A THEORY OF SYNTACTIC RECOGNITION FOR NATURAL LANGUAGE

MITCHELL P. MARCUS

Most natural language understanding systems model syntax using some form of the augmented transition network representation. Linguists have not been wild about augmented transition networks, however, feeling that the formalism lacks sufficient structure to capture properly the intricate constraints of natural languages. In this overview of his PhD thesis, Mitchell Marcus offers an alternative that features the structure linguists like while preserving the performance possibilities that only augmented transition networks offered before. In part, this is because Marcus' model differs from that of the augmented transition network in two particularly important ways: first, look ahead replaces backup; and second, transparent production-like rules replace obscure LISP code as a medium for representing procedural linguistic knowledge. Marcus' complete thesis is part of the MIT press series in Artificial Intelligence.
The Determinism Hypothesis

All current natural language parsers that are adequate to cover a wide range of syntactic constructions operate by simulating nondeterministic machines, either by using backtracking or by pseudo-parallelism. On the face of it, this seems to be necessary, for a cursory examination of natural language reveals many phenomena that seem to demand nondeterministic solutions if we restrict our attention to parsers that operate left-to-right.

A typical natural language parser is conceptually nondeterministic in that it will parse a given input if there is some sequence of grammar rules (however such rules are expressed) whose application yields a coherent analysis of the input, even if other legal sequences of rule application do not lead to such analyses. Since all physically existing machines must be deterministic, such a nondeterministic machine must be simulated by causing a deterministic machine to make "guesses" about what the proper sequence of actions for a given input should be, coupled with some mechanism for aborting incorrect guesses. For many inputs, this necessarily leads to the creation of some syntactic substructures which are not constituents in whatever final syntactic analysis is assigned to a given input.

To see that such an approach seems to be necessary, consider the sentences in figure 1. While the first seven words of both of these sentences are identical, their structures, as shown, are very different. In one "have" is the main verb of an imperative sentence, with the rest of the sentence a subordinate clause. In the other sentence, a question, "have" is an auxiliary of the main verb "taken," and "the students" is the subject of the main clause.

It would seem that to analyze the structure of these sentences in a left-to-right manner, a parser must necessarily simulate a nondeterministic process. Not only is it impossible to determine what role the word "have" serves in either of the given sentences on first encounter, but the two structures are identical up to the end of the NP "the students who missed the exam."
the input, it can also observe that the input was ambiguous, and flag the output analysis to indicate that this analysis is only one of a range of coherent analyses. Some external mechanism will then be needed to force the interpreter to reparse the input, taking a different analysis path, if the other consistent analyses are desired.

The internal state of the mechanism must be constrained in such a way that no temporary syntactic structures are encoded within the internal state of the machine. While this does not mean that the machine's internal state must be limited to a finite state control (the grammar interpreter to be presented below uses a push-down stack, among other control structures), it must be limited -- at least in its use -- in such a way that structure is not hidden in the state of the machine.

One immediate implication of all this is that a grammar for any interpreter which embodies these properties must constrain that interpreter from ever making a mistake, since the interpreter can only correctly analyze a given input if it never creates any incorrect structure. This means that such a grammar must at least implicitly specify how to decide what the grammar interpreter should do next, that is, it can never leave the grammar interpreter with more than one alternative.

The Structure of the Grammar Interpreter

Taking the Determinism Hypothesis as a given, an examination of natural language leads to a further set of properties which any deterministic grammar interpreter must embody. (Henceforth, the word "deterministic" means "strictly deterministic" in the sense discussed.) Any such interpreter must have the following properties: it must be at least partially data driven; but it must be able to reflect expectations that follow from general grammatical properties of the partial structures built up during the parsing process; and it must have some sort of look-ahead facility, even if it is basically left-to-right.

To show that each of these properties is necessary, it suffices to show a pair of sentences of English that cannot be distinguished by a mechanism without the given property, but which speakers of English understand without difficulty. The sentences shown in figure 2 below provide crucial pairs for each of the properties.

---

The parser must:

- Be partially data driven.
  - (1a) John went to the store.
  - (1b) How much is the doggie in the window?

- Reflect expectations.
  - (2a) I called [NP John] [S to make Sue feel better].
  - (2b) I wanted [NP John to make Sue feel better].

- Have some sort of look-ahead.
  - (3a) Have [S the boys take the exam today].
  - (3b) Have [NP the boys] [Vt taken the exam today].

---

Figure 2. Some examples which motivate the structure of the parser.

---

Almost by definition, a hypothesis driven parser cannot be deterministic, and thus a deterministic parser must necessarily be at least partially data driven. The essence of the problem is that any parser which is purely hypothesis driven, that is, which is purely top-down, must hypothesize several nested levels of structure before positing any constituents which can be checked against the input string itself.

For example, a top-down parser, newly given an input, might begin by hypothesizing that the input is a sentence. It might then hypothesize that the input is a declarative, and
therefore hypothesize that the input begins with a noun phrase. Assuming that the input begins with a noun phrase, it might finally hypothesize that the NP begins with a determiner, a hypothesis which is testable against the input string. At this point, the parser has created structures that correspond to the S and the NP, which will necessarily have to be discarded for at least some inputs. (These structures might be implicit in the state of the machine, but this is simply a matter of how the constituents are represented at this point in the parsing process.) To take a concrete example, even so different a pair of sentences as 2.1a and 2.1b cannot be deterministically analyzed by a hypothesis driven parser. The problem, of course, is simply that any hypothesis driven parser must either attempt to parse a given input as a declarative sentence, beginning, say, with an NP, before it attempts to parse it as a question, beginning with an auxiliary, or vice versa. Whatever order the parser imposes upon these two possibilities relative to each other, the clause type attempted first must be at least occasionally wrong. It is clear that if a parser is to be deterministic, it must look before it leaps.

A deterministic parser cannot be entirely bottom-up, however. Any parser that is purely bottom-up must initially misparse one of the two sentences given as 2.2a and 2.2b. The problem is that the string "John to make Sue feel better" can be analyzed in two different ways: as one constituent that is an infinitive complement, as in 2.2b; or as two unrelated constituents, as in 2.2a, with the NP "John" the object of the verb and the phrase "to make Sue feel better" an adverbial "purpose" clause. The difference in structure between 2.2a and 2.2b can be predicted, however, if the parser can note that "want" typically takes an infinitive complement, while "call" cannot take such a complement. Thus, a deterministic parser must have some capacity to use whatever information and expectations can be gleaned from an examination of the structures that have been built up at any given point in the parsing process. If a parser is to operate deterministically, it must use such information to constrain the analysis imposed on the remainder of the input.

Finally, if a deterministic parser is to correctly analyze such pairs of sentences as 2.3a and 2.3b above, it cannot operate in an entirely left-to-right manner. As was discussed, it is impossible to distinguish between this pair of sentences before examining the morphology of the verb following the NP "the boys." These sentences can be distinguished, however, if the parser has a large enough "window" on the clause to see this verb; if the verb ends in "en" (in the simple case presented here), then the clause is an yes/no question, otherwise it is an imperative. Thus, if a parser is to be deterministic, it must have some facility for look ahead. It must be stressed, however, that this look-ahead ability must be constrained in some manner; otherwise the determinism claim is vacuous.

We now turn to a grammar interpreter called PARSIFAL, whose structure is motivated by the three principles discussed above. This grammar interpreter maintains two major data structures: a push down stack of incomplete constituents called the active node stack, and a small three-place constituent buffer which contains constituents which are complete, but whose higher level grammatical function is as yet uncertain.

Figure 3 shows a snapshot of the parser's data structures taken while parsing the sentence "John should have scheduled the meeting." At the bottom of the stack is an auxiliary node labelled with the features modal, past, among others, which has as a daughter the modal "should." (This stack grows downward, so that the structure of the stack reflects the structure of the emerging parse tree.) Above the bottom of the stack is an S node with an NP as a daughter, dominating the word "John." There are two words in the buffer, the verb "have" in the first buffer cell and the word "scheduled" in the second. The two words "the meeting" have not yet come to the attention of the parser. (The structures of form "(PARSE-AUX CPOOL)" and the like will be explained below.) The constituent buffer is really the heart of the grammar interpreter; it is the central feature that distinguishes this parser.
from all others. The words that make up the parser’s input first come to its attention when they appear at the end of this buffer after morphological analysis. After the parser builds these words into some larger grammatical structure at the bottom of the active node stack, it may then pop the new constituent from the active node stack and insert this completed constituent into the first cell of the buffer if the grammatical role of this larger structure is as yet undetermined. The parser is free to examine the constituents in this buffer, to act upon them, and to otherwise use the buffer as a workspace.

In general, the parser uses the buffer in a first-in, first-out fashion. It typically decides what to do with the constituent in the leftmost buffer position after taking the opportunity to examine its immediate neighbors to the right. The availability of the buffer allows the parser to defer using a word or a larger constituent that fills a single buffer cell until it has a chance to examine some of the right context of the constituent in question. Thus, for example, the parser must often decide whether the word “have” at the beginning of a clause initiates a yes/no question, as in 1.2b, or an imperative, as in 1.2a. The parser can often correctly decide what sort of clause it has encountered, and thus how to use the initial verb, by allowing several constituents to “pile up” in the buffer. Consider for example, the snapshot of the buffer shown in figure 4. By waiting until the NP “the boys,” NP25, is formed, filling the 2nd

"Figure 3. PARSIFAL’s two major data structures."

"Figure 4. The buffer allows the parser to examine local context."

buffer position, and WORD37, the verb "do," enters the buffer, filling the 3rd buffer position, the parser can see that the clause must be an imperative, and that "have" is therefore the main verb of the major clause.

Note that each cell in the buffer can hold a grammatical constituent of any type, where a constituent is any tree that the parser has constructed under a single root node. The size of the structure underneath the node is immaterial; both "that" and "that the big green cookie monster’s toe got stubbed" are perfectly good constituents once the parser has constructed the latter phrase into a subordinate clause.

The constituent buffer and the active node stack are acted upon by a grammar which is made up of pattern/action rules; this grammar can be viewed as an augmented form of Newell and Simon’s production systems [Newell and Simon 1972]. Each rule is made up of a pattern, which is matched against some subset of the constituents of the buffer and the accessible nodes in the
active node stack (about which more will be said below), and an action, a sequence of operations which act on these constituents. Each rule is assigned a numerical priority, which the grammar interpreter uses to arbitrate simultaneous matches.

The grammar as a whole is structured into rule packets, clumps of grammar rules which can be turned on and off as a group; the grammar interpreter only attempts to match rules in packets that have been activated by the grammar. At any given time during the parsing process, the grammar interpreter only attempts to match those rules which are in active packets. Any grammar rule can activate a packet by associating that packet with the constituent at the bottom of the active node stack. If a node at the bottom of the stack is pushed into the stack, the active packets remain associated with it, but are only active when that node is again at the bottom of the stack. For example, in figure 3, the packet BUILD-AUX is associated with the bottom of the stack, and is thus active, while the packet PARSE-AUX is associated with the S node above the auxiliary.

The grammar rules themselves are written in a language called PIDGIN, an English-like formal language that is translated into LISP by a simple translator based on the notion of top-down operator precedence [Pratt 1973]. Figure 5 gives a schematic overview of the organization of the grammar, and exhibits some of the rules that make up the packet PARSE-AUX.

The parser (that is the grammar interpreter: interpreting some grammar) operates by attaching constituents which are in the buffer to the constituent at the bottom of the stack until that constituent is complete, at which time it is popped from the stack. If the constituents in the buffer provide clear evidence that a constituent of a given type should be initiated, a new node of that type can be created and pushed onto the stack; this new node can also be attached to the node currently at the bottom of the stack before the stack is pushed, if the grammatical function of the new constituent is clear at the time it is created. When popped, a constituent either remains attached to its parent, if it was attached to some larger constituent when it was created, or

(a) - The structure of the grammar.

(b) - Some sample grammar rules that initiate and attach auxiliaries.

Figure 5. The structure of the grammar and some example rules.
else it falls into the constituent buffer (which will cause an error if the buffer was already full).

This structure embodies the principles discussed above in the following ways:

- **A deterministic parser must be at least partially data driven.** A grammar for PARSIFAL is made up of pattern/action rules which are triggered, in part, when lexical items or unattached higher level constituents fulfilling specific descriptions appear in the buffer. Thus, the parser is directly responsive to the input it is given.

- **A deterministic parser must be able to reflect expectations that follow from the partial structures built up during the parsing process.** Since PARSIFAL only attempts to match rules that are in active packets, grammar rules can activate and de-activate packets of rules to reflect the properties of the constituents in the active node stack. Thus grammar rules can easily be written that are constrained by whatever structure the parser is attempting to complete.

- **A deterministic parser must have some sort of constrained look-ahead facility.** PARSIFAL’s buffer provides this constrained look-ahead. Because the buffer can hold several constituents, a grammar rule can examine the context that follows the first constituent in the buffer before deciding what grammatical role it fills in a higher level structure. I argue that a buffer of quite limited length suffices to allow deterministic parsing. The key idea is that the size of the buffer can be sharply constrained if each location in the buffer can hold a single complete constituent, regardless of that constituent’s size.

We now turn to the structure of individual grammar rules. Some example rules were given in figure 5; this section will explain the syntax of those rules.

The pattern of a grammar rule is made up of a list of partial descriptions of parse nodes which must all be fulfilled if the pattern is to match. There can be up to five partial descriptions in each pattern: up to three consecutive descriptions which are matched against the first, second, and third constituents in the buffer (in order), and two descriptions which match against the two nodes in the active node stack accessible to the parser. These two nodes are the bottom node on the stack, which will be referred to as the current active node, and the S or NP node closest to the bottom of the stack which will be called the dominating cyclic node or alternatively, if an S, the current S node. In figure 3, AUX1 is the current active node, and S1 is the current S node. As we shall see later, making the dominating cyclic node explicitly available for examination and modification seems to eliminate the need for any tree climbing operations that ascend tree structures.

The syntax of the grammar rules presented is self-explanatory, but a few comments on the grammar notation itself are in order. The general form of each grammar rule is:

```
{Rule <name> priority: <priority> in <packet>
 <pattern> -> <action>}
```

Each pattern is of the form:

```
[<description of 1st buffer constituent>] [<2nd>] [<3rd>]
```

The symbol "=" used only in pattern descriptions, is to be read as "has the feature(s)." Features of the form "<word>" mean "has the root <word>," for example "heave" means "has the root "have"." The tokens "1st," "2nd," "3rd," and "C" (or "c") refer to the constituents in the 1st, 2nd, and 3rd buffer positions and the current active node (that is the bottom of the stack), respectively. (I will also use these tags in the text below as names for their respective constituents.) The symbol "t" used in
a pattern description is a predicate that is true of any node; thus "1" is the simplest always true description. Pattern descriptions to be matched against the current active node and the current S are flagged by "..C" appearing at the beginning of an additional pattern description.

Each description is made up of a Boolean combination of tests for given grammatical features. Each description can also include Boolean feature tests on the daughters of the target node; the grammar language provides a tree walking notation for indicating specific daughters of a node. (While the parser can access daughter nodes, it cannot modify them.) While this richness of specification seems to be necessary, it should be noted that the majority of rules in even a moderately complex grammar have patterns which consist only of tests for the positive presence of given features.

The action of a grammar rule consists of a rudimentary program that does the actual work of building constituent structures. An action is built up of primitives that perform such actions as these:

- Creating a new parse node, pushing the newly created node onto the bottom of the active node stack. A new node is presumably created whenever the parser decides that the first constituent(s) in the buffer are the initial daughters of a constituent not yet created by the parser.

- Inserting a specific lexical item into a specific buffer cell, which causes the previous contents of that cell to be shifted one place to the right.

- Popping the current active node from the active node stack, causing it to be inserted into the first buffer cell if it has not been previously attached to another node, shifting the previous contents of that cell one place to the right.

- Attaching a newly created node or a node in the buffer to the current active node or the current cyclic node. After each grammar rule is executed, the grammar interpreter removes all newly attached nodes from the buffer, with nodes to the previous right of each deleted node shifting to the left. Note that a newly created node can either be attached to a parent node at the time of its creation, if its function in higher level grammatical structure is clear at that time, or it can be created without an attachment, in which case it will be dropped into the buffer when it is popped from the active node stack.

- Assigning features to a node in the buffer or one of the accessible nodes in the stack.

- Activating and deactivating packets of rules.

PIDGIN also provides primitives from which Boolean tests of the features of a node can be constructed, as mentioned above. These predicates can be used within conditional "if...then...else..." expressions to conditionally perform various operations.

With the exception of allowing conditional expressions, PIDGIN rules fall into the simple class of programs called fixed-instruction programs. The PIDGIN language imposes the following constraints on rule actions:

- There are no user-settable variables within the rule actions. The constituents in the first three buffer cells are available as the values of the parameters 1st, 2nd, and 3rd within each rule, but these parameters are given values by the grammar interpreter before each rule action is called, and are not resettable within an action. The values of the parameters C and the current cyclic node do change within a rule as nodes are pushed and popped
from the active node stack, but their values cannot be set by a grammar rule.

- **PIDGIN** allows no user-defined functions; the only functions within actions are PIDGIN primitives. (There is a limited ability for a rule to circumvent the pattern-matching process by explicitly naming its successor, but this is formally only a device for rule abbreviation, since the specification of the successor rule could simply be replaced with the code for the action of the rule named.)

- While conditional "if...then...else..." expressions are allowed, there is no recursion or iteration within actions.

- The only structure building operations in PIDGIN are (a) attaching one node to another, and (b) adding features to a node's feature set. In particular, the list building primitives of LISP are not available in PIDGIN.

### Parsing a Simple Declarative Sentence

Now I present a small grammar which is just sufficient to parse a very simple declarative sentence, and then trace through the process of parsing the sentence immediately below, given this grammar. The emphasis here will not be on the complexities of the grammar, but rather on the form of the grammar in broad outline and on the details of the workings of the grammar interpreter.

*John has scheduled the meeting.*

One simplification will be imposed on this example: it will be assumed that all NPs come into the buffer parsed, that the structure of NPs is determined by a mechanism that is transparent to the "clause-level" grammar rules that will be discussed here. Such a mechanism is in fact presented in Marcus [1977]; I will say here only that this mechanism involves relatively slight extensions of the mechanism presented here.

As a convention, the parser begins every parse by calling the grammar rule named INITIAL-RULE. This rule creates an S node and activates the packet SS-START (Simple Sentence-START), that contains rules that decide on the type of simple sentences. The parser's state after this rule is executed is depicted in figure 6. At this point there is nothing in the buffer.

Constituents enter the buffer on demand -- the buffer mechanism will get the next constituent from the input word stream when a rule pattern must be matched against a buffer cell that is currently empty. Furthermore, before the grammar interpreter will attempt to match a rule of a given priority, all higher prioritied rules must explicitly fail to match. This means that throughout the examples, constituents will often enter the buffer for no apparent reason. These constituents were requested by rules that ultimately failed to match, leaving no trace of why each constituent entered the buffer.

The packet SS-START, some of whose rules are shown in figure 7, contains rules which determine the type of a major clause. If the clause begins with an NP followed by a verb, then the clause is labelled a declarative; if it begins with an auxiliary verb followed by an NP, it is labelled a yes/no question. If the clause begins with a tenseless verb, then not only is the clause labelled an imperative, but the word "you" is inserted into the buffer, where it is inserted at the beginning of the buffer by convention.

After INITIAL-RULE has been executed and packet SS-START has been activated, the rule MAJOR-DECL-S matches, with the pattern matching process pulling the NP "John" and the verb "has" into the buffer. The action of the rule is now run, labelling the clause a major declarative clause, deactivating the packet SS-START, and activating the packet PARSE-SUBJ. The result of this is shown in figure 8.

The packet PARSE-SUBJ contains rules whose job it is
Figure 6. After INITIAL-RULE has been run.

Figure 7. Some rules that determine sentence type.

Figure 8. After the rule MAJOR-DECL-S is run.

to find and attach the subject of the clause under construction. It contains two major rules which are shown in figure 9. The rule UNMARKED-ORDER picks out the subject in clauses

Figure 9. Two subject-processing rules.
rule AUX-INVERSION picks out the subject in clauses where an element of the auxiliary occurs before the subject. Though the relevant rules will not be discussed here, the rule UNMARKED-ORDER will pick up the subject of imperatives and WH-questions where the subject of the clause is questioned, while AUX-INVERSION will pick up the subject of WH-questions that question other than the subject of the clause.

At this point in the parse, the rule UNMARKED-ORDER now matches, and its action is run. This rule attaches 1st, NP40, the NP "John," to C, the node S16, as subject. It also activates the packet PARSE-AUX after deactivating PARSE-SUBJ. After this rule has been executed, the interpreter notes that the NP has been attached, and removes it from the buffer. Figure 10 shows the state of the parser after

Figure 10. After UNMARKED-ORDER has been executed.

For brevity the process of parsing the auxiliary verbs in this example will not be discussed, although figure 11 provides a trace of the application of rules during the parsing of the auxiliary phrase, through the attachment of the auxiliary to the VP node. The rules referred to in the trace are included in figure 12.

It should be noted that the rules of packet BUILD-AUX,
About to run: AUX-ATTACH

The Active Node Stack (0. deep):
C: S16 (S DECL MAJOR S) / (PARSE-AUX)
   NP: (John)

The Buffer
1: AUX14 (PERF PRES V3S AUX) : (has)
2: WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...): (scheduled)

Yet unseen words: a meeting.

The Active Node Stack (0. deep):
C: S16 (S DECL MAJOR S) / (PARSE-VP)
   NP: (John)
   AUX: (has)

The Buffer
1: WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...): (scheduled)

Yet unseen words: a meeting.

Figure 11. Parsing the auxiliary of (i).

{RULE START-AUX PRIORITY: 10. IN PARSE-AUX
[=verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and vspl, v1s, v+13s, vpl+2s, v-3s, v3s.
%(The above features are "person/number codes", e.g. "vpl+2s" means that this verb goes with any plural or 2nd person singular
np as subject. The verb "are" has this feature.)%
Label C with the meet of the features of 1st and pres,
   past, future, tensless.
Activate build-aux.)

{RULE AUX-ATTACH PRIORITY: 10. IN PARSE-AUX
[=aux] -->
Attach 1st to c as aux.
Activate parse-vp. Deactivate parse-aux.)

{RULE PERFECTIVE PRIORITY: 10. IN BUILD-AUX
[=*have] [=en] ---> Attach 1st to c as perf. Label c perf.)

{RULE PROGRESSIVE PRIORITY: 10. IN BUILD-AUX
[=*be] [=ing] ---> Attach 1st to c as prog. Label c prog.)

{RULE PASSIVE-AUX PRIORITY: 10. IN BUILD-AUX
[=*be] [=en] ---> Attach 1st to c as passive. Label c passive.)

{RULE AUX-COMPLETE PRIORITY: 15. IN BUILD-AUX
[t] ---> Drop c into the buffer.)

Figure 12. Some rules which parse auxiliaries.
some of which are shown in figure 12, are the equivalent of the transformational rule of affix-hopping. Note that these rules concisely state the relation between each auxiliary verb and its related affix by taking advantage of the ability to buffer both each auxiliary verb and the following verb. It might seem that some patch is needed to these rules to handle question constructions in which, typically, the verb cluster is discontinuous, but this is not the case, as will be shown later.

The packet PARSE-VP is now active. This packet contains, among other rules, the rule MVB (Main Verb), which creates and attaches a VP node and then attaches the main verb to it. This rule now matches and is run. The rule itself, and the resulting state of the parser, is shown in figure 13.

At the time MVB is executed, the packet PARSE-VP is associated with S16, the current active node, as shown in the last frame of figure 11. The action of MVB first deactivates the packet PARSE-VP, then activates either SS-FINAL, if C is a major clause, or EMB-S-FINAL, if it is an embedded clause. These two packets both contain rules that parse clause-level prepositional phrases, adverbs, and the like. They differ in that the rules in EMB-S-FINAL must decide whether a given modifier should be attached to the current clause, or left in the buffer to be attached to a constituent higher up in the parse tree after the current active node is completed. Whatever packet is activated, the newly activated packet will be associated with C, S16, and thus the grammar interpreter will attempt to match the rules in it whenever S16 is the current active node.

The execution of the next line in the action of MVB results in a new VP node, VP14, being attached to S16 and then pushed onto the active node stack, becoming the current active node. Next the verb "scheduled", WORD127, is attached to the VP, and then the action of MVB activates the packet SUBJ-VERB. As is always the case with packet activation, this packet is associated with the node which is the current active node at the time of its activation, in this case, VP14. Thus, this rule leaves the parser with the packet SS-FINAL associated with S16, and the packet SUBJ-VERB associated with VP14. Since VP14 is the current active node, SUBJ-VERB is now active. Once VP14 is popped from the stack, leaving S16 the current active node, SS-FINAL will be active.

The packet SUBJ-VERB contains rules which involve the deep grammatical relation of the surface subject of the clause to the verb, and which set up the proper environment for parsing the objects of the verb. The major rule in this packet, the rule

{(RULE MVB IN PARSE-VP

[=verb] -->

Deactivate parse-VP.
If c is major then activate ss-final else
If c is sec then activate emb-s-final.
Attach a new vp node to c as vp.
Attach 1st to c %which is now the vp% as verb.
Activate sub-verb.)

The Active Node Stack ( 1. deep)
S16 (S DECL MAJOR S) / (SS-FINAL)
NP : (john)
AUX : (has)
VP : 
C : VP14 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer
1 : WORD127 (*A NGSTART DET NS N3P ... : (a)

Yet unseen words: meeting.

Figure 13. The rule MVB and the parser's state after its execution.
SUBJ-VERB is shown in figure 14. Note that this rule has a very low priority and a pattern that will always match; this rule is a default rule, a rule which will become active when no other rule can apply. While most of the code of the action of the rule SUBJ-VERB does not apply to our example, I have refrained from abbreviating this rule to give a feel for the power of PIDGIN.

The rule SUBJ-VERB is now the rule of highest priority that matches, so its action is now executed. The purpose of this rule is to activate the appropriate packets to parse the objects and complements of a clause; the activation of some of these packets depends upon the verb of the clause, while the activation of others depends upon more global properties of the clause. Thus, the next several lines of the action activate packets for various sorts of complement constructions that a verb might take: infinitive phrases in general (the packet INF-COMP), infinitive phrases that are not flagged by "to," (the packet TO-LESS-INF-COMP in addition to INF-COMP), and tensed complements (the packet THAT-COMP). The next long clause activates one of a number of packets which will attach the objects of the verb. The packet activated depends on the clause type: whether this clause still has a WH-head that needs to be utilized (WH-VP), whether this clause is secondary without a WH-head (EMBEDDED-S-VP), or whether this clause is a major clause (SS-VP).

This rule provides a good example of one difference between the packets used to organize this grammar and the states of an ATN. Packets do not simply correspond to ATN states, for several packets will typically be active at a time. For instance, if parsing the sentence "Who did John see Jane kiss?" this rule would activate three packets: INF-COMP, TO-LESS-INF-COMP, and WH-VP. In terms of the ATN model, one can think of this rule as dynamically tailoring the arcs out of a state of an ATN to exactly match various properties of the clause being parsed.

Of the complement-initiating packets, only the packet INF-COMP is activated in the example, since "schedule" can take infinitive complements, as in "Schedule John to give a lecture on Tuesday." The packet SS-VP is then activated to attach the verb's objects, the packet SUBJ-VERB is deactivated, and the rule SUBJ-VERB is through. This rule thus changes the state of the parser only by activating and deactivating packets; the packets now associated with the current active node are SS-VP and INF-COMP. Figure 15 shows the rules in the packet SS-VP that will come into play in the example. Note that the rule VP-DONE, like the rule SUBJ-VERB above, is a default rule.

The rule OBJECTS is now triggered by NP41, and attaches the NP to VP14. The state of the parser at this point is shown in figure 16.

Completing the parse is now a simple matter. The default rule in packet SS-VP, VP-DONE, now triggers, popping the VP node from the active node stack. Since VP14 is attached
to S16, the S node above it on the stack. It remains hanging from the S node. The S node is once again the current active node, and packet SS-FINAL is now active. The default rule in this packet, SS-DONE, shown in figure 17, immediately triggers. This

rule attaches the final punctuation mark to the clause, pops the S node from the stack, and signals the grammar interpreter that the parse is now complete, bringing our example to a close.

Implications of the Mechanism

This research began with the assumption that English could be parsed with greater ease than had previously been assumed. This assumption in turn leads to several principals which any such parser must embody, principals which PARSIFAL reflects rather directly. The grammar is made up of pattern/action rules, allowing the parser to be data directed. These rules themselves are clustered in packets, which can be activated and deactivated to reflect global expectations. The grammar interpreter’s constituent buffer gives it a small window through which to view the input string, allowing restricted look-ahead.

In conclusion, I will briefly sketch how specific structural properties of the parser allow the range of linguistic generalizations indicated at the beginning of this report to be captured.

Of the structures that make up the grammar interpreter, I believe, it is the constituent buffer which is most central to these results. The most important of these results follow from the
sorts of grammar operations that the buffer makes feasible; others follow from the limitations that its fixed length imposes. All however, hinge crucially upon the existence of the buffer. This data structure, I submit, is the primary source of the power of the parser.

For example:

- Because the buffer automatically compacts upon the attachment of the constituents that it contains, the parsing of a yes/no question and the related declarative will differ in one rule of grammar, with the key difference restricted to the rule patterns and one line of the rules' actions. The yes/no question rule explicitly states only that the NP in the second buffer cell should be attached as the subject of the clause. Because the buffer will then compact, auxiliary parsing rules that expect the terminals of the verb cluster to be contiguous will then apply without need for modification.

- Because the buffer provides a three-constituent window on the structure of a clause, diagnostic rules can be formulated that allow the differential diagnosis of pairs of constructions that would seem to be indistinguishable by a deterministic parser without such a buffer. Thus, evidence that would seem to imply that natural language must be parsed nondeterministically can be explained away.

- Because the buffer can only contain a limited number of constituents, the power of these diagnostic rules is limited, leading to a principled distinction between sentences which are perceived as garden paths and those which are not. This explanation for why certain sentences cause garden paths is consistent with a series of informal experiments conducted during this research. Furthermore, this theory led to the counter-intuitive prediction that as simple a sentence as "Have the packages delivered tomorrow," would cause a garden path, a prediction which was confirmed by informal experiment (20 out of 40 people questioned were "led down the garden path" by this sentence).

In short, this one mechanism not only allows the formulation of rules of syntax that elegantly capture linguistic generalizations, it also provides the underpinnings for a psychological theory that seems to have high initial plausibility.

Another important source of power is the use of Chomsky's notion of a trace, a dummy constituent that allows the parser to indicate the "deep" underlying positions of constituents in an annotated surface structure. Especially important is the fact that a trace can be dropped into the buffer, thereby indicating its underlying position in a factorization of the terminal string without specifying its position in the underlying tree.

From this follows a simple formulation of passive which accounts for the phenomenon of "raising." The essence of the passive rule -- create a trace, bind it to the subject of the current S; drop it into the buffer -- is noteworthy in its simplicity. Again, the availability of the buffer yields a very simple solution to a seemingly complex linguistic phenomenon.

Moreover, we also have an explanation for the phenomena which underlie Chomsky's Specified Subject Constraint and Subjacency Principle. It is most interesting that the simple formulation of Passive presented here -- perhaps the simplest possible formulation of this rule within the framework of PARSIFAL -- behaves exactly as if these constraints were true; that is, that the formulation presented here, by itself and with no extraneous stipulations, leads to the behavior that these constraints attempt to specify.

There are several important areas for which an account must be given if the model presented is to resemble a full model of the syntactic recognition of natural language. The following
is a list of several topics which have yet to be adequately investigated:

- Much further investigation of phenomena which I believe require extensive semantic processing, such as conjunction, or PP attachment, is still required. I believe that they will require mechanisms outside of the basic grammar interpreter presented here. It should be noted that Woods does not handle these two phenomena within the framework of his ATN mechanism itself, but rather he posits special purpose mechanisms to deal with them. I share his intuitions.

- I have not yet investigated lexical ambiguity, but rather have focussed on structural ambiguity. An ambiguity is structural when two different structures can be built up out of smaller constituents of the same given structure and type; an ambiguity is lexical when one word can serve as various parts of speech. While part of the solution to this very pervasive problem seems to involve notions of discourse context, I believe that much of the problem can be solved using the techniques presented in this paper, but have not yet investigated this problem.

- I have yet to provide an account in terms of this model for the psychological difficulty caused by center embedded sentences such as "The rat the cat the dog chased bit ate the cheese." For a recent theory of this phenomena which is consistent with the model presented here, see Cowper [1976]. Cowper's results have yet to be integrated into this model, however.

References


