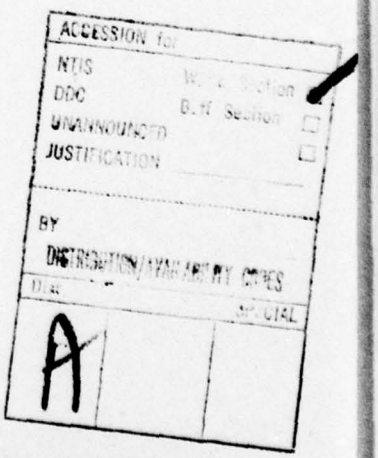
Office of Naval Research — 1 copy
Branch Office - Chicago
536 South Clark Street
Chicago, Illinois 60615

Office of Naval Research — 1 copy
Branch Office - Pasadena
1030 East Green Street
Pasadena, California 91106

Mr. Steven Wong — 1 copy
Administrative Contracting Officer
New York Area Office
715 Broadway - 5th Floor
New York, New York 10003

Naval Research Laboratory — 6 copies
Technical Information Division
Code 2627
Washington, D.C. 20375

Dr. A.L. Slafkosky — 1 copy
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380



C

Office of Naval Research — 1 copy
Code 455
Arlington, Virginia 22217

Office of Naval Research — 1 copy
Code 458
Arlington, Virginia 22217

Naval Electronics Laboratory Center — 1 copy
Advanced Software Technology Division
Code 5200
San Diego, California 92152

Mr. E.H. Gleissner — 1 copy
Naval Ship Research and Development
Computation and Mathematics Department
Bethesda, Maryland 20084

Captain Grace M. Hopper — 1 copy
NAICOM/MIS Planning Board
Office of the Chief of Naval Operations
Washington, D.C. 20350

Mr. Kin B. Thompson — 1 copy
Technical Director
Information Systems Division
OP-91T
Office of the Chief of Naval Operations
Washington, D.C. 20350

Advanced Research Project Agency — 1 copy
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209

Professor Omar Wing — 1 copy
Columbia University in the City of New York
Department of Electrical Engineering and
Computer Science
New York, New York 10027

Office of Naval Research — 1 copy
Assistant Chief for Technology
Code 200
Arlington, Virginia 22217

Captain Richard L. Martin, USN — 1 copy
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

D

Abstract

A new type of natural language parser is presented. The idea behind this parser is to map input sentences into the deepest form of the representation of their meaning and inferences, as is appropriate. The parser is not distinct from an entire understanding system. It uses an integrated conception of inferences, scripts, plans, and other knowledge to aid in the parse. Furthermore, it does not attempt to parse everything it sees. Rather it determines what is most interesting and concentrates on that, ignoring the rest.

Integrated Partial Parsing

by

Roger C. Schank, Michael Lebowitz, and Lawrence Birnbaum

1. Overview of CD Parsing

Over the course of the last ten years, researchers in our project have designed and programmed a large number of parsers. These parsers have had one goal in common: they were to be programs that mapped English sentences into the Conceptual Dependency (CD) representation of their meaning. They have also had one methodological assumption in common: the parsing algorithm that they employed was to be as psychologically correct as possible. To the extent that it was possible, our parsers were to model the way the human parsers work. This methodological assumption brought with it an operating principle which was (with one exception to be discussed later) always followed, namely that the parsing algorithm was a left-to-right, one-pass operation without backtracking.

The first parser that we worked on (Schank and Tesler, 1969), used what we called "realization rules" to map English syntactic structures into CD. (This term was taken from Lamb

(1966) and signified that we were mapping from one linguistic level to another.) The primary problem with this parser was that it violated our methodological goal of modeling human processes. For many English sentences that were ambiguous, the algorithm we used exhibited no clear preference for one interpretation over another, even though people clearly had such preferences.

In Schank et al. (1970) we proposed a solution to remedy this problem in the design of a new parser which we called SPINOZA II. SPINOZA II was to use the CD representation itself to drive the parse. That is, during the parsing process, the meaning that had been understood up to any point would help in the determination of the meaning of the rest of the sentence. This idea brought with it the concomitant idea that, since meaning would be driving the parse, we really might not have to rely very much on syntax to do our parsing. (Wilks (1973) was working on a similar idea at the same time and his view helped to support our own belief in the feasibility of the idea.) We did not believe that we could avoid syntax altogether. Rather, it was our view that relying on meaning considerations first would drastically reduce the number of syntactic rules necessary in any parsing system.

During this same time period, other researchers were investigating the problem of parsing natural language input into an internal representation. Virtually all of these methods concentrated on the syntax of sentences. One very popular technique has been the Augmented Transition Network (ATN).

Parsers of this sort have been discussed in Thorne, et. al. (1967), Bobrow and Fraser (1969), Woods (1969) and Kaplan (1975). A parser strongly related to ATN's is in Winograd (1972). ATN's have tended to deal almost exclusively with syntax, perhaps occasionally checking a few simple semantic properties of words. Even more closely tied to syntax are the parsers based on Transformational Grammar, such as Petrick (1973). A more recent parser which views syntax parser as an isolatable sub-part of language understanding is in Marcus (1975). The important thing to note about all of these programs is that they viewed parsing as a process totally isolated from the rest of understanding, and syntax as a process which can be isolated from the rest of parsing. Our view has always stressed the integration of semantics and syntax in parsing.

SPINOZA II was only partially finished when it was abandoned for reasons other than academic ones. A few other attempts were made at starting it up again until Chris Riesbeck designed a parser that was similar in spirit, but different in form from SPINOZA II (see Riesbeck (1975)). His program was based on what he termed requests, a form of productions (see Newell (1973)). Requests were activated whenever expectations about some syntactic or semantic information could be made, and were executed if the expectations they embodied came true. Thus, expectations guided the parse, making Riesbeck's system, later called ELI, very top down (see Riesbeck and Schank (1976)).

ELI was used as a front end to the SAM system (Schank et al. (1975) and Cullingford (1978)), and was combined with Gershman's (1977) work on noun groups to provide a parser that could handle very complex sentences.

One of the major problems with ELI is its fragility however. Granger (1977) designed FOUL-UP, an adjunct to ELI which determines the meaning of unknown words in context, and this produced a more robust parser. But, in actual day-to-day use, students have often found it simpler to design special purpose parsers patterned after ELI that are less cumbersome and easier to use. Carbonell's (1978) POLITICS program, for example, uses a parser that is similar to ELI, but was written by Carbonell himself.

Perhaps the most important feature of Carbonell's work is that it has pointed out to the rest of us a major flaw in our reasoning behind the design of large understanding systems. We have always leaned in the direction of modularity in the design of our programs, both because this has always been considered good programming style, and because, since our systems are very large, each separate module has often been the work of a different person.

But, this modularity has caused a number of problems. Any understanding system that we build, for example, should ideally use ELI as a front end. But ELI is, as we have said, a very large and cumbersome program to work with. Furthermore, there is another practical problem, namely that the vocabulary for any new

domain to be handled by some system we set up is unlikely to be already present in ELI. Since in ELI the definitions of words are in a sense programs themselves, any new system will require the writing of a large part of its parsing program from scratch in any case. This practical problem leads to a much more interesting issue. In the same way that we realized years ago that it was important to take advantage of the power of the CD representations available to us to build a more integrated parsing system, any new parser designed for a new system should, in principle, take advantage of the higher level understanding processes that are a part of the new system. Thus, POLITICS can parse more effectively if it can use not only the partially constructed CD representation of what it has already understood, but also its place in the ideology it is using, its overall significance, and so on. That is, modularity is, in an important sense, a disadvantage. Why not capitalize on everything that is available to help parsing along. People are no more likely to use only syntax and some particular notions of meaning (but not others) to help in the parsing than they are to use only syntax. Understanding is a completely integrated process. The idea of building modular systems has hampered advances in parsing, because the full range of our knowledge should obviously be available to help disambiguate, find appropriate word senses, and just as importantly (as we shall see later in this paper) to help us know what to ignore.

2. Paying Attention To Less Than Everything

One of the major problems with SAM, our script-based story understanding program, and also with ELI as a part of SAM, was its inability to handle genuinely new texts for which it was unprepared. A new vocabulary item, domain of discourse, or previously unencountered syntactic construction could, and often would, throw things into disarray. Since one of the outputs produced by SAM was a summary of what it had read, it seemed to us that we could produce essentially the same output with a much more robust and much faster program, FRUMP (DeJong, 1977). FRUMP does not read every word of every story that is input. Rather, it has embodied within it a theory of skimming that guides it in what it is reading. FRUMP skims for what it is interested in, in the usual case precisely the items of information it wishes to include in its summary for any particular domain that it has knowledge about. FRUMP is thus a very top down system and for this reason it cannot be considered as a replacement for SAM. SAM could, in principle, handle things it was unprepared for, although in practice this did not happen very often! FRUMP cannot respond to things it is unprepared for; but then, neither is it tripped up by such things.

For our purposes here however, what is particularly interesting about FRUMP is that it is an example of a working, robust, integrated (that is, non-modular) system. FRUMP's parser is virtually indistinct from its inferencer. The reason is simple. FRUMP knows what it needs to find in a story. It has

rules for how to find these things, which can be either inference rules or parsing rules. But such rules are really just the low-level manifestations of higher level decisions that have been made on the basis of many considerations, only some of which are related to parsing.

The fact that parsing is integrated with the rest of the understanding process is of significance for two reasons. First, this design decision has produced a fast, working, and non-fragile program (the combination of all three of which is a rarity in the world of AI). Second, FRUMP works as well as it does because its interests guide what it looks for. It can ignore what it is uninterested in and concentrate on what it wants to know.

Now let's consider how a normal, literate adult reads a story. We have considered seriously the issue of whether a human reader is more SAM-like or more FRUMP-like in his normal reading mode. And, although we possess no hard evidence one way or the other, we have come to the conclusion that a human is more FRUMP-like than we have previously had reason to believe. If this is true it has important implications about what a parser ought actually to look like. We are not suggesting that FRUMP's parser is adequate. Clearly it is not as it misses quite a bit. On the other hand, some kind of combination of FRUMP parsing and ELI might make for a very powerful and robust system indeed.

3.Time of Processing

One of the major factors to be considered in discussions of the design of a parser that is human-like is the speed with which humans can read text. Considering all the inferences and bringing in of background knowledge and other problems that an understander must deal with in the course of reading or listening to a sentence, people are very fast at the job. Now by this we do not mean that they are fast relative to some other mechanism. There really is no comparative measure available since nothing else can understand as well as people. People are very fast understanders considering the fact that they finish understanding, for the most part, as soon as the sentence they are hearing is finished being uttered. This implies that the amount of time that they have available for inferencing and knowledge application cannot wait until the end, after parsing is finished. Rather, such additional processes must be going on at the same time as parsing is going on. If this is true then it certainly makes the argument we were stating above much more significant. It implies that human processes are also non-modular. That is, people must be inferring from the early parts of a sentence before they even hear the latter parts of the sentence. If this is so then it also follows that people will make use of whatever else they discover, thus allowing word sense identification etc. to be affected by higher level processes. Thus, as models of human processing, parsers that first do their job completely and then send their results off to inferencers

make no sense.

There is a further consequence to this as well. We must ask ourselves when this non-parsing type processing takes place. There are two possible answers. Either people employ parallel processes and it all goes on at the same time, or if processing is serial, space must be being made to do this work at the expense of something else. This something else is likely to be the complete parsing of every word that is seen. That is, in the serial view, not all words are equal. Some words get a lot of the processing time (those that have great syntactic, semantic and inferential importance for example) and others hardly get noticed.

Now the question of whether the serial or parallel explanation is correct is really not resolvable here. However, even with some parallel processing, it seems plausible that the total processing capability available at any one time for use in understanding must have some bounds, and that the speed of input must often overwhelm those bounds. Thus we are still left with the necessity of processing some words at the expense of others.

The serial explanation then presumes a model of parsing which is in some places incomplete. Simply stated the idea is this: It takes n seconds to read a word and it takes m seconds to process a word. Since it seems quite likely that m is much larger than n in ordinary speech and reading, and since words come in streams in ordinary speech and reading, then it is obvious that people cannot be fully processing every word they

hear. What is more likely the case is that they are deciding what to pay serious attention to and what to pay casual attention to as they go. Such decisions can be explained on the basis of many factors. One most obvious one is interest. That is, people are liable to pay attention to (that is, devote their processing time to) what interests them. We have discussed the concept of interest and its ramifications for the inference problem in Schank (1978). The main conclusion there was that inference is controlled by interest. This is likely also to be true in this revised view of the parsing process then because we are now viewing the entire understanding process as an integrated phenomenon.

Consider the following sentence:

A small twin engine airplane carrying federal marshals and a convicted murderer who was being transported to Leavenworth crashed during an emergency landing at O'Hare Airport yesterday.

It seems obvious that some parts of this sentence are more interesting than others. But more than that, it is crucial, according to the idea stated above with respect to amount of processing time available, that the processing of some words take less time than the time it takes to read or hear them. Now at first glance this may seem a bit bizarre. How can a word be processed in less time than it takes to read or hear it if reading or hearing it is a part of that processing? Yet we are in precisely this paradoxical situation if we hold to the idea that the processing of any one word in a sentence can take longer

than the time it takes to read or hear it, since it takes no longer to process an entire sentence than it does to hear it and since the individual words come in at such a rate that there is no time between them in which to process. (This is obviously the case since just finding the word boundaries in a sentence is a very complex task because the speech stream is continuous.)

Some words then, must demand more processing time than others. Since the amount of processing time available is limited by the rate of flow of the input (which is continuous for speech), then some words must not be being processed at all (or in any case they are being processed so partially that they are hardly being seen). One hypothesis then is that the understanding process may not be entirely left to right. Since the most important words often come at the end of a phrase, the preceding words may be virtually ignored until they can be 'gathered up' right to left. To do this the understanding process must be top down in order to allow the understander to know what to ignore. By right to left, we mean that while the stream of input coming in is obviously left to right, words are stored in a buffer and virtually ignored until a word that initiates processing is found. When such a word is found, the words in the buffer are gathered up and placed in their appropriate slots. Such words usually appear at the end of phrases or breath groups.

In order to process a noun phrase of the type that opens sentence this then, we must assume that a processor virtually ignores all the words until 'airplane', simply marking their existence in short term memory for retrieval after the head noun is found. Once we know that 'airplane' is the subject of the sentence, expectations can be generated that allow us to have a better idea of what to look for (and therefore of what to ignore). For example, 'carrying' can be virtually ignored because while we are only beginning to recognize what word it is, we have already heard about the marshals and the murderer and have decided to pay attention to those items.

The point here is that we are really not seeing things one word at a time, but rather since we are seeing a continuous stream we can pick out what we find interesting, go back to discover just those relationships that connect together what we are interested in and virtually ignore the rest. Do we care that the verb 'carrying' was used instead of 'containing', or that the construction used was not 'in which they were flying'? We have already predicted that the relationship between the people and the airplane was containment because that is what it ordinarily is. We need only confirm the fact that nothing contradicts this prediction and this can be done on the fly. Under this theory there is little wonder at the fact that understanders frequently cannot remember the actual words that they read. They may never have actually read them at all!

A theory of partial parsing then, says that most words are barely noticed until some reason is found to pay attention to them. A major issue in our theory then is how we know what words we must pay attention to and begin to process seriously.

(There is an aside here that is worth making at this point. We have talked over the years about how expectations drive various parts of the understanding process (Riesbeck (1975) and Cullingford (1978) for example). The contrast here is between expectation-based processes and interest-driven processes. Obviously the most powerful and important mechanisms available to an understander are both expectation and interest driven at the same time.)

Reasons for continuing to process a given word occur at all levels of the system. Some of these are:

parser expectations: if the parser expects a certain kind of word, the satisfaction of that expectation can be taken as an extremely important force in the parser. Thus, a parser might function best that expected certain syntactic or conceptual types to the extent that it ignored everything else until it found them. This is again a violation of the idea of left to right parsing since a parser might not become interested in something until it had already passed it, ignored it, and then seen an item that caused expectations to be raised that could only be satisfied by checking backwards.

syntax: main nouns in a noun phrase can cause a processor to try

to gather up its modifiers for which there is a need or interest. Certain function words cause words to be paid attention to if their interest value has been predicted. Thus, 'to' is noticed to the extent that it can focus attention on the following head noun if it has already been determined that a location is expected and desired.

interest values: how does the parser decide what it wants to pursue? Obviously we need a fully integrated system where the parser and memory talk during the parsing of a sentence. Without such integration, there would be no overriding reasons for noticing one thing and not another. It is the role of memory and world knowledge to inform the parser of what to pay attention to. Interest values are stored in memory as part of the knowledge associated with concepts. Certain concepts are absolutely interesting, others are interesting in certain circumstances. More importantly, certain things are interesting on the basis of what has preceded them. Thus, the object of a shooting might be expected to be more interesting if the shooting took place in an embassy as opposed to a generally low-interest location such as a bar. (But of course, contexts can be created where bars can be very important. This is why it is necessary to have memory available as opposed to just a dictionary.)

top level expectations: if we are reading about an event that fits into a high level knowledge structure such as a script or a plan, predictions from that script or plan can focus interest during the processing of a sentence. Thus, we can know that the

target of an assassination and the identity of the assassin are of critical importance in reading a story about an assassination and we can thus focus in on those items as top down predictions during parsing.

To see how all this is used consider the sentence form "X went to visit Y": X is evaluated for interest value by the memory because it is a main noun and because it is a person. When no information is found, the processing should be faster then when information is found. Thus, when X is 'John' or 'Sam' we proceed quickly. If X were 'Henry Kissinger' or 'your mother' we would presumably proceed more slowly because more expectations about their behavior that are of interest would be found.

'Went' is an item that urges us to continue processing since it has no meaning by itself. (That is, we could have 'went crazy', 'went fishing', 'went to Boston', and we can't do anything until we see the next words. The theory here is why speculate at all, just ignore it.) 'Visit' is a word that calls up a script ($VISIT) if the object of the visit is an equal or a family member. But remember that other scripts can be called up by the word visit that are distinct from $VISIT. If the object of visit is 'museum' or 'Congress' we would get quite different scripts or even no available script at all. (What script does 'went to visit a mortuary' bring up?) Obviously, the problem here is that 'visit' is also almost totally ignored since it too means very little. Its real meaning is to get us very interested in the object that follows next. That is, we don't really start

processing this sentence seriously until we see what Y is. Then, if Y meets certain criteria we instantiate $VISIT. If Y is a member of the opposite sex, we have an ambiguous sentence from a scriptal (and thus from a processing) point of view. In that case, either $VISIT or $ROMANCE would be applicable, and we will now have to figure out which.

4. An Example

The following is a sentence taken from a front page story in the New York Times:

An Arabic speaking gunman shot his way into the Iraqi Embassy here (Paris) yesterday morning, held hostages through most of the day before surrendering to French policeman and then was shot by Iraqi security officials as he was led away by the French officers.

We will now examine this sentence word by word and consider the kind of processing that would be desirable in an integrated understanding scheme. In doing this, our model will skip the uninteresting parts and gather things together at the interesting parts, thus attempting to be completely finished with each sub-part at the right time. That is, any model we propose must finish processing only slightly after the input is finished being received.

One important point here is that although we will discuss this sentence in a left to right word by word fashion, there is no real reason to assume that actual processing goes one word at a time. Actually, words enter in chunks, both visually (in reading) and aurally (in speech). We can thus process the same way. That is, we can assume that the next word is available for any word under consideration. Such an assumption simplifies somewhat the problem of disambiguation.

****
An Arabic speaking gunman...
****

AN is a word that can be skipped initially. This means that it is looked up in the dictionary, and what is found there are instructions to go to the next word and place AN in some kind of short term memory (STM) to be examined later.

ARABIC is listed in the dictionary as a word that is skippable when it has been preceded by a skippable word, so it is skipped and placed in STM. (The information that ARABIC is skippable has already been compiled and it is simply looked up here. The procedure for determining what can be skipped is obviously one of the interesting problems in the issue of the development of language ability. (See Schank and Selfridge (1977) for a discussion of these issues.) In general, adjectives can be skipped, though not all can be. In particular, 'Russian' could not be skipped because it can also be a noun. Also, interesting adjectives may not be skipped, i.e. 'disgusting', 'murderous', 'lecherous' etc.)

SPEAKING is also skippable as long as no potential actors have been so far encountered. A search for ACTORs in STM finds none, so this word is also skipped.

GUNMAN is marked as an ACTOR, as a NOUN, and as a HIGH INTEREST ACTOR. The fact that we have a HIGH INTEREST word causes us to create top down requests to fill in certain information, in particular we now want to know the answers to the following questions:

WHO is he? ---- causes us to gather up stacked adjectives and add them to the memory token for this GUNMAN
WHAT did he do? ---- this is answered by an item found on the token for GUNMAN, namely SHOOT. Thus, an inference that the gunman shot or will shoot somebody is made here before anything else comes in as input
WHO did he shoot? ---- causes us to be interested in the syntactic object of the verb which we assume will be SHOOT
WHY did he shoot? ---- causes us to look for a reason
WHERE did this happen? ---- causes us to look for a location
WHAT SCRIPTS might this instantiate? ---- GUNMAN can itself cause a script to be instantiated. Prime candidates are $ROBERRY, $TERRORIST, $KIDNAP. We can now look for confirmation in the rest of the sentence.

The formulation of the above questions (as requests, see Riesbeck, 1975) now guides the parsing of the rest of this sentence. Here it is important to point out that much more than just parsing is being guided at this point by these requests. This information is what we are interested in as understanders. Thus the entire process of understanding this story is being attempted at once. These requests relate to matters of parsing and inference and scripts application and goal pursuit as well.

****
shot his way into the Iraqi Embassy...
****

SHOT is encountered and immediately is found to satisfy an expectation that was derived from GUNMAN. Satisfying an expectation of this sort is the way that conceptual structures are built and we now build the first one, namely the PROPEL action with 'bullets' as object; 'gun' as the originating direction; the gunman token as actor; and an unfilled final direction for the bullets. This unfilled slot is marked as the same one that satisfies the answer to the WHO question asked before and the parser now is interested in satisfying that request by looking for the next main noun in the next noun phrase.

HIS is skipped and held in STM as before.

WAY does not satisfy the expectation to fill the empty slot. WAY is also listed as both skippable and pointing to a direction or location. A request is set up for the location and we attempt to skip until we find it.

INTO says to keep on going and it is skipped.

THE is skipped.

IRAQI is skipped.

EMBASSY is found to be a location and is set up as the location of the PROPEL event. Furthermore, EMBASSY is marked as interesting and a place of political significance. This latter

piece of information satisfies the request for instantiating the $TERRORIST script that we had predicted (among others) from GUNMAN. Since EMBASSY is interesting, its requests are activated. One of these is for a country whose EMBASSY it is. IRAQ1 is thus found in STM as filling this request and is picked up.

Setting up $TERRORIST causes us to lose interest in the representation of the sentence as such and focuses us on setting up and filling requests from that script for the representation for the entire story. Thus, we now expect answers to the following requests:

```
Were HOSTAGES taken?
What demands were made? (money?) (free political prisoners?)
Was any damage done?
What measures to counteract the terrorist were made?
        (return fire; arrest; free hostages)
```

```
**************************
here yesterday morning...
**************************
```

HERE always means to add the dateline to the location in a news story.

YESTERDAY is found to be a time word and is thus added to the time slot of the event.

MORNING is also handled in this manner.

```
****************************************
held hostages through most of the day....
****************************************
```

HELD is skipped since it matches none of the requests. It matches none of them because the information found about HELD in the part of the dictionary we look at at this point is just that

it is a verb. No verbs were needed so we skip it. What could have changed this would have been some interest value marking or other item of significance. The advantages of this system of parsing is clearly shown here. HELD is a highly ambiguous word that previously might have caused us to make a great many predictions and look for evidence of what sense was intended. With an integrated understanding system we need not do that at all. The reason for this can be seen in what happens in subsequent processing of this phrase.

HOSTAGES is immediately found to satisfy an extant request. The TAKE HOSTAGES scene of $TERRORIST is instantiated. At this point a check is made on the stacked verb to see if doing this is okay. If the stacked verb were 'shot', for example, this instantiation would not work. HELD is found to be precisely the kind of word that fits here. The important point is that HELD never really had to be examined for its meaning which is nice because words like HELD really do not have any particular meaning. Its meaning is derived from its connection to HOSTAGES and HOSTAGES is understandable only through $TERRORIST. Thus integrated understanding plus skipping parsing facilitates processing tremendously.

THROUGHOUT is found to point to either a time or place so a request is made for a time or place word. However, at this point our understanding system knows what it is interested in. In particular satisfying the requests that are still active is very important because they are death-related requests (see Schank,

1978). Thus we virtually ignore the rest of this phrase due to lack of interest.

MOST is skipped and stacked.

OF is skipped.

THE is skipped.

DAY is stacked and ignored. It also satisfies the low level request for a time word and this information is added to what we know about time to be used later if we ever get interested in what we have now decided is uninteresting.

****
before surrendering to French policemen...
****

BEFORE is a time ordering word that prepares us to set up a new event and mark its time relative to the preceding event.

SURRENDERING is a word that is both marked as of high interest and as part of a number of scripts including $TERRORIST. The surrender scene of $TERRORIST is instantiated and requests are fired off concerning the reasons for this action, his captors etc. Certain words are marked as indicating which of these might follow. Thus, 'because' marks off reasons, and 'to' marks off captors.

TO tells us that captors is coming.

FRENCH is held on a stack.

POLICEMEN is marked as a noun that can be an ACTOR so STM is

consulted to gather up its relevant components. POLICEMAN also is a possible captor (because it is both a human and an institution either of which would do), so its satisfies two extant requests.

****
and then was shot by Iraqi security officials....
****

AND says a new event is coming whenever an event has just ended.

THEN orders the time of the event.

WAS specifies that the actor stored in STM is the conceptual object of the new event. This sets up requests for the actor and the action.

SHOT is found to be interesting and is treated similarly to the way that GUNMAN was except that we do not expect the things that were particular to GUNMAN as opposed to the action he was performing. Thus we have:

WHO did he shoot? ---- GUNMAN
WHO shot? ---- not answered
WHY did he shoot? ---- not answered
WHERE did this happen? ---- already known
WHAT SCRIPTS does this instantiate? ----
SHOOT can also cause a
script to be instantiated.
Prime candidates are $ROBBERY,
$TERRORIST, $KIDNAP ordinarily.
But we are in a context set up by
$TERRORIST. None of the above are
normal continuations of $TERRORIST.
This causes us to look
for plans and goals.

WHAT were the RESULTS of this action?--
A request is set up to find the
results. If this request is not
satisfied the usual results of this
action are inferred. In this case
death for the object.

Since SHOT is interesting we need to explain it. No scripts are available here so we need to ask who would want to kill the GUNMAN and why. These requests are added to the active requests.

BY tells us to go on.

IRAQI is skipped and stacked.

SECURITY is skipped and stacked.

OFFICIALS is used to end the processing of the noun group. It satisfies the requests for WHO did the shooting and, as we now have an actor we ask about why he would kill a TERRORIST. This causes us to examine the themes we have for why TERRORISTS might be killed after capture. We have none around directly. This causes us to be surprised by this event. We seek to explain it by postulating a REVENGE or SHUT HIM UP type theme, but we are not sure of this of course.

****
as he was led away by French officers.
****

We are basically done now as no further requests need to be satisfied immediately. (We know this after we have seen a period and found no new requests.) We are still interested in the goals of each of the actors however so 'because' requests are still alive.

AS is known to be a time co-occurrence word. Since we are not interested in anything that occured at the same time unless it is itself interesting we can now skip ahead looking for actions or

actors that are interesting.

HE is skipped.

WAS is skipped.

LED is uninteresting and is both noticed and then skipped.

AWAY is skipped.

BY is skipped.

THE is skipped.

FRENCH is skipped.

OFFICERS is skipped because there are no requests asking for it.

The period tells us we are done

The final representation for this sentence is:

```
$TERRORISM
    ACTOR - Arab gunman
    PLACE - Paris Iraqi Embassy
    SCENES
      $HOSTAGES - some
      $CAPTURE
           ACTOR - French policemen
           OBJECT- Arab gunman
           PLACE-  Iraqi Embassy

UNEXPECTED RESULT: ACTOR- Iraqi officers
                   ACTION- PROPEL
                   OBJECT- Arab gunman
                   ITEM -*BULLETS*
                   DIR-FR- gun
                   RESULT
                     ACTOR- Arab gunman
                     STATE- dead
```

5. Word Types and Processing Strategies in Integrated Parsing

The parsing scheme that we have outlined is dependent on classifying the words in the dictionary in terms of what the parser should do with each word as it sees it. Thus, labels such as noun, verb etc. only make sense in a parser if they cause different processes to activate that are dependent on seeing such classifications.

It is very well to say, as we have, that a given word should be skipped or stacked or whatever. We must make these determinations beforehand, however. Thus the key issue in the realization of this parser is, first, the establishment of a set of categories for the words in the dictionary that will be useful in such a scheme; and, second, a procedure for determining what category a given word fits into. As we will see shortly, the category a word is assigned to may depend upon the domain we are working in.

Looking back at the example in the last section, we can see that there are basically three different things that can be done with a word when it is read. The first possibility is that it may simply be skipped and completely ignored. There are many words which have no significant conceptual content at the level of analysis that we are trying to model here. Examples from the story in the last section include the words 'most', 'way', and 'held'.

The second possibility that we can see from the example is that a word may be skipped, but saved in some kind of short term memory. Words for which this processing strategy seems appropriate have some significant conceptual content or meaning, but of a rather dull and uninteresting sort. Nevertheless, we cannot simply ignore them, because their meanings may be important in elaborating our knowledge of the events or things that we are interested in. For example, they may be used to fill roles in the conceptual structures representing interesting events. They may also never be used again. Many of the words in our example are processed this way. Examples include the words 'Arabic', 'Iraqi', and 'his', as well as all articles.

Two things can happen with these words. Either their meaning does help elaborate something interesting, in which case that meaning will be incorporated in the representation, or it doesn't. For example, the meaning of the word 'police' in the phrase

(1) before surrendering to French police

is incorporated into the representation because we are interested in who the terrorist surrendered to. On the other hand, the meaning of the word 'officers' in the phrase

(2) as he was led away by the French officers

is not incorporated into the meaning representation because it does not add to our knowledge of anything interesting.

Words with some conceptual content will often also have some associated processing information, in the form of expectations, which can help to elaborate on their own meaning. Many of the words which are processed by the "skip and save" strategy have these sorts of expectations. For example, it seems quite plausible that the word 'embassy' has an expectation which looks for the name of the country which the embassy represents, and that the words 'police', 'officers', and 'officials' have expectations for the name of the governmental authority in whose name they operate. But if a word is subject to the "skip and save" strategy, these expectations should not be applied until we know that the concept associated with the word actually elaborates on our knowledge of something interesting. If it turns out that we don't care about the concept, the work will have been wasted. Let's compare again our processing of 'police' in phrase (1) with our processing of 'officers' in phrase (2). Since it turns out that the concept of police in the first case adds to our knowledge of an interesting event, it seems plausible that the expectation that the word 'police' has for the authority governing the police would be used. In the second case, since the concept of 'officers' does not add to our knowledge of anything interesting, there is simply no point in applying any similar rule.

The third possible processing strategy we can apply to a word is to immediately pay attention to its meaning and to the expectations it generates. This is the strategy that we apply when the word has a significant and interesting conceptual

content. It is these concepts and their associated expectations that drive the analysis. Examples from the story of the last section include the words 'gunman', 'shot', and 'hostages'. The expectations which these words generate include the same kind of simple elaborative, or "slot-filling", expectations associated with some of the words for which a "skip and save" strategy is appropriate. For example, it is quite plausible that the word 'gunman' generates an expectation that looks for the nationality or political affiliation of the gunman.

These words can also generate expectations which operate at a much higher level. For example, when we read the word 'gunman', we expect to read that he may have performed the action of shooting a weapon. We also expect the events associated with several possible scripts, including $ROBBERY and $TERRORIST. These expectations operate in a manner somewhat akin to script application (see Cullingford (1978)), i that they serve to recognize events, and so recognize that they are sensible in the given context. So, as described in the example of the last section, once we know that the gunman is quite likely a terrorist, we expect that he may hold hostages, that he may shoot or kill some people, and that he may make demands. We also know that there are only a small number of possible outcomes of the episode: the terrorist might be captured, he might surrender, he might be killed, or he might escape. These high level expectations help us decide what is important in the text in a very top-down way. The analysis process depends crucially on this. But its flexibility also depends on its ability to pursue

questions about interesting things and events, even if they were not anticipated.

Within the broad categories of words that are not skipped, and words that are skipped and saved, there are some subcategories that help to decide how to process a given word. There are two considerations in the processing of a word that affect its appropriate classification. The first issue is how a given word modifies the representation we are building. Second, the kind of expectations that a word sets up are an important basis of classification. The following classification scheme is based on these two considerations.

A - Words that are not skipped

Within a theory of integrated parsing, words are best classified according to the type of conceptual structures that they build. That is, the most important role that a word plays, in this conception of processing, is not its syntactic role such as noun or verb, or even its conceptual role, such as actor or action. The most significant thing about a word from this point of view is how it affects the processing within the integrated understanding process.

In the representation given above for the Arab gunman sentence there are two different kinds of items. There are the events involved, - the terrorism script, the capture scene, the gunman being shot, and so forth. There are also the individual concepts that play roles; the gunman who fills the ACTOR slot of

the terrorism script, or the Iraqi embassy, which fills the LOCATION slot of this script, for example. These role fillers we shall refer to as tokens. With the distinction between tokens and events in mind, we can look at a classification of words.

A1 - Event Builders

One class of word are those that build event structures. We call these Event Builders (EB's) This class of words includes many verbs, and a number of nouns, such as 'killing', 'riot', and 'demonstration'. All EB's have an associated interestingness value. This value helps determine whether an event is significant enough to be included in the final representation, whether it is interesting enough so as to cause us to construe it as a central event in the representation, and whether it is important enough that we should spend valuable processing time attempting to fill its open slots. All EB's also have an associated set of expectations that help to guide the rest of the parse. These expectations vary from explicit requests to place subsequent items in specific slots, to general expectations about events that are likely to occur eventually (such as the scenes of a script).

EB's are further subdivided according to the type of event they build. Many very common words, such as 'give', 'went', and 'ate', build simple (and not very interesting) events. These events are the kind that we have always been able to represent very easily in Conceptual Dependency (Schank, 1972,1975). In our recent work on higher level knowledge structures, we have found

that the kinds of representations that are most significant are those that relate to scripts, plans, and goals (see Schank and Abelson, 1977). Consequently, those EB's that build simple Conceptual Dependency structures, are precisely those that need the least processing because they are the least interesting. They constitute a special class of EB's then, (CDEB's), that rarely require us to spend much time on them. They have rather simple expectations, generally to fill in their ACTOR, OBJECT, TO, FROM, and INSTRUMENT slots. In order for us to attempt to find the information that fills these expectations, some more interesting event must expect them, or there must be an interesting actor who we expect to be involved with the action.

Other kinds of EB's are script builders ($EB's) and scene builders (SEB's) Both of these types can have rather more involved requests, often suggesting events that might occur. The only real difference between $EB's (words such as 'hijacked', 'kidnap') and SEB's ('surrendered', 'convicted') is that from SEB's we try to infer a script, since scenes cannot occur in isolation, and from $EB's we try to create expectations for the scenes of the script.

Another type of event is the state. State EB's (STEB's), such as 'felt', 'died', and 'was' (in certain situations) are much like CDEB's in the requests they create. The structures they build express states of the world.

Other knowledge structures used to understand stories, such as plans, goals, and themes also have associated EB's (that is, words that build these structures directly) but the EB's described so far are sufficient for our present purposes. (Higher level knowledge structures are generally not stated directly by any particular word. Rather, the presence of such structures usually must be detected by inference.)

A2 - Token Makers

Many words, including most nouns, such as 'gunman' and 'embassy', contribute to the process of understanding by filling open slots in event structures. We call this class of words Token Makers (TM's). These words cause a token to be built. If the word is interesting, or an interesting modifier has been saved in short term memory, then the words which modify the TM are gathered up. The tokens built are frequently objects looked for by expectations made during the processing of previous words in the sentence.

The class of TM's can be subdivided in two ways. There are several different types of tokens which can built, such as actor tokens, place tokens, organization tokens, vehicle tokens and time tokens. The type of token built is one factor in determining whether the new token satisfies an expectation made earlier. Time TM's, which are words like 'today', 'yesterday', or 'morning', seem superficially different from the other types, but they can be processed in a fashion similar to other TM's. Normally a time TM will satisfy an expectation made by an event

looking for the time it took place.

The other subdivision of TM's concerns the effects that TM has on subsequent processing. This division is based on how interesting the TM is. Interesting TM's (ITM's) generate expectations as to what we might see next in the sentence. Thus 'gunman', an ITM, generates expectations for shooting, hijacking, and robbery events for example. ITM's that fill the actor role in an event naturally generate expectations that more information about these people will be forthcoming.

TM's that are not interesting, and hence do not generate any expectations, can be placed into two classes, standard (STM's) and empty (ETM's). STM's can easily be associated with objects already in memory, even though they are not interesting. Examples of STM's are 'airport', 'Vermont', and 'officials'. The tokens built by STM's can be used to fill slots in the representation. ETM's, on the other hand, are words which are so indistinct in memory that it is virtually meaningless to include them in the final representation of the sentence. Words such as 'people', 'place', and 'runway' fall into this class. These words build tokens which can deactivate expectations, but they are not added into the final representation. If there is no expectation for the token built, it is ignored in our parsing scheme since there is little reason to remember it.

B - Words that are skipped and saved

There is a class of words which is simply saved in short term memory when first read. Their processing can be completed later, if necessary. There are two important points to recognize about skip and save words. First, the fact that we save a word does not commit us to doing any further processing of it. Most skip and save words are not very interesting, and unless a subsequent interesting word requests that saved words be considered, skip and save words can easily require no processing other than being saved. Presumably the process of saving a word is very easy, so that skip and save words often consume very little processing time. An important point about skip and save words is that which words are skip and savable, and which are totally skippable, depends heavily on the domain and perhaps the current context. So for example, a word like 'tall', is totally skippable in most domains (such as most newspaper stories, including the example given above), but when reading a sports story, it may become a skip and save word, since height can be salient in certain situations.

The fact that words may only be skip and savable in certain domains may make the skip and save class appear larger than it really is. It seems quite plausible that even though few words are totally skippable in all domains, many words are skippable in most domains. In any particular domain, it seems as if the class of skip and save words will be relatively small, and the class of totally skippable words large. Ideally, we would like our parser

to move words between the skip and save and totally skippable classes dynamically, based on context. This is done to some degree in the current program, and is a technique which should be further explored.

The class of skip and savable words can be subdivided into several classes, based on what we do with the word, if we do decide to process it further. (Remember - there is a good chance no further processing will be done.)

B1 - Token Refiners

One class of skip and save words, token refiners (TR's), add information to the tokens built by TM's. Most of the words which commonly appear in noun phrases, including many adjectives, are TR's in domains in which they cannot be skipped entirely. Above, 'Arabic' is a TR which refines the actor token built for the gunman, by marking it "nationality: Arabic". The processing for all TR's begins in the same way. Each TR is stored temporarily, until the TM it modifies is found, at which point it may be processed further, in a manner dependent on the TR type. (If the TM proves to be uninteresting, no further processing will be done.)

TR's can be subdivided three ways, based on how they alter the tokens they modify. A large class of TR's simply add a property to a token. These TR's, which will be referred to as simple TR's (STR's) include common adjectives, such as 'red', 'tall', and 'Arabic', in the cases where they are not just

skippable. Words like 'early', or 'late', fall into this class, usually modifying time TM's.

Other TR's modify properties added to a token by another modifier. For instance, in the phrase "about 20 gunmen," 20 would add to the token for gunmen, "NUMBER 20," and 'about' would alter this to "NUMBER (APPROX 20)." Words in this class of TR's are called TR modifiers, or TRM's. It is not clear how often words in this class are not simply skippable. It seems likely that most of these words tend to get ignored nearly all of the time, but sometimes they must be skipped and saved.

The third class of TR's are names (TRN's). They simply add to the token they modify the information about the token's name. So in "Kennedy International Airport," 'Kennedy' adds to the airport token the fact that its name is Kennedy. TRN's differ in processing from STR's only in that they are additive, a token can be given a multi-word name, and that they cannot be modified by TRM's.

One aspect of processing which is common to all types of TR's is that they can increase the interestingness of the token they modify. Effectively they can turn a STM or ETM into an ITM. Or they can make more interesting an already interesting token. So, "Arabic gunman," is more interesting than 'gunman', due to the addition of the TR 'Arabic'.

B2 - Event Refiners

Event refiners (ER) are very similar to TR's, except they modify events, not tokens. Typical of this class are adverbs such as 'quickly', 'stupidly', and other 'ly' words. Other words such as 'here' and 'away' also fall into this class, since they alter a slot of the event they modify, as in "was shot here," or "was led away." Words which might appear to fall into this class are even more likely to turn our to be skippable than TR's. The 'ly' words just mentioned are ER's when they are saved, but in general they are very dull words, and get skipped entirely. As mentioned above, the determination of whether the word must in fact be saved is domain dependant. ER's divide into standard ER's (SER's) and ER modifiers (ERM's) in a manners similar to STR's and SRMs. Processing is similar to that for TR's, except it occurs when an event is created, and can look ahead in the sentence, as well as backward.

B3 - Function Words

There is an important class of words in English which have little or no meaning of their own, but exist solely to guide processing. These words, known as function words (FW's), are quite common, and include articles, prepositions, and auxiliary verbs. Function words in general cannot be totally skipped, but quite often the parsing process never returns to them. They must be saved, since if interesting items follow they may become important, but by themselves, they do not demand processing.

The role of articles (a, an, the) is to mark the beginning of noun phrases, and help indicate which TR's go along with which TM's. When read, they are saved with the TRs. Then, when processing a TM, we look back on the words just encountered trying to find TR's. If we find an article, this search terminates.

Prepositions (with, to, from ...) have a variety of functions in English. Often they precede TM's and indicate how the TM should be added to the structure being built. In our system, the most frequent use for prepositions is an inactive one. An EB will often create expectations for a certain preposition, with instructions for what to do with the TM following the preposition. Thus 'shot' creates an expectation for 'with', and knows that the TM following 'with' should go into the INSTRUMENT slot of the event. A few prepositions, such as 'to' and 'from', have slightly more active roles in processing. If there is no outstanding expectation for a 'to' or 'from', an attempt will be made to place the TM following that word into the TO or FROM slot of the preceding event, as appropriate.

Auxilliary verbs have a variety of functions, such as setting time (did go), or making the event to follow hypothetical (may go). One of the more difficult uses of auxiliary verbs is the use of 'was', (or 'is', less frequently) to make a verb passive. In this situation, we must alter the low level processes in an appropriate manner.

B4 - Relational Words

Relational words create a link between two events. Processing of all these words tends be the same. The word is saved temporarily until a significant event is found. Then the proper link between that event and the previous event is made. If the relational word connects uninteresting events in the sentence, no additional processing will be done.

Relational words create two main kinds of links - causal and temporal. Words such as 'before', 'while', and 'after' indicate temporal relations between events, and 'because', 'since', and 'therefore', indicate causal relations.

C - Skippable Words

A somewhat surprisingly large class of words is entirely skippable. When we process them, absolutely nothing is done. This is presumably one means for saving substantial amounts of time during processing. Words such as 'and', 'who', and 'speaking' (as used above) fall into this class. An important topic of future study is to discover just what qualifies a word as skippable. The larger the skippable class becomes, the faster this program will be. It is certainly clear that the class of skippable words depends upon the domain. As mentioned earlier, this is what allows the class of skippable words to be quite large, since words which cannot be skipped in all domains can, nonetheless, be skipped in many domains.

Also words can be added to the skippable class dynamically, even seemingly quite interesting words. So if we already know the "hold hostage" script is taking place, words like 'terror', 'siege', and 'gunfire', become skippable, since we have already inferred anything they would build.

6. Processing in More Detail

Now that we have defined the classes of words used in integrated partial parsing (hereafter IPP), we shall look in detail at how the parser we have written to implement IPP operates. This parser was written to handle a limited class of stories, namely newspaper stories about terrorism and related areas. We have not tried to address all the issues involved in parsing. Rather we have concentrated on the areas which are crucial to IPP. One obvious problem we have not addressed is words with multiple senses. Fortunately, in the class of stories we are processing most words, especially the interesting ones, have one strongly preferred sense. (Notice that there is no problem with words with several senses, all skippable.) The parser currently handles a limited set of example sentences, its primary purpose being to test out the theories we have been describing here. We are in the process of increasing its vocabulary and its domain of interest. The parser is written in LISP, and runs on a DEC System 20/50.

We will begin to describe the program by looking at several data structures it uses. Then we will look at how top down expectations are implemented, since they tend to guide the overall processing. Finally we will look at how words in the individual classes are handled.

One of the most frequently used structures in IPP is a stack of words which have been read, but not fully processed. This stack, which will be referred to as the word stack, allows the processing of some words, especially TRs to be deferred until an appropriate time. One characteristic of the stack is that words can only be put onto, or removed from, the top of the stack. Another data structure is a list of the active expectations, also referred to as requests. The expectations in this list are tested at specified points during processing, to see if any of them have been satisfied. There are also several lists built up, such as a list of interesting actors, and several temporary slots, such as the last event encountered, which are used by the program. These will be described as needed.

Expectations

The expectations which constitute much of the system's processing knowledge are implemented in the form of requests (see Riesbeck (1975)). A request is a form of production, or test-action pair. If the test of an active request is checked and found to be true, then the corresponding sets of actions are performed. A request is active if it is being held in the active expectations list we described in the last paragraph. The

process of checking the requests in this list is often referred to as request consideration. The list is ordered so that when the active requests are considered, the most recently activated are considered first, since they represent newer, and so possibly better, expectations.

While in theory the tests and actions which requests perform could be arbitrary, in our system we have found that only a restricted set is necessary. Requests may do the following:

build new conceptual structures -- usually a given request will only build one such structure;

fill a slot in some conceptual structure with some other conceptual structure -- for example, filling the ACTOR slot of $SHOOT with the token for the gunman;

activate other requests -- these will often be requests trying to fill slots in the structure built by the activating request; they can also be expectations for possible actions, states, or more complicated episodes which may follow;

de-activate requests -- requests should be able to deactivate requests, including themselves, when they are no longer appropriate.

There are three types of tests which requests perform:

checks for specific lexical items -- for example, function words tend to be specific to a given construction; so in the phrase "surrender to French police", the requests associated with 'surrender' (or $SURRENDER), can look for the occurrence of the word 'to' to precede the authority to whom the surrender is taking place;

checks for lexical items satisfying some property -- for example, words which activate a specific script;

look for tokens or events of a specified type -- this might be as simple as matching a particular structure; or it may involve use of semantic tests such as 'human' or 'authority'.

The fact that requests can look for specific lexical items is very important in reducing processing time. This savings is

realized both by requests looking for function words to fill slots (such as 'to' or 'by'), and by requests which look for more substantial events. Often a word may create expectations which look for specific words which indicate what script is applicable. As an example, gunman creates expectations which look for the terrorist, hijack, and robbery scripts. The request looking for the hijack script may include tests for specific words (or phrases), such as 'diverted', 'hijack', 'took over', all of which indicate the hijack script. Requests will normally have checks at the conceptual level as well. The request activated by 'gunman' which checks for the terrorist script looks at the location of the action. If that location is the location of a political entity, such as an embassy or the office of some political organization, that is a good clue that the terrorist script may be relevant.

Sample Request

```
~ FIND-$HIJACK instantiates the hijacking script by noticing an
~               appropriate word or concept, builds a hijacking event,
~               and sets off several new requests, looking for scenes
~               and other actions. (Created by GUNMAN.)

(DEF-REQ FIND-$HIJACK
  TEST     (HIJACK-INSTANTIATOR *NEW-ITEM*)  ~ Test looks for words which
                                             ~  indicate the hijack script
  ACTION   (REQ-EVENT 8 (SCRIPT $HIJACK      ~ Action builds an event for
                         ACTOR       NIL     ~  the hijack script of
                         DEMANDS     NIL     ~  interest 8, with the slots
                         FROM        NIL     ~  shown here. It fills in
                         DESTINATION NIL     ~  the actor slot with the
                         TO          NIL     ~  last actor in *ACTOR-STACK*
                         PASSENGERS  NIL
                         VEHICLE     NIL)
                   ((ACTOR . (TOP-OF *ACTOR-STACK*)))
                    (REDUNDANT-HIJACK-WORDS    ~ These new requests are
                     FIND-HIJACK-DESTINATION   ~  activated.
```

FIND-HIJACK-VEHICLE
FIND-HIJACK-PASSENGERS
FIND-HIJACK-EVENTS
SURRENDER-SCENE
RECOGNIZE-DEMANDS
RECOGNIZE-COUNTER-MEASURES]

A1 - Event Builders

EBs build the event structures which provide the framework for the final representation. They also create expectations for what might follow. When an EB is read, an empty event structure is built from a template in the dictionary, with no slot values, except possibly some defaults, filled in.

Now the system checks to see if any requests are looking for this event. If not, the event's interest value, listed in the dictionary, is checked. If the interest is 0, the event is so dull it is not worth worrying about, and processing moves to the next word. If the event has a positive interest value, the expectations listed in the word's dictionary entry are instantiated, with the new event structure substituted in the proper places, so the requests created (which are added to the request list) can fill slots in the event structure.

The IPP system keeps a global slot called *MAIN-EVENT*, which at the conclusion of the sentence holds the most important event found. When a new event is created with interest value greater than that of *MAIN-EVENT*, or if

there is not yet a *MAIN-EVENT*, then the new event is placed in *MAIN-EVENT*. If an event with positive interest value lower than the interest value of *MAIN-EVENT* is created, and it does not fulfill an expectation, then the new event is saved in *UNEXPLAINED-EVENTS*, to indicate it should be explained later.

Sample Dictionary Entry

```
(WORD-DEF OCCUPIED
  INTEREST 5
  TYPE     EB
  SUBCLASS SEB
  TEMPLATE (SCRIPT   $DEMONSTRATE          ~ OCCUPIED builds a structure
            ACTOR    NIL                   ~   specifying the demonstrate
            OBJECT   NIL                   ~   script with an occupy
            DEMANDS  NIL                   ~   scene
            METHOD   (SCENE    $OCCUPY
                      ACTOR    NIL
                      LOCATION NIL))
  FILL     (((ACTOR)        (TOP-OF *ACTOR-STACK*))  ~ ACTOR slots
            ((METHOD ACTOR) (TOP-OF *ACTOR-STACK*))) ~  are filled.
  REQS     (FIND-DEMON-OBJECT  ~ Expectation we might see who is being
                               ~   demonstrated against.
            FIND-OCCUPY-LOC    ~ Expectation we might see the site of
                               ~   the demonstration.
            RECOGNIZE-DEMANDS] ~ Expectation we might see demands.
```

A2 - Token Makers

The discovery of a TM normally initiates a standard sequence of processing. The only exception to this, is when the TM is followed by another TM, in which case the first one is treated as a TR modifying the second TM, making it a skip and save word. Processing of a TM begins with the creation of a token to represent the word. The token is marked with its type (e.g. actor, place, etc.), as

indicated in the dictionary. If the TM is interesting, or interesting words are on the word stack (and only in these cases), then the TRs which modify the token are collected from the stack. This process consists of popping words off of the word stack one at a time, handling each TR popped as indicated in the next section. This process terminates when the word stack is empty, or when a function word which normally marks the beginning of a noun group (such as an article) is reached. Note that not all the words in a noun phrase will be on the word stack at the time the TM is reached - only those classified as skip and stackable. This means adjectives which are skippable get totally ignored.

Once the above processing is completed, either by collecting modifiers, or deciding the word stack can be ignored, the token is classified as interesting, standard, or empty. This determination is usually dependent upon the interestingness of the TM which created the token, but can be affected by one or more TRs. At this point the system checks to see if the new token satisfies any expectations. If so, and the token is not empty, then the request performs its action to the representation, and the request is deactivated. If the token is from an ETM which has not become more interesting, then the request is simply deactivated.

If no expectation grabs the new token, and the token is empty or standard, then processing of the TM is complete. For interesting tokens, more must be done. Specifically, new expectations must be set up. This is done by referencing the list of expectations included in the ITM's dictionary entry. Requests are created for each of these expectations, connected to the new token so that each request can use or modify the token. An example of this is that 'gunman' instantiates several requests which look for feasible scripts, such as the terrorism script, the hijacking script and the robbery script. It also creates a request which looks for words which indicate that a shooting has occurred. Words like 'shot' become partially redundant, since we now from the word 'gunman' that a shooting event is a likely possibility.

Sample Dictionary Entry

```
(WORD-DEF GUNMAN
  INTEREST 5                    ~ GUNMAN is an ITM
  TYPE     TM
  SUBCLASS ACTOR
  MEMORY   T
  REQS     (CONFIRM-SHOOT       ~ Expectation we might see a shooting.
            FIND-WHY-SHOOT      ~ Expectation we might see why the
                                ~   gunman would shoot someone.
            (FIND-$TERRORIST    ~ Set of expectations which specify
             FIND-$ROBBERY      ~   scripts we are likely to see.
             FIND-$KIDNAP       ~   If one is satisfied, the others
             FIND-$HIJACK]      ~   are deactivated.
```

B1 - Token Refiners

When a TR is read, it is added to the top of the word stack. If the TR is interesting, then the system recognizes that the word stack must be examined when the next TM is encountered, even if the TM itself is not interesting. When the next TM is reached, the word stack may be popped, word by word, and the processing of TRs will be completed. If neither the next TM, nor any TRs which have been stacked are interesting, then stacked TRs are simply ignored.

When an STR is popped, the property identified in its dictionary entry is set to set to the value in the entry on the token being modified. For instance, when 'Arabic' is popped back off the word list while doing the Arab gunman example, the token being modified has its NATIONALITY property set to ARABIC.

When a TRM is popped, the property identified in its dictionary entry is modified by the value in the entry on the token being modified. An example of this is "about 20 gunmen," described earlier.

When a TRN is popped, the value in its dictionary entry is added to the NAME property of the token being modified.

Part of the processing of all TRs is a check to see if the TR should increase the interestingness of the token, or should raise the status of an ETM token to that of an STM (i.e. check whether the TR identifies the TM with an item in memory).

Notice that this scheme would make it very easy to handle TRs whose meaning is dependent upon the words they modify, since the actual definition of the TR is not processed until the TM is known. It also simplifies cases where the TM actively looks for specific types of words which might modify it.

Sample Dictionary Entry

```
(WORD-DEF ARABIC
  TYPE     TR
  SUBCLASS STR
  MEMORY   T
  DEF      (NATIONALITY . ARAB]
```

B2 - Event Refiners

Most of the processing of ERs is just like that for TRs. When read, they are put on the word stack. The difference from TRs is that they get popped off the stack when an event is built, and they modify the event structure just built. The actual processing of an SER popped from the stack by an event is identical to a STR popped by creation of a token. The same is true for ERMs compared to TRMs.

The other difference between ERs and TRs is that and ER can appear after the event which creates it. So when an event is created a check is made to see if the words which follow immediately in the sentence include one or more ERs. If so, the are removed from what's left of the sentence (i.e. their processing is considered complete), and handled just as if they had been on the word stack.

Sample Dictionary Entry

```
(DEF-WORD AWAY
      TYPE     ER
      SUBCLASS SER
      DEF      (TO . NOT-HERE]
```

B3 - Function Words

The sole function of articles in this system is to mark the beginning of noun phrases. When an article is read, it is placed on the word stack. Then when words are being popped off the stack during TM processing, and the article is reached, i.e. popped off the stack, the popping of the word stack terminates.

As mentioned earlier, the role of prepositions is largely a passive one. Various events will create expectations for different prepositions, with instructions for what to do with the TM following each preposition that might be found.

The only auxiliary verb which the system pays any attention to at the moment is 'was', used to mark a passive construction. This word is handled by setting up an expectation which looks for the next event, and alters the processing which will fill in various slots.

Sample Dictionary Entry

```
(WORD-DEF A
  TYPE     FW
  SUBCLASS ART]
```

B4 - Relational Words

When a relational word is encountered, it is saved in the temporary slot *RELATION*. Its processing is continued when the next event is encountered. When an interesting event structure is about to be created, a check is made to see if *RELATION* is filled, and at least one other event has already been encountered. If so, then the type of link specified in the dictionary for the relation word saved in *RELATION* is made between the new event structure and the last event structure created. *RELATION* is also cleared. If the check fails, nothing is done. This allows sentences such as "After the gunman shot the policeman, he blew up the plane," to be processed correctly, as well as "The gunman blew up the plane after he shot the policeman."

Sample Dictionary Entry

```
(WORD-DEF BEFORE
  TYPE     RW
  SUBCLASS TRW
  RELATION AFTER]
```

C - **Skippable Words**

The processing of a skippable word is totally trivial. We notice it is skippable, do nothing, and proceed to the next word. The fact that the processing of this large class of words is so simple is one of the keys to IPP.

7. Examples of IPP

What follows are computer runs of our IPP parser on three sentences taken from the New York Times.

```
 TOPS-20 Command processor 3(414)
@DO IPP
@LISP


*(PARSE S1)

Input:

(AN ARABIC SPEAKING GUNMAN SHOT HIS WAY INTO THE IRAQI EMBASSY HERE
THIS MORNING HELD HOSTAGES THROUGHOUT MOST OF THE DAY BEFORE
SURRENDERING TO FRENCH POLICEMEN AND THEN WAS SHOT BY IRAQI SECURITY
OFFICIALS AS HE WAS LED AWAY BY FRENCH OFFICERS)
```

Output:

```
** MAIN EVENT **
EV1 =
 SCRIPT     $TERRORISM
 ACTOR      ARAB GUNMAN
 PLACE      IRAQI EMBASSY
 INTEREST   9.
 CITY       PARIS
 TIME       MORNING
 SCENES
    EV2 =
     SCENE      $HOSTAGES
     PLACE      IRAQI EMBASSY
     ACTOR      ARAB GUNMAN
     INTEREST   7.
     TIME       DAY
    EV3 =
     SCENE      $CAPTURE
     PLACE      IRAQI EMBASSY
     OBJECT     ARAB GUNMAN
     ACTOR      POLICEMEN
     INTEREST   6.
     AFTER      EV2

** UNEXPECTED EVENTS **
EV4 =
 ACTION     PROPEL
 ACTOR      IRAQI OFFICIALS
 OBJECT     ARAB GUNMAN
 ITEM       *BULLETS*
 DIR-FR     GUN
 INTEREST   5.
 AFTER      EV3
 RESULT
    EV6 =
     STATE      DEAD
     ACTOR      ARAB GUNMAN
     INTEREST   4.
```

673. msec CPU (0. msec GC), 944. msec clock, 1759. conses

```
*(PARSE S2)

Input:

(A GUNMAN WHO DIVERTED A VERMONT BOUND BUS WITH MORE THAN TWENTYFIVE
PASSENGERS FROM THE BRONX TO KENNEDY INTERNATIONAL AIRPORT AND KILLED
TWO HOSTAGES SURRENDERED ON A RUNWAY LATE LAST NIGHT ENDING A DAYLONG
SIEGE OF TERROR AND GUNFIRE)

Output:

** MAIN EVENT **                    ** UNEXPECTED EVENTS **
EV7 =
 SCRIPT    $HIJACK                   NONE
 ACTOR     GUNMAN
 FROM      BRONX
 TO        AIRPORT
 CARRYING  PASSENGERS
 VEHICLE   BUS
 INTEREST  8.
 RELATED
    EV8 =
     ACTION    DO
     ACTOR     GUNMAN
     RESULT
        EV9 =
         STATE     DEAD
         ACTOR     HOSTAGES
     INTEREST  5.
 SCENES
    EV10 =
     SCENE     $CAPTURE
     OBJECT    GUNMAN
     ACTOR     POLICE
     INTEREST  6.
     TIME      NIGHT

718. msec CPU (0. msec GC), 755. msec clock, 1598. conses

*(PARSE S3)

Input:

(ABOUT TWENTY PERSONS OCCUPIED THE OFFICE OF AMNESTY-INTERNATIONAL
SEEKING BETTER JAIL CONDITIONS FOR THREE ALLEGED WEST-GERMAN
TERRORISTS)

Output:

** MAIN EVENT **                    ** UNEXPECTED EVENTS **
EV11 =
 SCRIPT    $DEMONSTRATE              NONE
```

```
OBJECT    AMNESTY-INTERNATIONAL
DEMANDS   IMPROVED JAIL CONDITIONS FOR WEST-GERMAN TERRORISTS
METHOD
   EV12 =
    SCENE     $OCCUPY
INTEREST  5.

424. msec CPU (0. msec GC), 467. msec clock, 465. conses

[PHOTO:  Recording terminated  Thu 16-Nov-78 8:27AM]
```

8. Conclusion

Careful readers will note that we have used little in the way of Conceptual Dependency (Schank, 1972, 1975) in the final representations that we have used as the output of our parser. This represents a shift in our thinking about representations that has been going on for the last few years. In Schank and Abelson (1977), we proposed an additional level of representation, called the Knowledge Structure level, that represented larger structures of information than were available in our original view of Conceptual Dependency. In Schank and Carbonell (1978), we proposed yet another addition to our representational system to handle social and political acts that were handled rather poorly in the previous systems. We have, of course known that were a great many issues that could not be adequately represented in Conceptual Dependency. The need for additional representational schemes has been, and still is, obvious. But previously, we have always attempted to parse into

Conceptual Dependency first, preferring to write our inference mechanisms so as to begin with input represented in Conceptual Dependency. This had two main advantages. First, it allowed the large number of people, and programs that they built, that were working in our project to be able to communicate with one another. Conceptual Dependency was a kind of interlingua, or conceptual Esperanto in terms of which everyone could communicate. Secondly, aside from this pragmatic advantage, we believed that this kind of modularity was correct from a theoretical point of view. Simply stated, we believed that meanings were extracted from sentences and then operated upon by other processes.

The obvious proposal when we invented the two additional representational systems referred to above was to attempt to parse into them directly. Although we still believed that people extracted meanings from what they heard, there really was no reason to believe that these meanings could have one and only one form. If 'want' was best represented in a goal related fashion and rather complexly represented in Conceptual Dependency, what reason was there to believe that one had to go through the complex form to get to the simple one? Much of this kind of issue has formed the basis of various researchers objections to our notion of primitives. In particular, Bobrow and Winograd (1977) have made an issue of our primitives from time to time. They have proposed a notion of variable depth of processing as a

counterproposal to our primitive representations. In a sense the system we have described here makes use of that suggestion. Bobrow and Winograd are correct when they assert that different levels of processing make sense at different times. We disagree with them on the issue of what constitutes the appropriate set of levels. We do not believe in either words themselves or syntactic notions as sensible stopping points at any point. But the absence of Conceptual Dependency in parts of our final representations here concedes the larger point. That is, we agree that one ought to go as far as one needs to during the understanding process.

What then of Conceptual Dependency and primitives? In many places in our parser, Conceptual Dependency is used as a kind of internal language providing important information to components of the parsing process that never actually surfaces in the final output. Strangely enough, it has begun to bear a certain similarity to our use of syntax in the parsing process. That is, it is something that is there behind the scenes doing its job without ever surfacing much.

The major conclusion of all this then is that we believe that modular systems will eventually fall apart from there own cumbersomeness. Real life understanding systems must be integrated to the extent that they can be guided by their inherent interests, delving into what they fancy and skipping what they do not. This must be truly what is meant by variable

depth of processing. Another way of saying this is that if we actually pay equal and detailed attention to everything we are called on to understand, we may never finish the understanding process. To get all the inferences and relevant knowledge structures out all the time may be at the worst impossible and at the best unrealistic in terms of processing time. A language understander is guided by what he wants to know (and what he does not want to know). This enables him to not see all the ambiguities, triple meanings, myriad implications and other problems with what he hears. But what he loses in perfection he more than makes up for in lack of fragility and speed. Perhaps it is time to give our machines the same advantages.

References

Bobrow, D. G., and Fraser, J. B. (1969) An augmented state transition network analysis procedure. Proc. Int. Joint Conf. on AI 1

Bobrow, D. G., and Winograd, T. (1977) An overview of KRL, a knowledge representation language. Cognitive Science 1, no. 1

Carbonell, J. G. Jr. (1978) Ph. D. thesis, Yale University Department of Computer Science (in progress).

Cullingford, R. (1978) Script application: computer understanding of newspaper stories. Research Report 116, Department of Computer Science, Yale University.

DeJong, G. F. (1977) Skimming newspaper stories by computer. Research Report 104, Department of Computer Science, Yale University.

Gershman, A. (1977) Analyzing English Noun Groups for Their Conceptual Content. Research Report 110, Department of Computer Science, Yale University.

Granger, R. H. (1977) FOUL-UP: A Program That Figures Out Meanings of Words from Context. Fifth International Joint Conference on Artificial Intelligence, August 1977, Cambridge, Massachusetts.

Kaplan, R. M. (1975) On process models for sentence analysis. In D. A. Norman and D. E. Rumelhart, eds., Explorations in Cognition. W. H. Freeman and Company, San Fransisco.

Lamb, S. (1966) Outline of Stratificational Grammar. Georgetown University Press, Washington, D.C.

Marcus, M. (1975) Diagnosis as a Notion of Grammar. Theoretical Issues in Natural Language Processing, June 1975, Cambridge, Massachusetts.

Newell, A. (1973) Production systems: models of control structures. In Chase, W.C. (ed.), Visual Information Processing, Academic Press, New York.

Petrick, S. R. (1973) Semantic Interpretation in the REQUEST System. IBM Research Report RC 4457, IBM Corporation, Yorktown Heights, New York.

Riesbeck, C. K. (1975) Conceptual analysis. In R. C. Schank (ed.), Conceptual Information Processing. North Holland, Amsterdam.

Riesbeck, C. K. and Schank, R. C. (1976) Comprehension by computer: Expectation-based analysis of sentences in context. Research Report 78, Department of Computer Science, Yale University.

Schank, R. C. (1972) Conceptual Dependency: A theory of natural language understanding. Cognitive Psychology, Vol. 3, No. 4.

Schank, R. C. (1975) Conceptual Information Processing. North Holland, Amsterdam.

Schank, R. C. (1978) Interestingness: Controlling Inferences. Research Report 145, Department of Computer Science, Yale University.

Schank, R. C. and Abelson, R. P. (1977) Scripts, Plans, Goals and Understanding. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Schank, R. C. and Carbonell, J. C. (1978) Re: The Gettysburg Address: Representing Social and Political Acts. Research Report 127, Department of Computer Science, Yale University.

Schank, R. C. and Selfridge, M. (1977) How to Learn / What to Learn. Fifth International Joint Conference on Artificial Intelligence, August 1977, Cambridge, Massachusetts.

Schank, R. C. and Tesler, L. (1969) A conceptual parser for natural language. Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C.

Schank, R. C., Tesler, L., and Weber, S. (1970) Spinoza II: Conceptual case-based natural language analysis. Stanford Artificial Intelligence Project Memo No. AIM-109, Computer Science Department, Stanford University, Stanford, California.

Schank, R. C. and Yale A. I. Project (1975) SAM -- A story understander. Research Report 43, Department of Computer Science, Yale University.

Thorne, J., Bratley, P., and Dewar, H. (1968) The syntactic analysis of English by machine. In D. Michie (ed.), Machine Intelligence 3, American Elsevier Publishing Company, New York.

Wilks, Y. (1973) An artificial intelligence approach to machine translation. In R. C. Schank and K. Colby, eds., Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Winograd, T. (1972) Understanding natural language. Academic Press, New York.

Woods, W. A. (1969) Augmented transition networks for natural language analysis. Rep. CS-1, Computer Lab, Harvard University, Cambridge, Massachusetts.