An Integrated Understander

Roger C. Schank
Michael Lebowitz
Lawrence Birnbaum

Department of Computer Science
Yale University
New Haven, Connecticut 06520

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.

1. Overview of Conceptual Dependency Parsing

Over the course of the last ten years, researchers in our project have designed and programmed a large number of parsers. The task of these parsers was the initial mapping of natural language into an internal representation. (Note that the first phase of the understanding process traditionally refers to the discovery of the syntactic form of the input. However, the term "parsing" can just as meaningfully be applied to whatever the first phase of understanding might be.) In this paper we will discuss some of the problems which have arisen in the development of parsers, and present a new theory of the way parsing works in the normal reading process. We will describe a program which implements this theory and understands newspaper stories about terrorism.

All our parsers were programs that mapped English sentences into the Conceptual Dependency (CD) representation of their meaning. Underlying their construction was always the methodological assumption that the parsing algorithm that they were to employ was to be as psychologically correct as possible. Thus, our parsers are intended to model the way we believe people parse. 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. These parsers were not designed to handle true "garden path" sentences where people have to backtrack.

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 our parsers' dependence on syntax.

0362-613X/80/010013-18$01.00

While these parsers were being developed, other researchers were devising methods for parsing natural language input into an internal representation. Most 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 primarily 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 Plath (1974). A more recent parser which views syntax as an isolatable sub-part of language understanding is in Marcus (1979). The important thing to note about all of these programs is that they view syntactic parsing as a process totally isolated from the rest of understanding. Syntax drove these systems, although in some cases the syntactic parser was allowed to request semantic information. Some speech understanding systems, Hearsay-II in particular (CMU Computer Science Speech Group, 1976) use a more integrated approach, but they are only marginally concerned with the level of processing that we are interested in. 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 SAM our script-based understanding 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. 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 (1979) POLITICS program, a model of subjective understanding of political events, 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 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.

It should be emphasized what we mean when we say that modularity was a handicap in parser development. Clearly from a programming point of view our parsers must be modular. However, if modules seem to be spending all of their time communicating with each other, then the particular modularization scheme must be suspect - the modules really form an integrated unit. Since communicating amongst modules tends to be hard, the tendency is to avoid it.

This can result in processes which should interact strongly becoming isolated from each other. This is what happened with the modularization strategy we have described.

2. Paying Attention To Less Than Everything

One of the major problems with SAM, and also with ELI as a part of SAM, was its inability to handle 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. 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 process 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 - usually the items of information it wishes to include in its summary for any particular domain that it has knowledge about. FRUMP is thus a highly top-down system and for this reason it cannot be considered as a replacement for SAM. SAM could, in principle, respond to inputs it was unprepared for, although in practice this did not happen very often! FRUMP cannot respond to aspects of stories it is unprepared for; but then neither is it unable to process such stories at all.

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. FRUMP works as well as it does because its interests guide what it looks for. It can ignore what it is not interested in and concentrate on what it wants to know.

Now let's consider how a normal, literate adult reads a story, a newspaper story, for instance. We have considered seriously the question of whether a human reads in detail, like SAM, or skims, like FRUMP, in his normal reading mode. And, although we possess no hard evidence one way or the other, we now feel that a human is more FRUMP-like than we previously believed. 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 significant aspects of many stories. On the other hand, some kind of combination of FRUMP parsing and ELI parsing might make for a very powerful and robust system for story understanding.

3. Time of Processing in Parsing

One of the major factors to be considered in discussions of the design of a human-like parser 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. 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. This is confirmed by psychological evidence such as Marslen-Wilson (1975) which uses errors in the shadowing of sentences to show high-level processing must be occurring throughout the reading of a sentence. Such a conclusion certainly makes the argument we were stating above much more significant. It implies that human processes are highly integrated. 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 processing 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. As we have discussed elsewhere (Schank, 1975, for instance), the process of doing a complete parse is extremely complex. Simply stated, it takes n milliseconds to read a word and it takes m milli-seconds to completely 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 completely 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.

Intuitively, 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 the amount of processing time available, that the processing of some words must 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.)

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 are probably not being processed at all (or in any case they are processed so partially that they are hardly seen). 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. Then top down processing helps the understander know what to ignore. According to this scheme, 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 their analysis completed. Words which initiate processing usually appear at the end of phrases or breath groups.

In order to process a noun phrase such as "small twin engine airplane" 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, the meaning of 'carrying' can be virtually ignored, since as we are only beginning to recognize what word it is, we hear about the marshals and the murderer and decide 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.

If certain elements of language are skipped entirely, or processed only slightly in the course of normal understanding, one might ask why they appear at all. Why doesn't the author just omit them? The answer to this is that the same story can be understood by different readers with varying degrees of completeness. It is possible for a reader paying attention to every detail of a story to discover nuances that a partial understander like the one we

are describing would miss. However, it is our belief that most of the time, when dealing with media such as newspapers, people do not do extensive processing, and yet are able to extract the vast majority of information of interest. That is the process we are attempting to model here. We do not claim that it represents the only level of processing a person can use, but we do believe it is a very important and widely used level of understanding.

(There is an aside 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 completely processing 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 episodic 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 nearly always interesting, others are interesting in certain circumstances. More importantly, certain things are interesting on the basis of what has preceded them - interestingness is a dynamic property. 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 a dynamic 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": Memory is accessed to see if X is interesting because it is a main noun and because it is a person. When no information is found, the processing should be faster than 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 specific meaning in isolation. (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 go on.) 'Visit' is a word that calls up a script ($VISIT) if the object of the visit is an equal or a family member. But 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 purpose is to get us interested in the object that follows next. That is, we don't really start processing this sentence deeply 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 in terms of script (and thus processing). In that case, either $VISIT or the ROMANCE script would be applicable, and we will now want to figure out which.

Notice that while most of the examples we have presented, and those we will present, describe stories in terms of scripts, there is no reason our processing ideas could not be used to handle stories better represented by other knowledge structures, such as plans and goals (Schank and Abelson, 1977, Wilensky, 1978). In fact, we believe they will apply generally to whatever is represented in memory. Scripts are prominent in our early development of

our parsers, because the domain we are concentrating on, newspaper stories, tends to involve many stereotypical situations. However, we expect to extend this work to handle stories requiring different types of representations.

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 policemen 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 we desire in an integrated understanding scheme. The program we will describe later does such processing. Our model will skip the uninteresting parts and build up its representation when necessary, attempting to be completely finished with each sub-part at the right time. That is, we desire that the model we propose be finished processing only slightly after the input has been received - just as a person would do.

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 believe that human understanding 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. Thus, for example, the next word to be read is available for any word under consideration. Such an assumption can simplify the problem of disambiguation.

****

An Arabic speaking gunman...

****

AN is a word that need only be saved 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 form 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. (That ARABIC is skippable has already been determined and is simply looked up. 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 an actor. Also, adjectives designated as interesting 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 (e.g. ARABIC) 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

WHOM 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 $ROBBERY, $TERRORISM and $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. We are actually performing the entire process of understanding this story 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 a SHOOT action with the gunman token as actor and an unfilled final direction for the shooting. This unfilled slot is marked as the same one that satisfies the answer to the WHO question asked before and the parser now is inter-

ested 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 dictionary entry says to keep on going and it is skipped.

THE is saved and skipped.

IRAQI is saved and skipped.

EMBASSY is found to be a location and is set up as the location of the SHOOT event. Furthermore, EMBASSY is marked as interesting and a place of political significance. This latter piece of information satisfies the request for instantiating the $TERRORISM 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. IRAQI is thus found in STM as filling this request and is picked up.

Setting up $TERRORISM 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 refers to the dateline location in a news story. It adds this location to the story representation.

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 predicted so we skip it. What could have changed this would have been some interest marking under HELD or other item of significance. Note that 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 $TERRORISM 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 the meaning of HELD never had to be determined in isolation, 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 understood through $TERRORISM. Thus integrated understanding plus "save and skip" 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 saved and skipped.

OF is skipped.

THE is saved and skipped.

DAY is recognized 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 $TERRORISM. The surrender scene of $TERRORISM 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 in STM.

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 it satisfies two extant requests.

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

AND says whenever an event has just ended, a new event may be coming.

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:

WHOM did he shoot? ---- ARAB 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, $TERRORISM and $KIDNAP, ordinarily. But we are in a context set up by $TERRORISM. None of the above are normal continuations of $TERRORISM. 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 the actor is to follow.

IRAQI is stacked and skipped.

SECURITY is stacked and skipped.

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. At this point we might make the connection between IRAQI SECURITY OFFICIALS and IRAQI EMBASSY, but that does not lead to an explanation. 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 certainly not sure of it.

****
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 WHY requests are still alive.

AS is known to be a time co-occurrence word. Since we are not interested in anything that occurred at the same time unless it is itself interesting, we can now skip ahead looking for actions or actors that are interesting. AS can also indicate causality, but in that case the semantic predictions set up earlier will find the cause.

HE is skipped.

WAS is skipped.

LED is uninteresting and is recognized 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 - Paris, Iraqi Embassy
UNEXPECTED RESULT:
    ACTOR - Iraqi officers
    ACTION - SHOOT
    OBJECT - Arab gunman
    RESULT
        ACTOR - Arab gunman
        STATE - dead

5. Processing in Integrated Partial Parsing

We have written a program which implements the theory of parsing illustrated above - an Integrated Partial Parser (hereafter IPP). In this section we shall look at how this parser works. It 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. IPP has successfully processed over 200 stories taken directly from various newspapers. Many of these were processed sight unseen. IPP current has a vocabulary of over 2000 words. The parser is written in LISP, and runs on a DEC System 20/50.

The program's limitations center around its vocabulary and knowledge of the world. 2000 words, a sizable vocabulary for a typical AI program, is still a bit too small, even for stories about terrorism. We are currently expanding the vocabulary. The program is also limited to understanding stories for which it has appropriate world knowledge. We have concentrated on stories that are script-based, but as mentioned earlier, we believe the general techniques of IPP will extend to other forms of knowledge. IPP's ability can be increased both by adding more script-type information, similar to that it already has, and by considering these other types of world knowledge.

Another limitation IPP has is that it has problems with stories which are subtly phrased - those where jumping to a conclusion causes problems. But these are exactly the kinds of stories people have trouble with when they are reading quickly. The projected solution for this problem is to give IPP the capability of going back and reading text in a more careful mode than it normally does.

The parsing scheme implemented in IPP is based on classifying the words in the dictionary in terms of what the parser should do with each word as it reads it. Thus, labels such as noun, verb, etc. only make sense in a parser if they cause different processing dependent on seeing such classifications.

It is very well to say, as we have, that a given word should be skipped or saved or whatever. We must make these determinations beforehand, however. Thus the key issues in the realization of this parser are, 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 be domain dependent.

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. It can be skipped, it can be saved and then skipped, or it can be completely processed immediately.

The first possibility is that it may simply be skipped. There are many words which have no significant conceptual content for normal reading. 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 saved in some kind of short term memory and then skipped. Words for which this processing strategy seems appropriate have some functional purpose or significant conceptual content 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 'French' in the phrase
(1) before surrendering to French police
is incorporated into the representation because we are interested in whom the terrorist surrendered to, i.e. 'police'. On the other hand, the meaning of the word 'French' in the phrase
(2) as he was led away by the French officers
is not incorporated into the meaning representation because we never become interested in 'officers'.

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 "save and skip" strategy are objects of 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 "save and skip" 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, we don't want to have done unnecessary processing. Let's compare our processing of 'police' in phrase (1) above 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 process it immediately, i.e. pay attention to its meaning and 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 one expectation generated by the word 'gunman' 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 $TERRORISM. These expectations operate in a manner somewhat akin to script application (see Cullingford, 1978), in 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.

The expectations used by IPP 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. The list of requests is ordered so that when the active requests are considered, the most recently activated are considered first, since they represent newer, and so probably 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 are 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. A sample request is shown on the next page.

Within the broad categories of words that are processed immediately, and words that are saved and skipped, there are subcategories that help to

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]
```

decide what to do with a given word. There are two considerations that affect a word's classification. The first is how a given word modifies the representation we are building; the second is the kind of expectations that a word sets up. The classification scheme is based on these two considerations. We will now describe each class of words, and how IPP processes them. For each class, a sample dictionary entry is shown.

A - *Words that are immediately processed*

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 words 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 'hijacker'. All EB's have an associated interestingness. This helps determine whether an event is significant enough to be included in the final representation - whether it is interesting enough 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 intrinsically 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 whom 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 create expectations for the scenes of the script.

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 a large class of newspaper stories. (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.)

When an EB is read, an empty event structure is built from a template in the dictionary. IPP then checks to see if any requests are looking for this event. Expectations created by the context of the story frequently explain an event with little further effort. If there are no relevant expectations, the event's interest value, listed in the dictionary, is checked. If the event has little interest, processing moves to the next word. If the event has significant interest, the expectations listed in the word's dictionary entry are instantiated, with a pointer to the new event structure.

IPP keeps track of a story's main event. It checks to see if a new event is more interesting than its current main event. If an interesting event less interesting than the current main event is created, and it does not fulfill an expectation, then it is saved as an unexplained event, indicating IPP should look for an explanation.

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 (and only in these cases), then the words which modify the token are retrieved from short term memory. 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 be 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.

The other subdivision of TM's concerns the effects that TM has on subsequent processing. This

Sample Dictionary Entry (A1)

```
(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.
```

Sample Dictionary Entry (A2)

```
(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-$TERRORISM        ~ Set of expectations which specify
             FIND-$ROBBERY          ~   scripts we are likely to see.
             FIND-$KIDNAP           ~   If one is satisfied, the others
             FIND-$HIJACK]          ~   are deactivated.
```

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. For example, 'gunman' activates requests looking for feasible scripts.

TM's that are not interesting, and hence do not generate any expectations, can be placed into two classes, normal (NTM's) and empty (ETM's). NTM's can easily be associated with objects already in memory, even though they are not interesting. Examples of NTM's are 'airport', 'Vermont', and 'officials'. The tokens built by NTM'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 'someone' 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, and it is not interesting by itself, it is ignored in our parsing scheme, since there is little reason to remember it.

B - *Words that are saved and skipped*

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

The class of save and skippable 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 save and skip 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 retrieved and processed further, in a manner dependent on the TR type. (If the TM proves to be uninteresting, no further processing will be done.)

The class of 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 saved and skipped.

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 cannot be modified by TR Modifiers.

One aspect of processing which is common to all types of TR's is that their dictionary entries can indicate they should make the token they modify more interesting. So, "Arabic gunman," is more interesting than 'gunman', due to the inherent interest of the TR 'Arabic'.

Notice that "save and skip" processing would make it very easy to handle TR's 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 (B1)

```
(WORD-DEF ARABIC
  TYPE      TR
  SUBCLASS  STR
  INTEREST  2
  MEMORY    T
  DEF       (NATIONALITY . 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 out 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 dependent. ER's divide into standard ER's (SER's) and ER modifiers (ERM's) in a manner similar to STR's and TRM's. Processing is similar to that for TR's, except it occurs when an event is created, and ER's are looked for following the event, as well as those which have been saved in STM.

Sample Dictionary Entry (B2)

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

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 Token Refiners go along with which Token Makers. When read, they are saved with the TR's. 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.

Auxiliary 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 important uses of auxiliary verbs is the use of forms of 'to be' to make a verb passive. When an event is created by a past participle, IPP checks for such an auxiliary, and if one is present, modifies low-level processing appropriately.

Sample Dictionary Entry (B3)

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

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 - temporal and causal. Words such as 'before', 'while', and 'after' indicate temporal relations between events, and 'because', 'since', and 'therefore' indicate causal relations.

Sample Dictionary Entry (B4)

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

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 likely that few, if any, words are skippable in all domains, for all readers, whatever level they are processing. But for a given reader, working in a given domain, many words are skippable.

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. Expectations for such words are created which neutralize their inherent interest.

6. Examples of IPP

The first three examples shown here are computer runs of IPP on three stories taken from the New York *Times*.

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

*(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 **
SCRIPT    $TERRORISM
ACTOR     ARAB GUNMAN
PLACE     IRAQI EMBASSY
CITY      PARIS
TIME      MORNING
SCENES
    SCRIPT    $HOLD-HOSTAGES
    ACTOR     ARAB GUNMAN
    PLACE     IRAQI EMBASSY
    SCRIPT    $CAPTURE
    ACTOR     POLICEMEN
    OBJECT    ARAB GUNMAN
    PLACE     IRAQI EMBASSY
    AFTER     $HOLD-HOSTAGES SCENE

** UNEXPECTED EVENTS **
SCRIPT    $SHOOT
ACTOR     IRAQI OFFICIALS
OBJECT    ARAB GUNMAN
AFTER     $CAPTURE SCENE
RESULT
  STATE     DEAD
  ACTOR     ARAB GUNMAN

*(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 **
SCRIPT    $HIJACK
ACTOR     GUNMAN
FROM      BRONX
TO        AIRPORT
CARRYING  PASSENGERS
VEHICLE   BUS
SCENES
    SCRIPT    $KILL
    ACTOR     GUNMAN
    VICTIM    HOSTAGES
    SCRIPT    $CAPTURE
    OBJECT    GUNMAN
    ACTOR     POLICE
    TIME      NIGHT

** UNEXPECTED EVENTS **
NONE

*(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 **
SCRIPT    $DEMONSTRATE
OBJECT    AMNESTY-INTERNATIONAL
DEMANDS   IMPROVED JAIL CONDITIONS FOR
               WEST-GERMAN TERRORISTS
METHOD
    SCRIPT    $OCCUPY

** UNEXPECTED EVENTS **
NONE

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

The next two examples, the first from the Boston *Globe* and the second from the New York *Times*, illustrate how stories with similar content are processed similarly by IPP, despite differences in syntax.

```
 Yale TOPS-20 Command processor 3A(415)
@DO IPP

*(PARSE S4)

Input:

(IRISH REPUBLICAN ARMY GUERRILLAS
AMBUSHED A MILITARY PATROL IN WEST
BELFAST YESTERDAY KILLING ONE BRITISH
SOLDIER AND BADLY WOUNDING ANOTHER ARMY
HEADQUARTERS REPORTED)

Output:

** MAIN EVENT **
SCRIPT    $TERRORISM
ACTOR     IRA GUERRILLAS
PLACE     BELFAST
TIME      YESTERDAY
SCENES
    SCRIPT    $KILL
    ACTOR     IRA GUERRILLAS
    VICTIM    1 ENGLISH SOLDIER
    PLACE     BELFAST
    SCRIPT    $WOUND
    ACTOR     IRA GUERRILLAS
    PLACE     BELFAST

** UNEXPECTED EVENTS **
NONE

*(PARSE S5)

Input:

(GUNMEN BELIEVED TO BE BASQUE GUERRILLAS
TODAY SHOT AND SERIOUSLY WOUNDED A
PROVINCIAL SECRETARY OF THE RIGHT-WING
POPULAR ALLIANCE PARTY POLICE SOURCES
SAID)

Output:

** MAIN EVENT **
SCRIPT    $TERRORISM
ACTOR     BASQUE GUERRILLAS
TIME      TODAY
SCENES
    SCRIPT    $SHOOT
    VICTIM    SECRETARY
    ACTOR     BASQUE GUERRILLAS
    SCRIPT    $WOUND
    ACTOR     BASQUE GUERRILLAS
    VICTIM    SECRETARY
    EXTENT    GREATERTHAN-*NORM*

** UNEXPECTED EVENTS **
NONE

[PHOTO: terminated Tue 5-Jun-79 1:08PM]
```

Notice that in S5, the primary designation of the actor is given in the participial phrase, "believed to be Basque guerrillas," while in S4 "Irish Republican Army Guerrillas" is simply the subject of the sentence. IPP identifies the actors in the same way, just as people would normally do. (I.e. the qualifier "believed to be" is normally ignored.) Also notice that the events described in S4 by "killing" and "wounding" are no more difficult for IPP to understand than those described by "shot" and "wounded" in S5. In fact, the processing is virtually identical.

The final two examples illustrate how IPP can jump to a conclusion about the representation of a story and then drop that representation when it finds a more interesting possibility. The first example, S6 is the initial fragment of the full sentence processed in the second example, S7, which is from the New York *Times*.

```
 Yale TOPS-20 Command processor 3A(415)
@DO IPP

*(PARSE S6)

Input:

(A YOUNG JAPANESE GUNMAN BROKE INTO A
TOKYO BANK TODAY AND KILLED TWO
POLICEMEN)

Output:

** MAIN EVENT **
SCRIPT    $ROBBERY
ACTOR     JAPANESE GUNMAN
PLACE     TOKYO BANK
```

```
TIME        TODAY
SCENES
    SCRIPT      $KILL
    ACTOR       JAPANESE GUNMAN
    VICTIM      2 POLICEMEN
    PLACE       TOKYO BANK

** UNEXPECTED EVENTS **
NONE

*(PARSE S7)

Input:

(A YOUNG JAPANESE GUNMAN BROKE INTO A
TOKYO BANK TODAY KILLED TWO POLICEMEN
HELD 36 PERSONS HOSTAGE AND VOWED HE
WOULD NOT LEAVE UNLESS HE RECEIVED
$250000)

Output:

** MAIN EVENT **
SCRIPT      $TERRORISM
ACTOR       JAPANESE GUNMAN
DEMANDS     $250000
PLACE       TOKYO BANK
SCENES
    SCRIPT      $KILL
    ACTOR       JAPANESE GUNMAN
    VICTIM      2 POLICEMEN
    PLACE       TOKYO BANK

    SCRIPT      $HOLD-HOSTAGES
    HOSTAGES    36 PERSONS
    ACTOR       JAPANESE GUNMAN

** UNEXPECTED EVENTS **
NONE

[PHOTO: terminated Tue 5-Jun-79 5:20PM]
```

Initially IPP assumed the story to be an instance of the ROBBERY script, since 'bank' triggered a prediction from 'gunman'. The ROBBERY script has completely disappeared in the final representation of S7. Once IPP discovered the HOLD HOSTAGES script, from which it inferred TERRORISM, it decided that was much more interesting than ROBBERY, and selected it as its representation. However, it did reincorporate the KILL scene into the new representation.

7. 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 either words themselves or syntactic notions are ever sensible stopping points. 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 our parser, Conceptual Dependency is used as a kind of internal language used in situations where the final representation is not apparent. Its

use can allow conceptually based inferences to be made. 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 their own cumbersomeness. Human-like 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 speed and lack of fragility. 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

CMU Computer Science Speech Group (1976). Working Papers in Speech Recognition - IV - The Hearsay II System.

Carbonell, J. G. Jr. (1979). Subjective understanding: Computer models of belief systems. Research Report 150, Department of Computer Science, Yale University.

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 Francisco.

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

Marcus, M. (1979). *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, Massachusetts.

Marslen-Wilson,W. D. (1975). Sentence Perception as an Interactive Parallel Process. *Science*. Vol. 189, pps. 226-228.

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

Plath, W. J. (1974). Transformational Grammar and Transformational Parsing in the REQUEST System, *Computational and Mathematical Linguistics*, in A. Zampolli ed., Proc. of the Int. Cong. of Computational Linguistics, Pisa, 1973, Casa Editrice Olschki, Firenze.

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.

Wilensky, R. (1978). Understanding Goal-Based Stories. Research Report 140, Department of Computer Science, Yale University.

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.