WORKING PAPER 75

WAIT-AND-SEE STRATEGIES FOR PARSING NATURAL LANGUAGE

Mitchell Marcus

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

August 1974

Abstract

The intent of this paper is to convey one idea central to the structure of a natural language parser currently under development, the notion of wait-and-see strategies. This notion will hopefully allow the recognition of the structure of natural language input by a process that is deterministic and "backupless", that can have strong expectations but still be immediately responsive to the actual structure of the input. The notion is also discussed as a paradigm for recognition processes in general.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defence and monitored by the Office of Naval Research under Contract number N00014-70-A-0362-0005.

WAIT-AND-SEE STRATEGIES FOR PARSING NATURAL LANGUAGE

Mitchell Marcus

During the past year, I have been developing a natural language parser that differs from existing parsers in what I believe are significant and theoretically interesting ways. The intent of this paper is to try to convey the one central idea behind this research, the notion of wait-and-see strategies. This notion, I believe, allows the recognition of the structure of natural language input by a process that is deterministic and "backupless", that can have strong expectations but still be immediately responsive to the actual structure of the input. Much of this paper is also intended to argue that one does, in fact, want to build a parser with these characteristics.

It should be noted at the outset, though the theme will not be explicitly developed until the end of this paper, that the notion of a wait-and-see strategy is not specifically a notion about how to parse, but rather a more general notion of how to recognize. It might be of interest to the reader while reading this paper to keep in mind the question of to what extent this notion is useful for other sorts of recognition. The initial inspiration for this work was the thought that the notion of local features constraining the surrounding environment, developed in the vision work of Dave Waltz <Waltz '72>, might be combined with the essentially top-down parsing strategies developed in the work of Terry Winograd <Winograd '72> and Bill Woods <Woods '72>. It would be pleasing if this wait-and-see notion, developed here for language, had application in other areas of A.I. as well.

Since this paper does focus on one issue, it will not discuss many other aspects of the parsing system that deserve at least passing mention. A language called PARSIFAL for writing grammars within the wait-and-see framework is being implemented, modelled on Winograd's PROGRAMMAR. A case frame system is being developed along with a notion of where the use of case frames fits into a natural language understanding system. PARSIFAL is being used to write a grammar which currently handles only very simple sentences, but which is soon to be much extended. This paper is intended as a progress report; the notions that it describes will hopefully come to fruition in the coming months. Finally, it must be admitted, the discussion in this paper has been greatly oversimplified at several points to avoid wandering into any of the above areas.

GUESS-AND-THEN-BACKUP VERSUS WAIT-AND-SEE STRATEGIES

To get an initial grasp on the notion of wait-and-see strategies, it seems appropriate to compare this notion with the current prevailing strategy, which can be termed guess-and-then-backup, embodied in its purest form in Woods' Augmented Transition Network parser <Woods '72>, but which is also central to the parser in Winograd's SHRDLU <Winograd '72>. At the heart of the parser for William Woods' LUNAR system, which implements a version of his Augmented Transition Network formalism, various options are enumerated in the code for the next possible constituent at a given point in the parse and these options are tested one at a time against the input string. The parser assumes tentatively that one of these options is correct and then procedes with this option until either the parse is completed and the parser is done or the option fails, at which point the parser simply backs up and tries the next option enumerated in the code. A parser based on the ATN formalism abstractly has the power to use its knowledge of why the failure occurred to reorder the remaining options at that point, or else to choose an alternative option pending elsewhere, but this ability is not extensively used, to the best of my knowledge. This is exactly the strategy of guess-and-then-backup: enumerate all options, pick one, and then (if it fails) backup and pick another. Much (though not all) of the parser for Terry Winograd's SHRDLU is also structured along guess-and-then-backup lines, although some of this structure uses the potential of PROGRAMMAR to be quite bright about correction of false parse paths. But even here, the structure is one of guess-and-then-backup(-cleverly).

A Wait-and-See Parser (WASP) on the other hand, rather than immediately picking a possible option and attempting to push it through, backing up and picking another option if necessary, prefers to wait and see exactly what it should do at any point in the parsing process by doing limited lookahead of a particular sort. Attempting to eliminate blind search through the space of possible grammatical structures, a Wait-and-See Parser is expected to know, for each point in the parse at which several different options are possible, a small set of easily answered questions which determine exactly which options will succeed. At each step of the parse, the WASP first calls on low level "middle-down" processes to parse the essentially finite-state substructures of English, verb clusters, noun groups up to the head noun, etc., from a little ahead in the input. The parser is then expected to determine exactly what it should do next by inquiring about some small number of properties of the structure it has already parsed and the few substructures lying before it. Once it decides on a course of action, it is expected to proceed without error. (One allowed course of action, it should be noted, is to conglomerate two or more pieces of unassimilated substructure and then look still farther ahead into the input.) Thus, the parser can allow some limited number of unincorporated substructures to pile up before use, but it can never change a decision once made.

WHY ANOTHER PARSER?

One question immediately springs to mind: Why bother to build yet another parser? Isn't the existing theoretical framework for parsing sufficient for the time being when there seem to be more pressing areas in natural language research? There are a number of reasons for building such a parser and attempting backupless wait-and-see parsing, the most important of which can be divided into two categories, the practical and the theoretical. To take the two in order:

(a) I believe a parser of this type will make much simpler adequate handling of several important phenomena of natural language that are either handled only in special cases or are handled by ad-hoc mechanisms (or both) in existing parsers. These phenomena include ellipsis, the ability to parse sentences that are not strictly grammatical, and the ability to make some sense out of utterances that cannot be fully parsed or understood. More will be said about these phenomena below, after the structure of the parser has been explained in more detail. (I believe it will also make much more accessible the problem of parsing conjunction and other coordinate structure, but will say nothing more about conjunction in this paper.)

(b) I believe that people don't in fact do unconscious backup in the process of understanding language, and that natural language is 'designed' to allow backupless language understanding. (I exclude here the understanding of 'garden path' sentences, which do require backup, but also are understood in a mode quite different from normal language.) The central hypothesis is that natural language provides clues in quite local contexts to allow the deterministic decision of what to do next. Writing a grammar for the WASP involves finding out exactly what these clues are and what form they take, and should therefore shed a new sort of light on the structure of language.

Consider the following sentences:

(1) I told the boy the dog bit Sue would help him.

(2) When Red Moon saw the pony he was to choose from his face flowed many tears.

(3) In the book the girl took the basket had magical powers.

(The last two sentences are admittedly underpunctuated; with proper punctuation added they read:

(2a) When Red Moon saw the pony he was to choose, from his face flowed many tears.

(3a) In the book the girl took, the basket had magical powers.

Sentence (1) becomes clearer with the following change:

(1a) I told the boy the dog bit that Sue would help him.

If you had no trouble getting these interpretations of the sentences the first time through, be informed that you are in a very small minority.) Most people are led down the garden path by these sentences, parse them wrong and then boggle. But if backup were done at an unconscious level, none of these sentences should cause problems. Note that in (2) the misparse is limited to either incorrectly assigning the prep 'from' to the verb 'choose' (the pony he was to choose from) or assigning the entire prepositional phrase 'from his face' to 'choose' (the pony he was to choose from his face). Either error is locally correctable. It would seem that if there were an unconscious backup mechanism such errors should be handled by such a mechanism. (In passing, also note that both misparses of (2) are semantically absurd -the only thing one might be able to choose from a pony is a steak - yet people buy the misparses anyway. Semantics, or at least very deep semantics, does not dominate the parsing process at some fairly high level.) An even stronger argument can be made from (1). Here the garden path is at worst four words long, but if there were an unconscious backup mechanism, it must certainly be able to go back more than four words to find a branch point without overloading. For example, for a guess-and-then-backup parser to parse the pair of sentences

(4) Is the block sitting in the box?

(5) Is the block sitting in the box red?

it must do a backup of at least four words on one or the other of them. If people were guess-and-then-backup parsers, and if they can get (4) and (5) without any trouble, then they most certainly should not find (1) to be the fairly confusing garden path it is perceived to be.

A MINI-THEORY OF SYNTACTIC ENCODING

If people don't do backup when they parse, how do they avoid it? How can language be parsed deterministically? I believe, as mentioned above, that natural language is specifically designed (as the result, as Minsky puts it, of a million year long research project) to allow deterministic parsing. I believe that language provides distinct signposts as to the actual structure of an utterance, if viewed properly. To make this notion both a little more definite and a little more plausible, it is necessary to present a brief mini-theory of syntax, a good mini-theory in that it is highly oversimplified, but still quite suggestive.

I believe that syntax should be viewed as nothing more that a coding scheme to allow semantic messages to cross a linear, slow, potentially noisy acoustic channel. Furthermore, this encoding should be such that the message is easily recoverable, that the code should allow efficient decoding wherever compatible with the constraints of the channel. It is therefore reasonable to hypothesize that coding is done in as local an environment and as low a level as possible, and that decoding clues are given on as local a level of structure as can be achieved. The obvious example that comes to mind: The passive marker is encoded in the local environment of the verb group and can be entirely decoded within the same limited context.

One important efficiency trick is the local isolatability of group structures - NGs, VGs, and the like. (In this paper, the term NG, noun group, refers to the portion of a noun group up to and including the head noun; the term VG, verb group, refers to the main verb and its immediately preceding auxiliaries.) Strong local clues, at the beginning of groups especially, allow parsing without any reference to global structure. NGs provide the obvious example here, especially in text: The structure of NGs is such that it is usually clear where a NG begins; "the", "a", and other articles are dead tip-offs, as are ordinals, most adjectives, etc. One need not wait until much of a NG is spoken (or, more accurately in this context, typed in) before the structure of the group can be begun to be built - almost assuredly without error.

Another efficiency trick in the syntax coding scheme is that of an 'unmarked' default order of constituents vs. 'marked' orders. In the speaking of semantic messages containing an actor and some affected object, for example, the unmarked order is actor - verb - object. When the expected order is varied such deviance is often marked at a quite local level by the addition of some sort of marker. The passive is again a good example here; the order object - verb (- actor) is marked by the passive (be - en) locally decodable in the VG. If one wishes, this can be thought of as a "transformation" (in a very loose sense) from the expected order; a transformation that contains two parts - one a shift from the normal order and another a marker of the change. The process of generation involves deciding to use the "shifted" word order, the process of interpretation involves decoding the marker so that interpretation may be "shifted" as well. Often, of course, the marked order of constituents is quite distinct from unmarked "normal" order so that the order itself marks which shift has been chosen by the speaker; in this case no extra marker is necessary. The most obvious example here is the marking of yes/no questions by the "inversion" of the subject of the question and the first element of the verb group of the sentence. Thus the yes/no question corresponding to the declarative

(i) Herbert has eaten all the cookies.

is marked by the inversion of the auxiliary "has" before the subject "Herbert":

(ii) Has Herbert eaten all the cookies?

If one considers clause structure built up from the input string in successive levels, it is reasonable to expect that after each level is constructed, markers then local to that level - i.e. determined by given strings of elements on that level - will be picked up. I believe that as each level is examined, first the level of a string of individual words, then the level of a string of individual groups, different markers are locally decodable. I believe, for example, that the "descent" boundaries between embedded clauses and higher clauses (as well as often being marked by binders like 'who', 'that', etc.) are marked at the level of the group string. To give a very trivial example, the string NG NG VG, e.g. "the boy the girl kissed", almost always marks the structure NG ! NG VG (where ! indicates the beginning of an embedded clause) unless a pause of some length would exist between the two NGs in speech, a situation usually indicated in written text with a comma. It is my hypothesis that such indicators are given in a local enough environment to allow a parser to do, at each level, unerring parsing of the structures at the next level.

THE GENERAL FRAMEWORK OF A WASP

It is now appropriate to attempt the dissection of a WASP, to lay the groundwork for discovering exactly how its inner workings make possible the realization of wait-and-see strategies. In this section, and those below the discussion will focus on the structure of the WASP that is currently being implemented.

The WASP consists of two layers of procedures: the lower layer a battery of group level parsers and the upper a battery of clause constructors. The group level parsers build noun groups (more exactly NGs up to the head noun), verb groups (i.e. the verb cluster up to the main verb), and the like, while the clause constructors build these substructures into clauses. The group level (GL) parsers all look at the input string, each free to build a group level constituent at any time another GL parser isn't chomping on the input. When a GLP considers itself finished, it dumps the constituent it has built onto a hook from which the clause constructors can take the constituent when it can be glued into some larger structure. The clause constructors are semi-syntactic, semi-semantic beasts in that they attempt to build both an annotated surface structure parse and a semantically oriented structure around each verb by 'decoding' the syntax of the clause around that verb. When the clause constructors find a verb, they seize a Fillmore-like case frame associated with the verb and attempt to fill in the slots. The clause constructors needn't immediately use each group when it is placed on a hook, but rather can let some fixed small number of group level constituents "pile-up" on the hooks of the group level buffer before deciding how to glue the first of the constituents into a larger constituent. In this sense the clause constructors have a limited and sharply constrained look-ahead ability. The hypothesis central to the wait-and-see strategy is that this amount of look-ahead is sufficient to 'wait-and-see' what the correct parse should be, so that no backup is necessary.

One more point should be made about the notion of 'hooks'. Any constituent whose function at a higher level isn't clear is placed on a hook until it is seized by a higher constituent. That larger constituent itself might then find itself on a hook until it too can be placed into a still larger constituent. The point is that a hook can hold the top node of any unincorporated constituent; whether the structure underneath is "the" or "the big green cookie monster's toe that got stubbed" is immaterial. Lest this leave an impression that the WASP is basically bottom-up, it should be added that the clause constructors will quite willingly start a higher level constituent as soon as they are confronted with solid evidence that such a constituent is on the way, and will then pursue additional constituents for the structure in a fairly top-down manner.

This, in outline, is the general framework of the WASP.

FILLING OUT THE FRAMEWORK

It is quite pleasant that one mechanism seems to be appropriate for the realization of both the group level parsers and clause constructors discussed above; the parsers at group level, it turns out, are very naturally realized as a degenerate form of the clause constructors at the higher level. Both are realized in the WASP as a kind of pattern-invoked demon which will be called a module. Each module consists of a pattern, a pretest procedure, and a procedure to be executed if the pattern matches and the pretest succeeds. (A pattern consists of a list of sets of features, e.g. "((NG DEFINITE) (VG INFINITIVE))". For a pattern to match, each feature set of the pattern must be subsumed by the set of features on the corresponding structure in the relevant string of syntactic units - words at one level, groups at the other.) At the group level, the patterns tend to consist of single features on single words with pretests that act as 'no-ops'; at the clause level, the patterns and pretests are much more complex. The point to be made about these patterns is that they are very simple tests of very local syntactic properties. While the pretests can be arbitrarily complex, it seems to be the case that in most instances the local properties reflected in patterns suffice by themselves to get the right module to the right place at the right time.

Since the organization and action of group level modules tends to be much simpler than that of clause level processes, nothing more will be said about group level processes below. Let us add only that the group level processes conceptually serve to conglomerate very locally determined simple structures, so that processes with a more global interest will not be overwhelmed by irrelevant detail.

At the clause level, modules are organized into packets, which can be activated and deactivated to reflect the parser's global expectations about syntactic constructs which may be encountered at any particular point in the parser's analysis. (This notion of packet derives from the work of Scott Fahlman <Fahlman '73>.) Some typical packets within a grammar of English might be: MAJOR-CLAUSE-START, a packet of modules which will parse the initial segments of major clauses up to and including the verb; MAJOR-CLAUSE-AFTER-VERB (M-C-A-V), a packet which parses objects, prepositional phrases, and the like; EMBEDDED-CLAUSE-AFTER-VERB, a packet similar to M-C-A-V, except that its modules are responsible for deciding exactly what role the relative head plays (if there is one), and when the embedded clause is complete and further groups belong to higher clauses; etc. Packets of modules thus roughly correspond to states in an Augmented Transition Network parser, with some major differences: not only do packets correspond to much larger grammatical "chunks" than typical ATN states, but more importantly, modules in a given packet may interact with each other quite strongly. For example, several modules in a packet might modify structures in the GL buffer so that a more general module will then apply to the resulting common structure; in the packet MAJOR-CLAUSE-START, for instance, the initial structure of yes/no questions, wh- questions, declaratives, etc. might be parsed by an almost reverse-transformational mechanism. In such a grammar, the module which picks up imperatives (with pattern "(VG INFINITIVE) (NG)") might insert a dummy "You" NG into the GLB after marking the clause IMPERATIVE, the module which parses yes/no questions might mark the clause type and then rearrange the GLB to reflect the declarative "underlying" "subject - verb" pattern, etc., all so that a common "subject - verb" parsing module with pattern "(NG) (VG)" can work on all three of these sentence types. It should also be noted that more than one packet may be active at any given point in the parse; for example, if a sentence or a single PG might be expected as an answer to a question (e.g. "Where did John go?"), packets for each can be initially active.

It is important that a recognition process have expectations that impose some structure on the search space that it expects to examine; it is also important that such a process not be blinded or enslaved by these expectations. By overlaying a pool of pattern-invoked demons, i.e. high-level interrupts, with this packet structure - a means for enabling and disabling blocks of these interrupts - we obtain a mechanism which can have expectations but still remain interrupt driven and thus responsive to its input. Higher resolution of levels of expectation can be achieved by assigning priorities to these interrupts, as will be discussed in more detail in the following section. For now, let us just note that this priority mechanism allows the WASP to have some slight expectation of infrequent but possible sorts of grammatical events; modules to parse single NG or PG utterences can always be lurking far in the depths, ready to snap if a fragment drifts down to them without being grabbed by some higher module, i.e. if no higher-prioritied interrupt occurs.

Within a packet, each individual module is a specialist for the particular syntactic situation its pattern and pretest define. Since modules are not only responsible for building structures, but for building the right structures, each must be an expert at the differential diagnosis of all the possible global situations which are consistent with the local environments in which it may be invoked. Thus when invoked, an efficient module whose conditions of invocation are consistent with more than one large scale structure must procede to ask just the right questions of the world to resolve the question of what global environment it is in, and thus what actions it should take. If necessary, and if the group level buffer isn't full, it can ask for group level structures further on in the input to be generated, but within these limits it must resolve the situation on the basis of knowledge at hand. (There are exceptions to this, but they are beyond the scope of this discussion.) This ability of a module to determine the relevant properties of the global context in which it becomes active by testing a few specific properties of a limited number of groups is the key to building wait-and-see strategies, as an example or two in the next section will make clearer. It should not be overlooked, however, that this diagnosis process depends as much on the interrupt structuring scheme of packet, pattern, pretest, and priority that gets the right module to the right place at the right time as it does on the knowledge that the triggered module contains. Half the information needed by an activated module to solve the problem of what to do next is provided by the simple fact that the environment was such that it was triggered.

A module, though a narrow specialist, can use many different sorts of knowledge. It may use syntactic or semantic information to decide on one hypothesis as opposed to another, referring questions where necessary to the full-scale semantic inference mechanism (to be simulated by me for now) for which it hypothetically serves as front-end. Though little will be said in this paper about the sorts of semantic-syntactic interactions that will be necessary to build a functioning natural language system with a wait-and-see parser as front end, one point should be made: Because of the differential diagnosis nature of its job, it has become clear that a module often needs to know - and must be able to ask the semantic component - not simply whether some constituent is semantically suited to serve in some capacity, but which constituent of several is better suited to serve in a given capacity - and by how much, based not only on simple case or semantic marker checks, but often on the surrounding context as well. This particular problem seems to raise many questions about the interaction of the individual components in a natural language system, for here the task of semantics is not merely filtering, but supplying more complex forms of information.

What if a module, within these limitations, makes a mistake in its decision? Then the sentence is a garden path for the parser, and it should fail gracefully. As the size of its grammar grows, one would hope that the parser will be confused by only those sentences which confuse people, and one might even hope that it will be confused by exactly those sentences which people find to be garden paths.

AN EXAMPLE OF A WAIT-AND-SEE STRATEGY

To make the discussion above a little more concrete, consider the problem presented to a wait-and-see parser by the following pair of sentences:

(6) Is the block sitting in the box?

(7) Is the block sitting in the box red?

The problem is quite simple: the parser must somehow realize that while the sentences differ in but one word, in (6) "sitting in the box" is part of the predicate of the main clause of the sentence, while in (7) it is the predicate of a (very) reduced relative clause modifying "block", i.e. the declarative sentences corresponding to (6) and (7) are

(6a) The block is sitting in the box.

(7a)The block (which is) sitting in the box is red.

A very informal answer to this is quite simple: simply see if it is true that the only groups after the "participial phrase" are adverbs or time phrases, if so the sentence has the general form of (6), otherwise it has the general form of (7). What the discussion below centers on is how this knowledge can be formalized and embedded within a wait-and-see parsing framework. At the point in the parse of either (6) or (7) when the parser has seen

(8) Is the block sitting in the box...

a module must become active which knows about some general form of this problem and embodies some sort of knowledge which resolves this dilemma. How can a module be formulated so that it does the right thing at the right time?

Before we can formulate a diagnostic module for this situation, we must know what substructures will be present when the module is intended to become active and what processes are necessary to create either of the structures implicit in (6) and (7). First of all, since the diagnosis of either sentence's real structure can be done only by looking past the end of the "participial predicate" beginning with "sitting", this predicate must exist as a coherent substructure before the module is triggered. We will consider this participial predicate to be a clause which is missing its subject and some of its auxiliaries, and build it as such, calling the temporary substructure - a clause without subject- a <u>verb phrase</u>. Since this verb phrase can be built upon one hook, it is a legitimate substructure by the wait-and-see rules. It will have the features CLAUSE and VP, and it will also be labelled with the most important features of its VG, namely ING and PARTICIPAL (indicating the ING form of a PARTICIPAL), since it will turn out to be useful for the outside world to see these features at a glance. (Exactly how this clause is constructed, and how the processes which construct it decide that it is complete must unfortunately be left to another paper. To do the topic justice would expand this paper by a factor of two.) Since parsing of groups is done locally, "is" will be parsed as a VG, but it will also have the feature AUX, which indicates that it might be the separated initial auxiliary of a larger VG, and the feature BE, which indicates that the initial word of the VG is a form of "be". (Note that the decision of what the "main verb" of a VG is must be delayed until clause-level processes are assured the entire VG has been found.) The NG "the block" will also be a substructure and will have features that indicate that it is a NG, that it is singular (NS), and that it has a definite determiner (DEF). Thus the substructures in the group level buffer after (8) has been partially digested will be (ignoring all but the top level structure):

(9) (VG AUX BE) is
    (NG NS DEF) the block
    (CLAUSE VG PART ING) sitting in the box.

Given this level of structure, processes must be specified that can take these substructures and somehow put them together to build larger substructures consistent with either the structure implicit in (6) or the structure implicit in (7), ignoring for now the problem of deciding which structure to build.

If these substructures in the GL buffer are to be interpreted as the initial parts of sentence (7), the VP must be completed as a reduced relative clause. To do this, we propose a module called PARTICIPIAL-MODIFIER (PART-MOD) with pattern "NG (VP PART)" which will (a) complete the VP as a CLAUSE by adding the NG as grammatical subject, (b) modify the tense of the VG by adding "present progressive" to the tense of the VG, and (c) hook the clause onto the NG as a rank shifted qualifier (or if you prefer, a relative clause) of the NG, adding the feature QUALified to the NG. Thus, after this module has been loosed on (9), the group level buffer will contain:

(VG AUX BE) is
(NG NS DEF QUAL) the block sitting in the box.

Since this sort of relative clause can occur anywhere, this module should live in the packet CLAUSE-POOL, which is the clause level packet of always active modules. Since a NG can't take relative clause qualifiers if it is a proper noun and thus has the feature NPR, the pretest for this module checks to make sure the NG does not have the feature NPR.

If the substructures of (9) in the GL buffer are to be interpreted as the initial parts of sentence (6), rather than of sentence (7), the VP becomes the foundation of the major clause of the input with the VG "is" as an auxiliary "shifted" to form a yes/no question. We propose a module YES/NO-QUESTION (Y/N-Q) to put the group level substructures of (9) together to form this structure. Y/N-Q has the pattern "(VG AUX) NG VP" and a pretest which checks that the first word of the VG of the VP is marked with the appropriate "suffix" feature for the initial AUX, i.e. the features ING or ED for an AUX of BE, EN for HAVE, INFINITIVE for MODAL, etc. When activated, this module (a) adds the initial VG to the VG of the VP, (b) puts the VP CLAUSE onto a special hook, labelled S, which is reserved for the root of what is to become the full parse tree (Thus the VP becomes the "major clause" of the sentence.), (c) adds the NG to the CLAUSE as its subject (which causes the right thing to happen to the case frame representation of the sentence which is being built in parallel with the annotated surface structure parse), and (d) notes that the CLAUSE is a yes/no question by adding the features YES/NO and QUEST and removing the features VP, PART and ING from the CLAUSE. After Y/N-Q has been run, nothing is left in the group level buffer, but S hooks the structure

(CLAUSE QUEST YES/NO) is the block sitting in the box...
    (NG NS DEF SUBJ) the block
    (VG PRES V3PS) is sitting
    (PG MOBJ) in the box.

(Where V3PS means the verb is 3rd person singular; MOBJ means that the PG is a "marked object" of the verb, and SUBJ means the NG is the subject of the CLAUSE.) Much of the lower level structure was constructed when the VP was constructed, for now it is fair to consider it formed by magic. Since this module should be active only at the beginning of clauses, it lives in packet CLAUSE-INITIAL, which is activated only when the parser expects the beginning of a major clause or something which looks like a major clause.

The dilemma presented to the WASP by (6) and (7) above can now be restated quite precisely in terms of the two modules PART-MOD and Y/N-Q. At a time when both these modules' packets are active, and thus when both these modules can become active, the group level buffer will contain the structures (repeating (9) above):

(VG AUX BE) is
(NG NS DEF) the block
(CLAUSE VP PART ING) sitting in the box.

The problem now is simply that both modules are applicable in this situation, since their patterns are:

PART-MOD - (NG (VP PART))
Y/N-Q - ((VG AUX) NG VP)

What we need is a module which can diagnose which sort of structure should be built, i.e. which of these two modules should become active, and we also need some way to force this diagnostic module to become active before either of the others.

If modules were "pure" demons, as would be true, for example, if these modules were a simple variation of PLANNER theorems <Hewitt '72>, any applicable module in any active packet might jump first in any given situation, since there is no way conceptually to order relevant modules without some super-module giving suggestions. Thus even if there were a diagnostic module, there would be no way to assure that it would become active. In the WASP, however, modules are ordered by priority. Every module in the WASP has associated with it a positive integer which is its priority, with smaller integers carrying higher priority than larger integers. (It should be noted that since any number of modules can have the same priority, this ordering need only be partial.) In any situation, only one module, the module of highest priority of those applicable, ever becomes active unless the active module explicitly calls another module by name and tells it to run. Thus the module of highest priority of those applicable in any situation bears the onus of diagnosing the correct path for the parser to follow and of assuring that only correct structures are built. It must be noted that this control structure is very similar to the production systems of Newell and Simon <Newell & Simon '72>, but we will not dwell on this fact here.

Now that we have at hand all the tools necessary to formulate a module which can diagnose between structures like that of sentences (6) and (7), designing the module itself is almost trivial. First of all, the diagnostic module's pattern should be the intersection of the patterns of PART-MOD and Y/N-Q and should be in a packet which is active only when both of the two conflicting modules are active. Thus, the pattern of our module PARTICIPIAL-DIAGNOSTICIAN (PART-DIAGN) will be "(VG AUX) NG (VP PART)" and it will be in packet CLAUSE-INITIAL. It will have the same pretest as Y/N-Q, namely that the first word of the VG of the VP has a "suffix" feature appropriate for the initial alleged AUX. We make this the pretest, as opposed to the pretest for PART-MOD, rather arbitrarily, since once either pretest succeeds, this module knows it assuredly can diagnose the correct action for the parser to take. Thus, the module's first action is to test whether the pretest of PART-MOD succeeds, if not, i.e. if the NG has the feature NPR, then it knows PART-MOD does not apply, and tells Y/N-Q to run. (Modules have the ability to send other modules any mail that they choose, though it is necessary that the receiver know how to read his mail.) If PART-MOD is applicable, the diagnosis must be made. This can be done in very simple grammars (be prepared for a great anti-climax) simply by seeing if the next group after the VP has the feature QPUNC, i.e. by seeing that the next group is a question mark. In more complex grammars, we need only check to see whether or not all the groups that fall before the question mark are marked with either the feature ADVERB ("for sure", "like you told me", ...) or the feature TIME ("this year", "on Tuesday", if the tense of the initial BE is PAST: "when the pyramid was picked up", ...). (The very interesting question of whether to parse the end of

Was the block sitting in the box last Tuesday...

as "(sitting in the box last Tuesday)" or "(sitting in the box) (last Tuesday)" is decided by the process that builds the participial phrase and is outside the scope of this discussion.) If the only groups before the question mark are ADVERBS or TIME phrases, the module tells Y/N-Q to run, otherwise it activates PART-MOD. This entirely diagnoses the situation, at least for moderately simple grammars, and PART-DIAGN is finished. To summarize, the module (a) checks whether the NG has the feature NPR, if so it tells Y/N-Q to run. If not, it (b) checks whether or not it finds a group marked QPUNC before it finds a group marked with neither the feature TIME nor the feature ADVERB, if so it tells Y/N-Q to run, otherwise it tells PART-MOD to run.

Two facets, in particular, of this module deserve mention, as they are very illustrative of other diagnostic modules. The first is the simplicity of the diagnostic procedure; the second is the general structure of the module. It seems to be the case that many similar sorts of locally ambiguous syntactic situations can be resolved by diagnostic modules with complexities of the same order of magnitude as PART-DIAGN, with a substantial portion of the questions asked requiring nothing more than simple feature checks. Some situations do need an elaborate structure of such questions to diagnose all possible outcomes, but the actual number of questions which need to be answered for any instance of such a situation is still usually less than five or six. (This sort of complexity is needed to resolve the termination of embedded clauses; this is one reason why this topic requires a paper of its own.) There are two reasons for this seeming disparity. The first is that, as mentioned above, the "intelligence" of the parser depends as much upon the interrupt control structure that gets the right module to the right place at the right time, as it does upon the knowledge that given modules contain. The second reason - and the second facet of the module deserving mention - is that the general structure of diagnostic modules, like the module developed above, is conceptually that of a decision tree. Even in complex situations, it seems to be the case that decision trees are a rich enough structure for doing the diagnosis and that these trees tend to be shallow, even if quite bushy. Given this structure, of course, only a single path through the tree need be followed in any given situation, so that the number of questions that need be answered is proportional to only the height, and not the breadth, of the tree.

A RETURN TO ELLIPSIS AND UNGRAMMATICALLITY

It was suggested above that the WASP is better equipped than guess-and-then-backup parsers to handle phenomena like ellipsis, the ability to parse sentences that are not strictly grammatical, and the ability to make sense out of input that cannot be fully parsed. Let us now compare some approaches to these problems within both frameworks.

It would seem that there are two plausible methods for adding to parsers like SHRDLU's and the LUNAR system's the capability to handle sentences that are to some degree ungrammatical. The first is to exhaust all grammatical parses and then effectively start over again, loosening up various grammaticality constraints. The second is to originally throw away those constraints which can usually be violated and still leave sentences understandable to people, making less use of the syntactic guides still present in fully grammatical sentences. It would seem that option (2) is not very good, and that option (1) implies that all the enumerated options must be exhaustively attempted and fail before ungrammatical sentence handling can begin. Ellipsis handling is much the same sort of problem. A guess-and-then-back-up parser, it would seem, either needs to enumerate all possible ellision options in its code and then guess them or else must bomb out and only then go into a special ellipsis handling mode. Even more difficult is trying to pull something useful out of input which the parser is unable to parse. Given such a sentence, a guess-and-then-backup parser exhaustively goes through its search space and then throws up its hands knowing - and able to tell the user - only that it has failed. Not only is this distressing to the user, but it also fails to be a very good model of how people behave.

These difficulties all arise directly from the basic nature of a guess-and-then-backup parser and hence seem to be very difficult to overcome within that framework. The problem is quite fundamental: Such a parser chooses to follow a given path in its search space for no other reason than the fact that that path is the next untried option in its code, hence it must assume that any given path is quite likely to be wrong and thus that it should constantly be on the lookout for signs that it has made a wrong turn. When such a parser runs into difficulty on any given path, it is almost always correct for it to assume that it should abandon that path and try another; furthermore, there seems to be no good way for it to spot the cases when that assumption is false. Thus, if such a parser is given a sentence that doesn't exactly conform to any given path through the parser, it will attempt and abandon every path through its search space, and finish with no more information than the knowledge that the sentence "does not parse".

One great advantage that the WASP has over such parsers is that it always believes that the path it is on is the only feasible path (ignoring for now the problem of ambiguity). Thus even in the worst case, given a sentence (or non-sentence) it cannot parse, when the parse becomes blocked, it still has some structure with which to say "here is as far as I got, what is the difficulty?". In fact, it would seem that the parser can often do much better, simply building the largest pieces it can out of remaining input, and then either handing the pieces to higher level processes to be handled in some sort of problem-solving mode, or else giving the pieces to the user and asking for some glue (if the parser isn't completely astray). For example, assume the parser takes the garden path on

I saw the grand canyon flying to New York.

It will first attempt to use "flying to New York" as a modifier of "the grand canyon", reject the modification on semantic grounds, (laugh(?),) and then end up with two fragments, the first a clause "I saw the grand canyon", the second an "ing"-complement "flying to New York". If there is a special higher level syntactic problem-solver that knows about dangling modifiers, perhaps an error recovery can be made. Otherwise, the user can be told "I know you saw the grand canyon, but what's this about flying to New York?".

Handling ellision within the WASP framework is facilitated by the fact that the WASP is never so blinded by its expectations that it cannot immediately realize the real form of the input. Modules with very low priority can always be active in the parser to handle ellided structures or the sorts of sentence fragments that constantly occur in conversation. The parser can have expectations of a full sentence in that the high priority modules are those that put together substructures to build a sentence, but low priority modules can lurk in the background ready to spring if their patterns match when no higher priority module applies. It is also quite easy within this framework for given sorts of fragments to be expected (i.e. the corresponding modules or packets of modules to be made active) following some sort of comment in conversation. After a curious natural language understanding system asks the user "What are blocks good for?", it can activate a set of modules to expect the sorts of fragments likely in answer to such a question. The response "To build arches with." will be caught by a module with pattern "(VP INF) (PREP) ()" that will add the PREP onto the infinitive phrase as an incomplete PG, noting that an implied relative head is missing. A higher level context-oriented mechanism can then (somehow) fill out the ellided "blocks are good for...".

The "differential diagnosis" nature of the module's task seems to be an excellent view of the world for handling minor ungrammaticality. A module never asks if a path is worth attempting, but rather which of the possible paths is the best. Grammatical constraints thus become evidence that a path is more correct than another, rather than entrance exams which must be passed before a path can be entered at all. If some constraint, such as person-number agreement, is absolutely essential to diagnose which of two possible paths is correct, it can be used as the final arbiter of the decision. If the decision is otherwise clear, such a constraint can be ignored. If the differentiation is very difficult (which I believe is rarely the case) more complex evidence weighing can be effected by giving the module a more complex decision tree.

It should be noted that in the case of minor ungrammaticality that the parser knows how to overcome, as well as in the case of utterances so bad they are unparsable, it is central to the above strategies that the parser is always sure that the path it is following is the only feasible path. Indeed, perhaps the greatest benefit that wait-and-see strategies provide is that the determinism I believe they make possible allows the parser to assume that the obstruction in its way is to be climbed over at all costs rather than taken as an indication that the road is closed.

A PARADIGM FOR RECOGNITION

Let us now back off from the specific problem of parsing natural language, and examine the structure of the WASP as a paradigm for recognition problems in general. Most recognition systems, whether for language, vision, or whatever, fall into either the top-down or bottom-up recognition paradigms, i.e. they are, for the most part, either data-driven or hypothesis-driven. Both paradigms have advantages, but both have distinct disadvantages. A data-driven system can react immediately and smoothly to the specific environment in which it finds itself, but will quickly become swamped if the domain in which it is to function provides too much information; in such an enviroment a data-driven system quickly suffers from serious sensory overload. A hypothesis-driven system, on the other hand, need look at only some small amount of data to confirm its preconception of its environment; it can ask its senses for only the specific information it needs to assure itself that its current hypothesis is valid. The problem with such a system, of course, is that if the hypothesis is wrong, alternative hypothesis must be found and tried until one is found which seems to be an accurate model of the world. (The notion (first suggested by Marvin Minsky <Minsky '74> and developed by Ben Kuipers as a model for block recognition <Kuipers '74>) of a transition net connecting stereotypic hypotheses, with transitions triggered by observations that conflict with the current hypothesis yet suggest another, may eliminate some thrashing between hypotheses, but I do not believe that it will entirely solve the problem, especially in complex domains.) The WASP, however, is neither data-driven nor hypothesis-driven, but rather a mixture of both. It is a combination of several notions that attempts to utilize the advantages of both paradigms.

The central idea behind the WASP recognition paradigm is the notion of situation specialists, actually a collection of notions. A situation specialist, implemented in the WASP as a module, is initially data-driven in that it is triggered by combinations of low-level features that jointly define the general "shape" of the environment, the "situation" that exists in the world. Determining this situation requires only a very low level of "sensory" resolution, hence this data-driven portion of the situation specialist need not be swamped by irrelevant detail and too much data. In such a world, with little detail and a need for only limited resolution, data-driven systems behave very well, as noted above, immediately responding to the actual structure of the environment.

Once a situation specialist is active, it then acts as a "differential diagnostician", becoming hypothesis-driven in a very special sense. A situation specialist can be said to be hypothesis-driven in that it knows exactly what the competing hypotheses are which may be models of that situation and in that it does "differential diagnosis" to determine exactly which one of them is right. It can more strictly be said to be hypothesis-driven in that it gathers information only by making specific requests of its senses, thereby retaining the principal advantage for recognition of hypothesis-driven systems without the need to confirm or reject possible hypotheses one at a time. Furthermore, the differential diagnosis notion allows the situation specialist to use information global to the entire set of competing hypotheses to efficiently determine which is best.

Another important notion is that of maintaining intermediate levels of structure. While this notion is certainly central to present data-driven vision systems that first find lines, then regions, then objects, the notion of intermediate levels of structure is also useful to systems that are less strictly bottom-up and that work in domains where the levels of structure are not quite so conceptually distinct. Forming an intermediate level of structure in a "middle-down" fashion, based on local criteria, allows detail irrelevant or of limited usefulness to higher level processes with more global concern to be hidden from the casual gaze of those processes. This conglomeration of data greatly reduces the complexity of the data-domain in which higher level processes must operate.

To what extent are these notions useful for othere sorts of recognition? While a general answer to this question will be left both as an exercise to the reader and as a "suggested area for future research", it is interesting to note that these notions are in part quite similar to those developed by Andee Rubin within the framework of a model of "disease recognition", i.e. medical diagnosis <Rubin '74>. In Rubin's model, disease hypotheses are first triggered by symptoms or combinations of symptoms, and then attempt to substantiate themselves in a very top-down manner. Her model first conglomerates data into symptoms as an intermediate level of structure, and then shifts to disease-centered processing, as higher level hypotheses become active.

CONCLUSION

As a final point, it is perhaps most appropriate to restate the key hypothesis of the theory of language on which the WASP is based. The WASP itself is very precisely a test of this theory: to the extent that this theory is true, the WASP will succeed; to the extent that it is false, the WASP will fail. The central hypothesis of the theory is simply this: that the process of building up the structure of natural language utterances should be viewed as consisting of several strata of processes, and that natural language provides explicit clues in local contexts at each level that allow a deterministic decision of what to do next. By "local", we mean within an environment of 3 or 4 or 5 contiguous units at any given level of structure. By "deterministic", we mean that except for "garden path" sentences, those on which people need to do some sort of conscious processing to determine the correct structure, decisions based on these local clues can be perfectly accurate. The task of writing a grammar for the WASP, to repeat a statement made in an earlier section, involves finding out exactly what these clues are and what form they take. Thus, if this theory is correct, the grammar of the WASP should shed a new sort of light on the nature of language.

BIBLIOGRAPHY

Scott Fahlman, A Hypothesis-Frame System For Recognition Problems, Working Paper 57, MIT-AI Lab, 1973

Carl Hewitt, Description and Theoretical Analysis of PLANNER, MIT-AI-TR-258, 1972

Benjamin Kuipers, An Hypothesis-Driven Recognition System for the Blocks World, Working Paper 63, MIT-AI Lab, 1974

Marvin Minsky, A Framework for Representing Knowledge, A.I. Memo 306, MIT-AI Lab, 1974

Allen Newell and H.A. Simon, Human Problem Solving, Prentice Hall, 1972

Andee Rubin, Hypothesis Formation and Evaluation in Medical Diagnosis, E.E. thesis, MIT, May 1974

David Waltz, Generating Semantic Descriptions from Drawings of Scenes with Shadow, MIT-AI-TR-271, 1972

Terry Winograd, Understanding Natural Language, Academic Press, 1972

William Woods, The Lunar Sciences Natural Language Information System, BBN Report No. 2378, 1972