DOCUMENT RESUME

ED 107 151 FL 006 923

AUTHOR Herman, L. Russell, Jr.
TITLE Detecting Syntactic Ambiguity: Three Augmented Transition Network Techniques.
PUB DATE 21 Mar 75
NOTE 21p.; Paper presented at the Southeastern Conference on Linguistics (SECOL) (13th, Vanderbilt University, March 1975)

EDRS PRICE MF-$0.76 HC-$1.58 PLUS POSTAGE
DESCRIPTORS *Ambiguity; Artificial Intelligence; *Computational Linguistics; *Computer Programs; Computer Science; *Information Processing; Programing Languages; Semantics; Sentence Structure; Structural Analysis; *Syntax

ABSTRACT

When a grammar is expressed in augmented transition network (ATN) form, the problem of detecting syntactic ambuguity reduces to finding all possible paths through the ATNs. Each successfully terminating path through the ATN generates an acceptable parsing of the input string. Two ATN forms, minimal-node and pseudo-tree, are described along with the conventions for traversing each. The two forms are compared in regard to efficient use of computer time and space and in regard to appropriateness for each of the three path-finding techniques. Three techniques are discussed for finding all acceptable paths through ATNs. The techniques are "Backtracking," "Simultaneous Parallel Analysis," and "Amputate And Re-enter." Relative merits of the three techniques are discussed in terms of computer execution time, required data storage, programmer time, and amenability of the program to modification. A rudimentary ATN-based parser for English has been written in SPITBOL to test the implementation of these techniques. (Author)

ERIC

ED107151

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Detecting Syntactic Ambiguity: Three Augmented Transition Network Techniques

L. Russell Herman, Jr.

North Carolina State University

The problem of computer use of natural human language may be thought of as consisting of three sub-tasks. The first sub-task is to devise programs which accept as input well-formed expressions in a natural human language and extract from the expression the same meaning as would a native speaker of the language. The second sub-task is to devise appropriate data structures for storing the meaning extracted from a natural language expression. The third sub-task may be thought of as the inverse of the first. It is to take information stored in memory and produce well-formed natural language expressions which will be correctly understood by a native speaker.

The scope of this paper is limited to discussing some techniques useful in the first sub-task. We will first review the use of augmented transition networks as parsers for deriving the underlying syntactic structure of a natural language expression. Two paradigmic forms for ATNs will be compared briefly. With that background, we will then present three techniques for handling expressions containing ambiguity in their syntactic structure. The two ATN forms will be compared in their appropriateness for each of the three techniques and the relative merits of the three techniques will also be discussed.

TRANSITION NETWORKS AS PARSERS

We need some formal conventions for describing the grammar we use in our parser. Augmented transition networks (ATNs) will serve

FL006 723

this purpose admirably.

Simple phrase structure grammars may be represented by finite-state transition networks such as the example in figure 1.

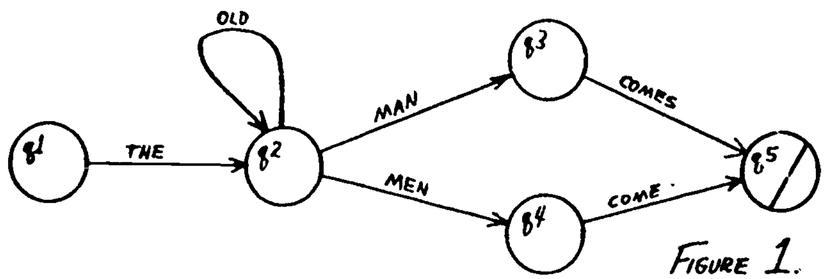


FIGURE 1.

A finite-state transition network consists of a finite set of nodes (represented in the diagram by circles) and a finite set of labeled, directed arcs (represented by arrows) connecting the nodes. A transition may be made from a node at the tail of an arc to the node at the arc's head only if the current input symbol of the string being parsed is in agreement with the label of the arc. When an arc is successfully traversed, the input symbol pointer is advanced to the next symbol of the input string.

The usual convention in drawing transition networks is to place the entrance node on the left and the terminating node(s) on the right with a diagonal slash. In seeking an arc out of a node, emanating arcs are attempted in a clockwise order beginning at the point of entry for that node. An input string is acceptable if and only if a terminating point in the network is reached and the input symbol pointer has reached the end of the string.

The grammar represented by the transition network in figure 1 will accept an infinite set of English sentences of the form:

the men come
the man comes
the old men come
the old man comes
the old old old men come
the old old old man comes
etc.

Though this set is infinite, it is not a very interesting subset of English. Indeed, it has been shown (Chomsky, 1957) that no mere finite-state grammar will suffice as a grammar of English.

We can, however, increase the power of our transition networks by allowing recursive calls both within and between networks. We achieve recursion by allowing our network to transfer control ("push down") to other networks or another point in the same network. When a "push" occurs, the location of the arc in the calling network is placed on a push-down stack and control is transfered to the called network. When a called network terminates (whether successful or unsuccessful), a "pop" occurs and control is transfered to the location on top of the stack.

If the sub-network called by the arc was traversed successfully, then control is transfered to the node at the head of that arc. Note that the input symbol pointer is not advanced for arcs which call for a "push" since the pointer will have been advanced by the successful traversal of the sub-network. If the sub-network called by the arc was not traversed successfully, then the input symbol pointer will remain at the position it occupied at the beginning of the calling arc and the next arc emanating from the node will be attempted.

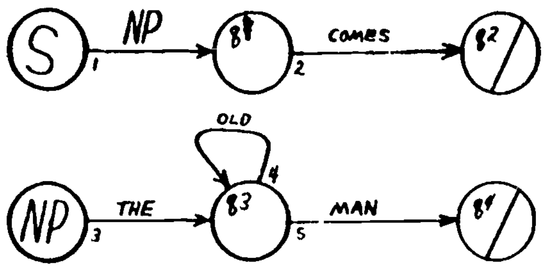


FIGURE 2.

The recursive transition network of figure 2 recognizes a portion of the sentences accepted by the finite-state network of figure 1. The network is entered at node S with the input symbol pointer at the beginning of the sentence to be parsed (for instance, "the old man comes"). We use the convention of upper-case letters to represent calls of sub-networks. There is only one arc leaving node S so we attempt to traverse it. The label on arc 1 invokes sub-network NP. Our current location (arc 1) is placed on the stack and control is transfered to node NP. At node NP, arc 3 is attempted. The word following the current location of the input symbol pointer is indeed "the" so arc 3 is traversed successfully, the input symbol pointer is advanced, and control is transfered to node q3.

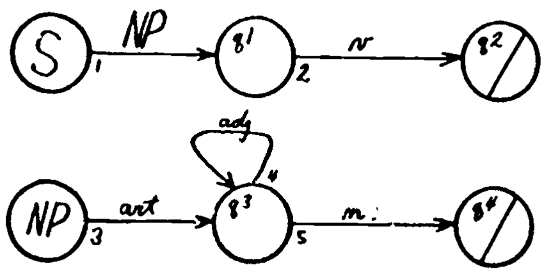
The first arc leaving node q3 is arc 4. The next word of the input string matches the word required by arc 4 ("old"), the input symbol pointer is advanced, and control is transfered to the node at the head of arc 4. The loop of arc 4 returns us to node q3. Attempting the first arc in clockwise order from the point of entry of node q3, we once again attempt arc 4. This time, however, the next word of the input string is "man" and arc 4 fails. The next arc leaving node q3 is arc 5 which accepts "man," advances the input pointer, and transfers control to node q4.

As is indicated by the diagonal slash through the circle, node q4 is a terminal node for the NP sub-network. Control is now transfered to the location stored on top of the stack, that is, arc 1. Since the NP network was traversed successfully, arc 1 is traversed successfully and control is transfered to node q1. When node q2 is reached, the input symbol pointer will be at the end of the input string and the control transfer stack will be empty. The parsing created by the successful traversal of the recursive transition network will be complete and the input string will have been found acceptable.

Even though by adding recursion to our network, we have increased its power from that of a finite-state recognizer to a context-free recognizer, the limitations of even context-free grammars create difficulties in handling English (Chomsky, 1957; Postal, 1964).

By augmenting our recursive transition networks with a set of conditions associated with each arc and a set of actions to be carried out if that arc is traversed, we will strengthen the network substantially. The actions may take the form of setting values of registers or variables and the conditions may depend for their success or failure on the values of registers or variables. Such an augmented transition network (ATN) will have the full power of a Turing machine and will be able to represent any grammar that could be parsed by any machine (Winograd, 1972, p.43).

A suitable ATN for our example set of sentences might be the one shown in figure 3.



arc	conditions	actions
1	none	set subject to NP parsed
2	no. of subject = no. of verb	set verb to word parsed
3,4,5	none	form noun phrase structure

FIGURE 3. A simple ATN with summarized augmentations.

With the lower-case labels of arcs 2, 3, 4, and 5 we introduce the concept of syntactic categories of words. A syntactic category is a set of words which perform the same function in the syntax of a language. For instance, the English words "a," "an," and "the" all perform closely similar syntactic functions. We call them "articles" and abbreviate their category "art." All three are used only in limited contexts, usually before a noun or before a modifier preceeding a noun.

An arc with a syntactic category label can be successfully traversed when any word from that category is encountered. The vocabulary of the language is stored in a lexicon and the parser has the ability to look up the lexical entry for words in order to determine their syntactic category. Additional conditions can be imposed via the augmentations. For instance, in the ATN of figure 3, we might impose the augmentation condition that the number of the article from arc 3 be in agreement with the number of the noun from arc 5. In this way, we would accept "the clowns" and reject "a clowns."

GENERATION OF UNDERLYING SYNTACTIC STRUCTURE VIA ATN PARSING

The sequence of transitions by which a string is parsed generates a structure in tree form which represents the underlying syntactic relations among the constituents of the surface string.

If the ATN of figure 3 is applied to the sentence "the merry clown pirouettes," the sequence of transition arcs traversed can be represented by the diagram of figure 4.

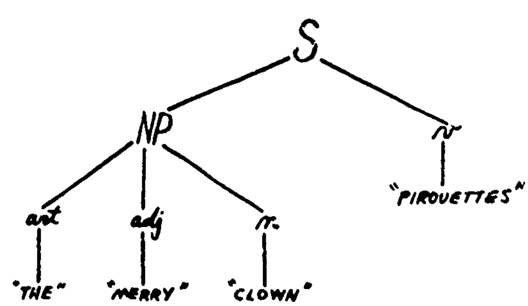


FIGURE 4. Underlying syntactic structure for "the merry clown pirouettes."

The ATN of figure 3 is constructed in such a way that its sub-networks represent significant syntactic constituents. For instance, any string parseable by the NP sub-network will qualify as a noun phrase. This tree structure of parse transitions is the underlying syntactic structure corresponding to the surface string.

Tree diagrams of syntactic structure are easy to read, but they require excessive space to draw and are not readily amenable to computer print-out. Frequently we will prefer to represent the syntactic structure in a nested parentheses form. Figure 4 would then become

(S(NP(art("the")adj("merry")n("clown"))v("pirouettes"))).

TWO FORMS FOR ATNs

In the article inaugurating ATNs in natural language parsing applications, W.A. Woods (1970, p.601) suggests using "standard finite state machine optimization techniques" to reduce repetition of arcs with the same label. We call this approach the "minimal-node" form. To quote Woods:

> Whenever the grammar contains two or more subgraphs of any size which are essentially copies of each other, it is a symptom of a regularity that is being missed. That is, there are two essentially identical parts of the grammar which differ only in that the finite state control part of the machine is remembering some piece of information, but otherwise the operation of the two parts of the graph are identical. To capture this generality, it is sufficient to explicitly store the distinguishing piece of information in a register (e.g. by a flag) and use only a single copy of the subgraph.
> (Woods, 1970, p.601)

The truth of this observation is obvious, but the advantage Woods imputed to it seems illusory. While the practice of minimalizing the ATNs will undoubtedly simplify the graphs themselves, it merely transfers the burden to the augmentations. In reducing the number of arcs and nodes, it increases the conditions which must be tested for the remaining arcs and multiplies the space required to store the registers.

We do not take minimalization nearly so far as Woods. Instead, we prefer a pseudo-tree form for our ATNs. In this form, when two or more arcs diverge from a node we do not allow them or any of their descendants to merge into common nodes. We say "pseudo-tree" instead of "tree" because we do allow an arc to loop back and re-enter the node from which it originated, thus violating the usual definition of a tree.

Contrary to the horizontal form which Woods uses to draw his minimal-node ATNs, we follow the usual tree conventions for our pseudo-tree ATNs. The starting node or origin of the ATN will be placed at the top with its arcs descending from it. At each node the exit arcs will be attempted from left to right (i.e., counter-clockwise from the point of entry).

In pseudo-tree form, the ATN will have only a few augmentations (conditions to be tested and actions to be implemented) per arc and will instead store such information in the structure of the graph itself. Since this information usually refers to the syntactic structure of the input string, it seems appropriate (aesthetically, if not also logically) to make it explicit in the structural component of our grammar, that is, the ATN graph.

SYNTACTIC AMBIGUITY IN ATNs

We say that a sentence is ambiguous in its syntactic structure if it is possible to find two or more syntactic structures for it such that each word has the same syntactic category assignment and the same meaning in every parse.

Consider the string "put the blue book on the red book on the shelf." The two ambiguous interpretations of this string are diagrammed in figure 5.

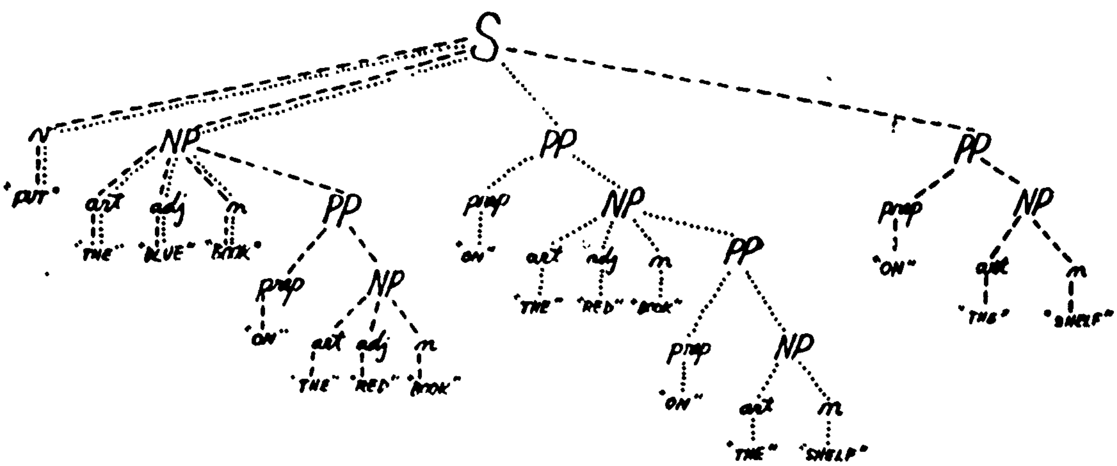


FIGURE 5. Two possible parsings for "put the blue book on the red book on the shelf." Parse I (----) might be paraphrased as "the blue book that is on the red book is to be put on the shelf." Parse II (.........) might be paraphrased as "the blue book is to be put on the red book on the shelf."

These two parsings are obtained by taking two different paths through the ATN. For parse I, we push down to PP during the first pass through NP thereby interpreting "on the red book" as an adjectival prepositional phrase and then pop up to S where we push down to PP, interpreting "on the shelf" as an adverbial phrase. For interpretation II, we do not push down to the PP network during the first pass through the NP network and so "on the red book on the shelf" becomes an adverbial prepositional phrase.

Note that our definition of syntactic ambiguity excludes the linguistically interesting form of ambiguity which results when a word has multiple meanings. Consider the sentence "the pitcher hit the umpire." This string is ambiguous because "pitcher" may be interpreted as either denoting the ball-player who stands on the mound and throws balls across the plate or denoting an utensil from which drink is poured at the dining table (as in "a pitcher of beer"). This form of ambiguity is excluded by our restriction of syntactic ambiguity to those forms in which ambiguity occurs without varying the meaning of the words.

When the grammar of the parser is represented by ATNs the problem of finding all interpretations of a syntactically ambiguous string reduces to a simple and direct task. One merely finds all paths through the ATN which accept the string without requiring variation of the lexical data of any of its words.

THREE ATN METHODS FOR PARSING SYNTACTICALLY AMBIGUOUS STRINGS

There are at least three methods available by which an ATN-based parser may detect syntactic ambiguity and, furthermore, generate all possible parsings (i.e., all possible syntactic interpretations) for the string. These methods are Simultaneous Parallel Analysis (SPA), Backtracking (BT), and Amputate And Re-enter (A&R).

If we merely wish to test for the existence of syntactic ambiguity, we attempt all possible paths until all paths are tested and only one is found acceptable (no ambiguity) or two acceptable paths are found (indicating ambiguity). Alternately, if we wish to generate all possible interpretations we do not stop with merely two paths but continue until all paths are attempted.

1. Simultaneous Parallel Analysis

In the Simultaneous Parallel Analysis (SPA) technique when the parse arrives at a node from which two or more arcs diverge, a copy of the previously formed portion of the underlying structure and all other information stored in registers is made for each arc and both routes are attempted simultaneously.

When implemented, the simultaneity of SPA is usually simulated by sequentially executing one operation of each of the "simultaneous" parsings. For this reason, SPA is sometimes referred to as "Parallel Partial Analysis" or "Simultaneous Partial Analysis." When an arc is not traversed successfully, the registers for that path and any underlying structure components that had been constructed are wiped from memory.

Note that while SPA is an effective technique for parsing syntactic ambiguities, it requires duplication of storage since an independent copy is needed for each active path. Though many paths will drop from consideration soon after they are begun, the combinatorial explosion of the worst case would impose a relatively enormous peak demand on memory space. As mentioned earlier, the register storage requirement will be slightly less for pseudo-tree ATNs than for minimal-node forms since the structural relationships of the grammar are stored in the structure of the ATN graph. In many cases, however, this advantage of the pseudo-tree form would be slight compared to the worst case peak demand of the SPA method.

The space requirements for SPA are larger than for either of the other two techniques. However, if truly simultaneous processing were available, SPA might well be faster than the other two techniques since its time would be equal to the time required for the slowest path, rather than the sum of the times for all paths (as is the case for A&R). In practice, however, this advantage is reduced by the requirements to execute each path one step at a time and swap back and forth among copies of memory. In this form, the time for SPA is the sum of the traversal time for each arc plus the time for swapping memory.

2. Backtracking

Contrary to the breadth-first method of SPA, the technique of Backtracking (BT) is a depth-first method. The parser first traverses the ATN until an acceptable termination is reached. The syntactic structure for this parse is stored. Then, beginning from the terminating end of the path, the parser backtracks up the path (traversing the arcs in reverse, as it were) until a node is reached from which diverges an arc that was not attempted on the first parsing. The registers are reset to the values they had previously at this node and parsing begins anew. The parser switches back to a forward (downward, for pseudo-tree ATNs) direction and attempts to find a new path in the usual way.

Note that a set of register values must be stored for each node in order to reset memory when the parser backtracks to that node. Obviously, nodes with mutually exclusive exit arcs cannot qualify as origins for new ambiguous parsings after backtracking. This is because to backtrack to an arc one must have descended from it successfully and cannot therefore successfully traverse any of the other mutually exclusive arcs emanating from it. For nodes of this sort, it is not necessary to preserve a copy of the register values.

The backtracking mechanism itself is much simpler to implement for pseudo-tree ATNs than for minimal-node since most nodes have at most one immediate ancestor. The exceptions are nodes with arcs which loop back to that node. An effective though simple-minded method is to use the so-called "Dewey Decimal notation" (Knuth, 1973) to number the arcs of the tree. Then, from the number of the current arc, it is a simple matter to derive the number of the parent. The insertion of pre-determined heuristics can often generate more efficient backtracking at the cost of a more complicated control structure.

The time required to develop all possible parsings using BT will be the sum of the traversal time for each arc plus the time to compute the backtracking destination plus the time to reset the registers for the nodes backtracked to.

3. Amputate And Re-enter

The third method available to test for syntactic ambiguity and to find all possible parsings is Amputate And Re-enter (A&R). A&R, like BT, is a depth-first method. Unlike SPA and BT, A&R does not require that copies be stored of registers nor does it require the computation of backtracking destinations.

In the A&R method, when the parser in traversing the ATN encounters a node whose descendants provide a possibility for syntactic ambiguity, an amputation flag is set on the exit arc traversed from that node. When the path terminates successfully, the syntactic structure for that interpretation is stored. The parser then resets the input symbol pointer to the beginning of the string, re-initializes all other registers except the amputation flags, and re-enters the ATN at the beginning.

The parser will, of course, retrace the same path through the ATN until it encounters an arc with an amputation flag. The amputation flag does not allow the parser to traverse that arc and so the next arc leaving that node is attempted. As with the first pass through the ATN, if new nodes are encountered with the option of ambiguity new amputation flags are set. The parser continues to re-enter the ATN until it is no longer able to traverse a successful path. When that happens, all possible syntactic interpretations of the string have been found.

The advantage of A&R over BT or SPA is the simplicity of its control structure and its saving in storage space. This elegance of the control structure and saving in storage are, however, paid for with a decrease in speed. In theory (i.e., discounting time for swapping memory and computing backtracking) A&R is slower than either SPA or BT. The time for developing all possible parsings using A&R will be the sum of the time for each path from the origin to its termination. Obviously, A&R will traverse some of the early arcs repeatedly. It is this repetition that increases the time required for A&R.

While A&R is theoretically slower than BT or SPA, it clearly requires much less memory space than either of those methods. There is no need in A&R to store an independent copy of memory for every active branch in the parsing tree (as is the case for SPA). Nor is there a need to store a copy of memory for every node with an option for ambiguity (as is the case for BT).

The time-space tradeoff is not the only difference to consider. With a decline in the relative cost of computer time and space, we have seen an increasing emphasis on minimizing programming time (not only in creating the program but also in subsequent modification). The A&R control structure has this advantage of programming simplicity. There is no need to constantly switch back and forth among several active paths (SPA), nor to repeatedly swap copies of memory (SPA), nor to compute backtracking destinations (BT). With A&R, the control structure merely re-initializes all registers except amputation flags and jumps back to the ATN entrance node. A&R's advantage of simplicity is relatively small for simple ATNs (e.g., the illustrative examples used above). However, as the ATNs become more complex, A&R's advantage of simplicity far outstrips SPA or BT.

IMPLEMENTATION

An experimental parser, FROTH, based on ideas presented here has been implemented in SNOBOL4 / SPITBOL (Griswold, 1973; Tharp, 1974) on the Triangle Universities Computation Center's IBM 370/165 system. Since FROTH was merely constructed to test the implementation of A&R on pseudo-tree ATNs, it does not attempt to serve as a general-purpose parser for English. The program is deliberately limited to a few of the more simple syntactic structures of English.

The grammar for FROTH is built into the code of the program in the form of a pseudo-tree ATN. For this reason, FROTH is a "syntax-oriented" parser rather than being "syntax-directed." In other words, the program is identical to the grammar rather than using an external grammar. Froth is oriented to the particular built-in syntax rather than being directed by whatever syntax is provided to it as input.

This syntax-oriented aspect of FROTH is not an innate characteristic of ATN parsers. Instead, it represents a programming decision resulting from the experimental nature of FROTH. In fact, we suspect that pseudo-tree ATNs may well be a naturally appropriate format for input of grammars to a syntax-directed parser.

Within its limited domain, FROTH rejects syntactically ill-formed strings and successfully uses A&R to detect syntactic ambiguity by generating two parsings of syntactically ambiguous sentences. For example, in response to the input sentence "the student obeyed the queen who made the laws on the sidewalk," FROTH returns two syntactic structures. The first interprets "on the sidewalk" as describing which laws the queen made. The second construes "on the sidewalk" as an adverbial phrase telling where the queen made the laws. If FROTH had been programmed to produce all possible interpretations, it would have returned a third syntactic structure with "on the sidewalk" as an adverbial phrase telling where the student's obedience occurred.

SUMMARY AND COMMENT

We have reviewed here the well-known application of ATNs to parsing natural language. In examining the two pre-dominant forms of ATNs, we found that while minimal-node simplifies the ATN graph, it merely transfers the burden to the augmentations. Pseudo-tree form, however, stores the syntactic structure of the grammar explicitly in the structure of the ATN graph and lightens the burden on the augmentations.

When three methods (SPA, BT, A&R) for parsing syntactically ambiguous strings were compared, SPA was found to be the fastest (in its theoretical form) and to require the most storage. A&R minimized storage and was slowest (in the worst case form). BT has time and space requirements intermediate between the extemes of the other two methods.

A&R has the further advantage of an extremely simple control structure. This advantage increases with increasing complexity of the ATNs. While at present the author prefers A&R over BT or SPA, it would be unwise to make a blanket prescription. The choice should be determined by the constraints of the particular application. One might find advantage in combining two or more methods. For instance, if the ATN had no possibility for ambiguity in its early nodes, one might choose to backtrack to the first ambiguous node and then use A&R from there.

ACKNOWLEDGMENTS

FROTH and the associated research presented here began as a project for a course in computational linguistics taught during the fall semester of 1974 by Dr. Alan L. Tharp of the Computer Science Department, North Carolina State University. Without Dr. Tharp's patient advice and counsel and a grant of computer time from the Computer Science Department, the project would not have been completed. I am grateful to Nancy Nunnery for clarification of my thoughts as a result of her probing questions. These and other friends and colleagues provided sustaining encouragement.

REFERENCES

Chomsky, N., Syntactic Structures, Mouton, The Hague, 1957.

Griswold, R.E. and Griswold, M.T., A SNOBOL4 Primer, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

Knuth, D.E., The Art of Computer Programming vol.1 Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, second edition, 1973.

Postal, P., "Limitations of Phrase Structure Grammars," in The Structure of Language, Katz, J. and Fodor, J. (eds.), Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

Tharp, A.L., (ed.), SPITBOL: A Guide To Its Use, Lexicon Press, Raleigh, North Carolina, 1974.

Winograd, T., Understanding Natural Language, Academic Press, New York, 1972.

Woods, W.A., "Transition Network Grammars for Natural Language Analysis," Communications of the ACM, vol.13, no.10, October, 1970.