American Journal of Computational Linguistics *Microfiche 32 : 9*

P E D A G L O T AND UNDERSTANDING NATURAL LANGUAGE PROCESSING

WILLIAM FABENS

Computer Science Department
Rutgers University
New Brunswick, New Jersey 08903

ABSTRACT

PEDAGLOT is a programmable parser, a 'meta-parser.' To program it, one describes not just syntax and some semantics, but also--independently--its modes of behavior. The PEDAGLOT formulation of such modes of behavior follows a categorization of parsing processes into attention-control, discovery, prediction and construction. Within these overall types of activities, control can be specified covering a number of syntax-processing and semantics-processing operations. While it is not the only possible way of programming a meta-parser, the PEDAGLOT mode-specification technique is suggestive in itself of various new approaches to modeling and understanding some language processing activities besides parsing, such as generation and inference.

This work was sponsored by through NIH Grant #RR643.

Introduction

It is well known that to process natural language, one needs both a syntactic description of possible sentences, blended in some way with a semantic description of a certain domain of discourse, and a rather detailed description of the actual processes used in hearing or producing sentences.

An augmented transition network (Woods, 1970) is an example of the blending of syntactic and quasi-semantic descriptions. Here registers would be repositories of, or pointers to, semantics. When used in conjunction with a semantic network, an ATN can be used to parse or to generate (Simmons and Slocum, 1972) sentences. The issue of changing the description of the actual processes used in such systems has been touched on by Woods (in using a 'generation mode'), to some extent by Simmons and Slocum (using decision functions to control style of generation), and to a larger extent by Kaplan (1973), in his General Syntactic Processor, GSP. GSP indeed is one example of a system in which syntax, semantics and to some extent processes can each be usefully defined.

If we look at syntax, semantics and processes as three describable components, these systems just mentioned illustrate how thoroughly intertwined they can become--to the extent that theorists from time to time deny the existence or at least the importance of some one of them. Ignoring that dispute, I would like to concentrate on the question of being able to comprehensively describe one's theory of language in terms of its syntax, semantics and processes in a way that allows for their necessary and extensive intertwining connections, but at the same time allows one to describe them independently.

I came to the need for doing this while designing a 'relaxation parser,' a parser which can make grammatical relaxations if it is given an ill-formed string, so as to arrive at a 'closest' possible parse for the string. This problem involved describing a 'correct' grammar and then (in some way) describing a space of deviations

that might be allowed by the parser. Thus the syntax would be fixed and the way the parser uses it would separately have to be described. It was soon noticed that efficiency could be greatly enhanced if some rudimentary notion of semantic plausibility could also be used. It would have to be described in a way related to the correct syntax but still be usable by the parser. Thus, for my purposes, the descriptions had to be independent of one another.

One feature of a relaxation parser is that it can 'fill in the gaps' of a string that is missing various words. If one could, which my relaxation parser did not, specify the semantic context of a sentence, the generated sentence might be semantically rather plausible. In any case, the relaxation parser operates in various respects like an actual parser or like a generator, and it was this relationship between parsing and generating that became of interest.

Out of the design of the relaxation parser, the notation (independent of syntax) which to some extent describes various processes and choices of alternate ways of processing was developed. Thus, one may take a set of syntax and semantic descriptions and then through describing the processing 'modes' involved, define a processor which uses the particular algorithm that the individual processes together define. One may call the parser that is programmable in its processes a meta-parser, of which various existing parsers and generators appear to be special cases.

A closer examination of the parser I have developed (called PEDAGLOT*) may show some such aspects of meta-parsing, especially as regards the relationship between parsing and generating. I will describe the syntactic and semantic parts of the parser first by noting its resemblances to the parser of J. Earley (1970) and the ATN system of Woods. Then I will describe the process-type specifications that are available, and the use of meta-parsers as a basis for defining general language behaviors. Further detail can be found in the PEDAGLOT manual (Fabens, 1972 and 1973).

*for pedagogic polyglot

1. The Core of the Parser

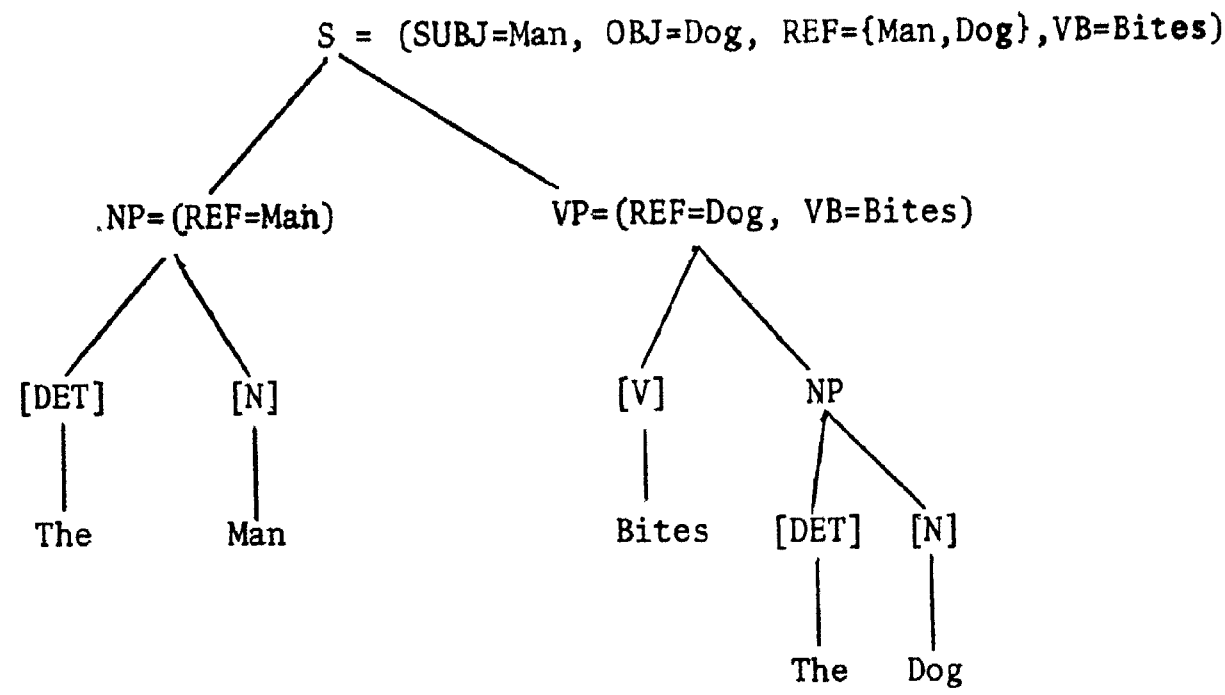
The fundamental operation of the parser is very similar to the operation of Earley's parser, with augmentations for recording the results of parses (e.g., their tre structure, and various of their attributes, which I call 'tags'). It is given a grammar as a set of context-free rules with various extensions, most important of which are that LISP functions may be used as predicates instead of terminals, and that each rule may be followed by operations that are defined in terms of the syntactic elements of the rule in question.

An example of this notation is as follows:

```
S → NP  VP
  => [AGREE [REF NP][VB VP]]
     [SUBJ = [REF NP]][OBJ = [REF VP][VB = [VB VP]]
S → NP [BE][VPASS] BY NP
  => [AGREE [REF NP][VB[BE]]]
     [SUBJ = [REF NP']][OBJ = REF NP]][VB = [VB[VPASS]]]
NP → [DET][N]
     => [REF = [N]]
VP → [V]NP
     => [VB = [V]][REF = [REF NP]]
```

Here, each bracketed symbol is the name of a recognition predicate (e.g., [N] recognizes nouns, [BE] recognizes forms of 'to be'). Following the => are the post-recognition functions. For instance [AGREE [REF NP][VB VP]] specifies a call to the AGREE function which is given, as arguments, the REF attribute (tag) of the sub-parse involved in that rule and the VB attribute of the VP part of the rule.

Following is a parse tree for 'The Man Bites the Dog' and values of tags after the parse.



The general flow of the parser is from top-down, and as the lowest components (symbols in the string) are found, the post-recognition functions that are associated with the rule that recognized them are applied. Tags become associated with sub-parses when the post-recognition operation uses the form [x = y] (in which the value referenced by y is stored as the x tag of the sub-parse). In the example, [DET] and [N] recognize 'The Man' and 'Man' is used as the REF attribute of the first NP. In the second S rule, the operation of [SUBJ = [REF NP']] would be to retrieve the REF tag of the second NP (thus the prime), and to store that as the SUBJ tag of the final parse.

As in most top-down parses, this parser begins with S and its two rules, since S is non-terminal. S is expanded into the two sequences of matches it should perform. This expansion results in various (in this case, two) predictions of what to find next. When the initial symbol in some rule is a terminal or a predicate, a discovery is called for (in which a match is performed, possibly involving the known values of the tags). When some complete sequence of elements is found (here, for instance, when NP → [DET][N] has matched the [N]). Construction invokes the post-recognition operations and then usually completes some earlier part of a rule (here, the 'NP' of S → NP VP) so further predictions (involving VP) or discoveries

are then specified.

I have broken up the parsing process into these three parts so as to similarly catalog the 'parsing modes,' turning this parser into a meta-parser. Before doing so, I should note that this parser stores each result under construction in a 'chart' as is done by Kaplan in his GSP, so that, for instance, the NP 'test' will only have to be evaluated once for each place one is wanted in the string.

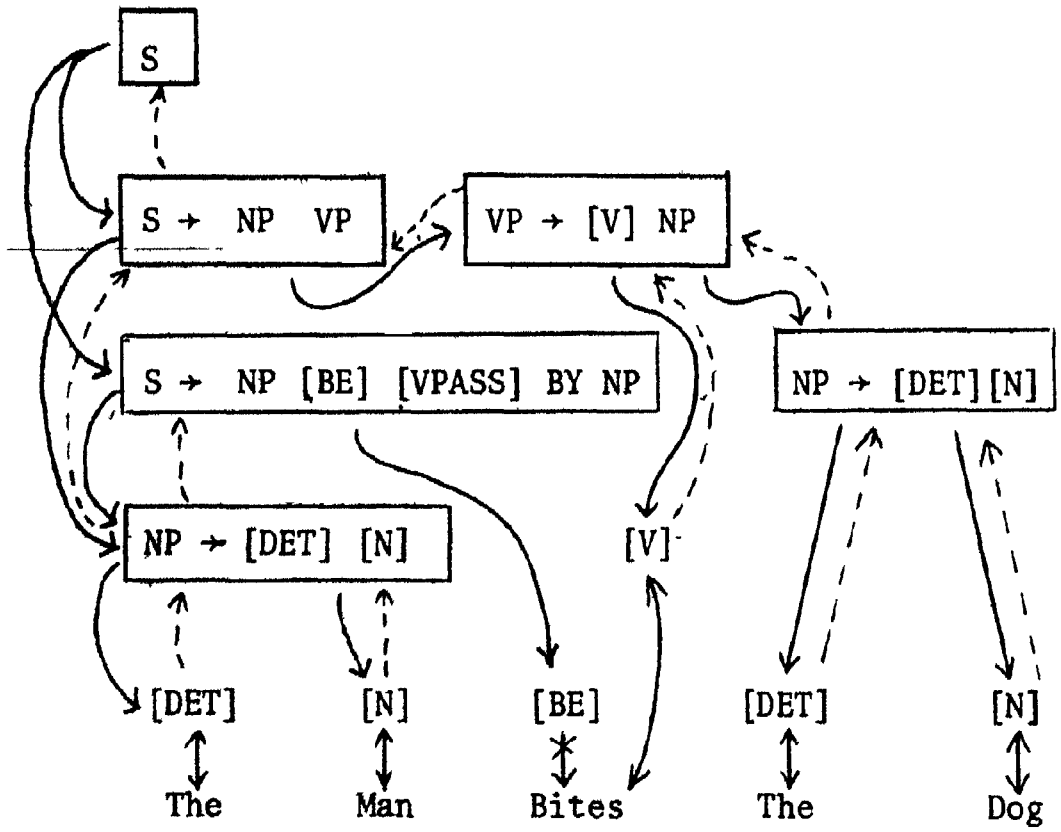


Illustration of PEDAGLOT's Parsing Chart

Simple Arrows indicate 'Predictions.'

Double Head Arrows indicate 'Discoveries.'

Dotted Arrows indicate "Construction.'

Also, for various well known reasons of efficiency, Earley's concept of independent processing of syntactic events is used (combined conceptually with the chart), so that a main controller can evaluate the individual syntactic 'tests' in almost any order, and not just in a backtracking sense (cf. Woods, 1975). The efficiency is realized here since many 'partial parses' (partially recognized forms)

are effectively abandoned if other results can complete the parse, or a sub-parse, first.

2. Meta-Parsing Modes

One can see that, except for the notational inefficiencies of the context-free formalism (as opposed to the augmented transition network form), this parser is very much like other standard parsers (especially ATN's). It differs in that there is a way of specifying how to proceed. Currently, this system has approximately a dozen 'modes' and I will present some of them here. Each mode specifies how to handle a certain part of the parsing process. They can be classified into four categories: attention control, prediction, discovery and construction.

a. Attention Control Modes:

Since the parser operates on a chart of independent events ('parsing questions'), one must give the parser a method of sequencing through them. Thus, one may specify 'breadth-first' or 'depth-first' and the appropriate mechanism will be invoked (this merely involves the way the processor stacks its jobs). A 'best-first' option is under development, which, when given an evaluation function to be applied to the set of currently active partial parses, allows the system to operate on the 'best' problem next. Experiments with this mode have so far been inconclusive.

One also can specify when to stop (i.e., at the first complete parse, or to wait until all other ambiguous parses have been discovered). The disambiguation routine (which is described as a part of the construction modes) defines which parse is 'best'. Further, one may specify a left-to-right or right-to-left mode of how to progress along the string.

b. Discovery Modes:

The starting point of building a relaxation parser is to specify what to do when an exact match is not made. If the parser is expecting one word and finds another it can look around the indicated place in the string to find

what it is looking for, or it can in certain other circumstances simply insert the expected word into the string. Thus, under discovery modes, there are various options: either the parser is allowed to attempt matches in out-of-sequence parts of the string, or not. And if not, or if no such match is found, the parser may or may not be allowed to make an insertion.

So in PEDAGLOT, there is an INSERT mode (and various restricted versions of it) and a 'where to look' mode which is used to control the degree to which the parser can try to find out-of-place matches. There are tags associated with these two specifications, the INSERT tag and the OMIT tag, which are associated with the parses involving insertions and omissions that contain the number of insertions made and the number of input symbols omitted in building the parse.

There is also a rearrangement mode. Thus, given certain constraints, the parser could be given 'The Bites Man Dog' and produce a parse for 'The Man Bites the Dog' since it would have found 'Man,' by temporarily omitting 'Bites,' but then it looks for and finds 'Bites' and finally, finding no second 'the,'the,' inserts one (or some other determiner because of the [DET] function]) and finds 'Dog. In a similar way it would try to produce a passive form (i.e., the Man Is Bitten By the Dog) but since this involves more insertions, etc. it would not be chosen.

These heuristics are controlled by recording numerical summary tags with each sub-parse that participate in, and are 'judged' by the disambiguation routines. Similar ideas are used by Lyon (1974).

c. Prediction Modes:

As Woods (1975) has pointed out, the extent to which a parser's prediction increases efficiency varies with the quality of the expected input. This fact affects greatly our discovery procedures, since, if insertions are to be made, one ought to be rather sure of one's predictions, or risk a combinatorial ex-

plosion. In PEDAGLOT, there is a programmable 'choice' function that controls predictions. Specifically, when the parser encounters a non-terminal symbol, that symbol is the left-hand side of various rules. An uncontrolled prediction (used by a canonical top-down parser) is to select each such rule as the expansion. Intuitively, however, people do not seem to do this. Instead, as in an ATN, they try one and only if that fails, go into the next. In PEDAGLOT, the choice of which rule to try can be defined as the result of the call to a 'choose' function (or it can be left uncontrolled). We have designed various approaches to such predictions (e.g., a limited key-word scan of the incoming string, and the use of 'language statistics' such as the set of rules which can generate the next symbol in the string as their left most symbol).

The prediction is currently made once for any given choice point; its outcomes are expected to be an ordered set of rules to try next.

d. Construction Modes:

The phase of parsing in which the parts of the parse tree and associated tag values are formed, is a place where most of the non-syntactic information (tags) about the string being parsed can come into play.

In the first place, new tags can be formed as functions of lower level parse tags through a process called melding. Thus, 'nonsense' can be discovered and pronoun references can sometimes be tied down. In the second place, it is a result of construction that ambiguity is discovered and dealt with.

Since these features of parsing deal primarily with semantics (and since, if anywhere, semantic representations of the string reside in the tags), most of the PEDAGLOT construction modes involve tags.

One may explicitly meld tag values by using post-recognition operators, or one may define an 'implicit' melding routine that is associated with the tag

names themselves instead of with individual rules. In our example we use this device to implicitly form a simple list of the two REF tags that become associated with the S rule. This implicit melding operation can also include a blocking function, or some reference to a data base. The tags that contain INSERT and OMIT information are used in this way to keep running totals of, and to minimize the number of such heuristics in the relaxation parsing modes. One may also associate a LIFT function which, when the partial parse becomes complete, specifies a transformation of that tag to be used as the tag of the next higher level parse.

Ambiguity is discovered when two parses from the same symbol, covering the same string segment are found. For this case, an AMBIG function is associated with tag names, and it makes a 'value judgement' of which tag is 'better, hence which interpretation to use. (Other types of criteria can also come into play here such as user interaction, (cf. Kay, 1973).

3. The Uses of Meta-Parsers

I have just catalogued some of the parsing modes available in PEDAGLOT. Others, such as Bottom-Up (instead of Top-Down) or Inside-Out (instead of Left-to-Right, etc.), are envisioned but not implemented. Since PEDAGLOT is an interactive program, the user can change modes at will, just as he can change syntax or introduce new tags. Thus, the obvious first use of meta-parsers is that one may use them to design language processors without having to tie oneself down from the start to say, a depth-first parser.

Meta-parsers also have a certain amount of tractibility that parsers that blend all activities into one huge network may not. One may see at a rather high level what is going to be happening (i.e., all tags of a certain name will meld together in a certain way, unless the grammar specifies otherwise). If one, however, wants certain forms of local behavior, one may use predicates or functions on individual rules. Further, if one wants to change the order in which predictions are evaluated,

one can program a 'choose' function which will make that global change. To a large extent, the language designer may specify much of the processor in broad terms and still be able to control local events where necessary.

In a more general sense, a meta-parser allows one to understand and build higher order theories about how people might represent and process language.

For instance, while it may be true that generating is the inverse of parsing, there is more than one way to do such inverting. One could start from a semantic network, using the choose function along with the INSERT mode to restrict means of expression consistent with the intended message, and using AMBIG functions to weed out all but reasonable messages from among the many the parser may produce or one might simply take from the semantic network a simple string of meaningful words, and then use a less tightly programmed 'relaxation parser' to rearrange these words to be syntactically correct. We are now considering using a crude 'backwards' mode which begins with the operation part of a rule and, by using predicates (e.g., AGREE) to yield inverses, specifies what the context-free pattern must produce. Thus there are many variations of how to generate using a meta-parser.

In the area of language inference, to take another example of language processing, PEDAGLOT suggests various differing ways of approaching the problem. First, one may use it as a 'relaxation-parser,' the 'parse tree' can be pattern-matched against the new sentence, and hypotheses can be formed. Or, one could place a more rudimentary inference system on the 'prediction' part of the processor itself, and using other controls, the predictions that are successful could be rewritten as a new grammar. These two learning paradigms could each be strengthened by way of the use of tags to contain (in a sense) the meaning of the sentences to be learned. Each of these paradigms can be modeled using a meta-parser like PEDAGLOT. Thus, a meta-parser can raise (and be prepared to answer) a number of interesting questions.

References

Earley, J. (1970), "An Efficient Context-Free Parsing Algorithm," Comm. ACM 13, number 2, (February 1970), pp. 94-102.

Fabens, W. (1972), PEDAGLOT Users Manual, Rutgers University CBM-TR-12, Oct. 1972.

Fabens, W. (1973), PEDAGLOT Users Manual: Part II, Rutgers University CBM-TR-23, Nov. 1973.

Kaplan, R.M. (1973), "A General Syntactic Processor," in R. Rustin (ed.) Natural Language Processing, New York: Algorithmics Press, (1973), pp. 193-242.

Kay, M. (1973), "The MIND System," in R. Rustin (ed.) Natural Language Processing, New York: Algorithmics Press, (1973), pp. 155-188.

Lyon, G. (1974), "Syntax-Directed Least-Errors Analysis for Context-Free Languages: A Practical Approach." Comm. ACM 17, number 1, (January 1974), pp. 3-13.

Simmons, R. and Slocum, J. (1972), "Generating English Discourse from Semantic Networks." Comm. ACM 15, number 10, (October 1972), pp. 891-905.

Woods, W.A. (1970), "Transition Network Grammars for Natural Language Analysis." Comm. ACM 13, number 10, (October 1970), pp. 591-606.

Woods, W.A., (1975), Syntax, Semantics, and Speech, BBN Report No. 3067, A.I. Report No. 27. Bolt Beranek and Newman Inc., to appear in D.R. Reddy (ed.) Speech Recognition, Academic Press (1975).