TRANSFORMING SYNTACTIC GRAPHS INTO SEMANTIC GRAPHS*

Hae-Chang Rim
Jungyun Seo
Robert F. Simmons

Department of Computer Sciences
and
Artificial Intelligence Laboratory
Taylor Hall 2.124, University of Texas at Austin,
Austin, Texas 78712

ABSTRACT

In this paper, we present a computational method for transforming a **syntactic graph**, which represents all syntactic interpretations of a sentence, into a **semantic graph** which filters out certain interpretations, but also incorporates any remaining ambiguities. We argue that the resulting ambiguous graph, supported by an exclusion matrix, is a useful data structure for question answering and other semantic processing. Our research is based on the principle that ambiguity is an inherent aspect of natural language communication.

INTRODUCTION

In computing meaning representations from natural language, ambiguities arise at each level. Some word sense ambiguities are resolved by syntax while others depend on the context of discourse. Sometimes, syntactic ambiguities are resolved during semantic processing, but often remain even through coherence analysis at the discourse level. Finally, after syntactic, semantic, and discourse processing, the resulting meaning structure may still have multiple interpretations. For example, a news item from Associated Press, November 22, 1989, quoted a rescued hostage,

> "The foreigners were taken to the Estado Mayor, army headquarters. I left that hotel about quarter to one, and by the time I got here in my room at quarter to 4 and turned on CNN, I saw myself on TV getting into the little tank," Blood said.

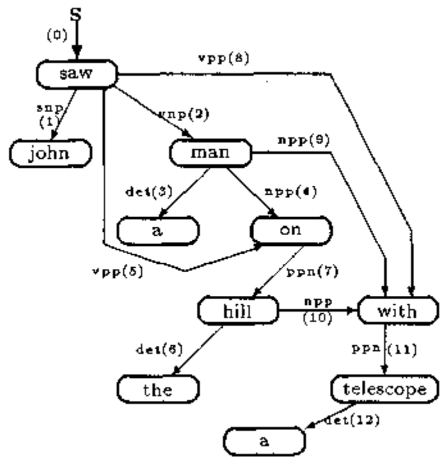
The article was datelined, Albuquerque N.M. A first reading suggested that Mr. Blood had been flown to Albuquerque, but further thought suggested that "here in my room" probably referred to some sleeping section in the army headquarters. But despite the guess, ambiguity remains.

In a previous paper [Seo and Simmons 1989] we argued that a syntactic graph — the union of all parse trees — was a superior representation for further semantic processing. It is a concise list of syntactically labeled triples, supported by an exclusion matrix to show what pairs of triples are incompatible. It is an easily accessible representation that provides succeeding semantic and discourse processes with complete information from the syntactic analysis. Here, we present methods for transforming the syntactic graph to a functional graph (one using syntactic functions, *SUBJECT, OBJECT, IOBJECT* etc.) and for transforming the functional graph to a semantic graph of case relations.

BACKGROUND

Most existing semantic processors for natural language systems (NLS) have depended on a strategy of selecting a single parse tree from a syntactic analysis component (actual or imagined). If semantic testing failed on that parse, the system would select another — backing up if using a top-down parser, or selecting another interpretation

*This work is sponsored by the Army Research Office under contract DAAG29-84-K-0060.



1 (SNP saw John)
2 (VNP saw man)
3 (DET man a)
4 (NPP man on)
5 (VPP saw on)
6 (DET hill the)
7 (PPN on hill)
8 (VPP saw with)
9 (NPP man with)
10 (NPP hill with)
11 (PPN with telescope)
12 (DET telescope a)

	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
1													
2													
3													
4						1							
5					1					1			
6													
7													
8										1	1		
9						1			1		1		
10									1	1			
11													
12													

Figure 1: Syntactic Graph and Exclusion Matrix for *"John saw a man on the hill with a telescope."*

from an all-paths chart. Awareness has grown in recent years that this strategy is not the best. Attempts by Marcus [1980] to use a deterministic (look-ahead) tactic to ensure a single parse without back-up, fail to account for common, garden-path sentences. In general, top-down parsers with backup have unpleasant implications for complexity, while efficient all-paths parsers limited to complexity $O(N^3)$ [Aho and Ullman 1972, Early 1970, Tomita 1985] can find all parse trees in little more time than a single one. If we adopt the economical parsing strategy of obtaining an all-paths parse, the question remains, how best to use the parsing information for subsequent processing.

Approaches by Barton and Berwick [1985] and Rich *et. al.* [1987] among others have suggested what Rich has called **ambiguity procrastination** in which a system provides multiple potential syntactic interpretations and postpones a choice until a higher level process provides sufficient information to make a decision. Syntactic representations in these systems are incomplete and may not always represent possible parses. Tomita [1985] suggested using a *shared-packed-forest* as an economical method to represent all and only the parses resulting from an all-paths analysis. Unfortunately, the resulting tree is difficult for a person to read, and must be accessed by complex programs. It was in this context that we [Seo and Simmons 1989] decided that a graph composed of the union of parse trees from an all-paths parser would form a superior representation for subsequent semantic processing.

SYNTACTIC GRAPHS

In the previous paper we argued that the syntactic graph supported by an exclusion matrix would provide all and "only" the information given by a parse forest.[1] Let us first review an example of a syntactic graph for the following sentence:

Ex1) John saw a man on the hill with a telescope.

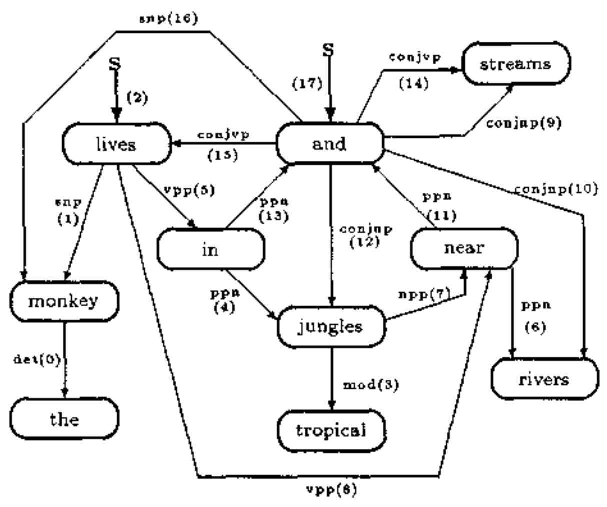
There are at least five syntactic interpretations for Ex1 from a phrase structure grammar. The syntactic graph is represented as a set of *dominator-modifier* triples[2] as shown in the middle of Figure 1 for Ex1. Each triple consists of a label, a head-word, and a modifier-word.

Each triple represents an arc in a syntactic graph in the left of Figure 1. An arc is drawn from the head-word to the modifier-word. The label of each triple, SNP,VNP, etc. is uniquely determined according to the grammar rule used to generate the triple. For example, a triple with the label SNP is generated by the grammar rule, $SNT \rightarrow NP + VP$, VPP is from the rule $VP \rightarrow VP + PP$, and PPN from $PP \rightarrow Prep + NP$, etc.

We can notice that the ambiguities in the graph are signalled by identical third terms (i.e., the same modifier-words with the same sentence position) in triples because a word cannot modify two different words in one syntactic interpretation. In

[1]We proved the "all" but have discovered that in certain cases to be shown later, the transformation to a semantic graph may result in arcs that do not occur in any complete analysis.

[2]Actually each word in the triples also includes notation for position, and syntactic class and features of the word.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0																		
1															1	1	1	1
2															1	1	1	1
3																		
4													1	1				
5																		
6											1	1						
7									1									
8								1					1	1				
9															1	1	1	1
10							1						1	1	1	1	1	1
11							1						1	1	1	1	1	1
12					1				1		1	1			1	1	1	1
13					1				1		1	1			1	1	1	1
14		1	1							1	1	1	1	1				
15		1	1							1	1	1	1	1				
16		1	1							1	1	1	1	1				
17		1	1							1	1	1	1	1				

Figure 2: Syntactic Graph and Exclusion Matrix for *"The monkey lives in tropical jungles near rivers and streams."*

a graph, each node with multiple in-arcs shows an ambiguous point. There is a special arc, called the **root arc**, which points to the head word of the sentence. The arc *(0)* of the syntactic graph in Figure 1 represents a root arc. A root arc contains information (not shown) about the modalities of the sentence such as voice: passive, active, mood: declarative or wh-question, etc. Notice that a sentence may have multiple root arcs because of syntactic ambiguities involving the head verb.

One interpretation can be obtained from a syntactic graph by picking up a set of triples with no repeated third terms. In this example, since there are two identical occurrences of *on* and three of *with*, there are $2*3 = 6$ possible sentence interpretations in the graph represented above. However, there must be only five interpretations for Ex1. The reason that we have more interpretations is that there are triples, called **exclusive triples**, which cannot co-occur in any syntactic interpretation. In this example, the triple **(vpp saw on)** and **(npp man with)** cannot co-occur since there is no such interpretation in this sentence.[3] That's why a syntactic graph must maintain an **exclusion matrix**.

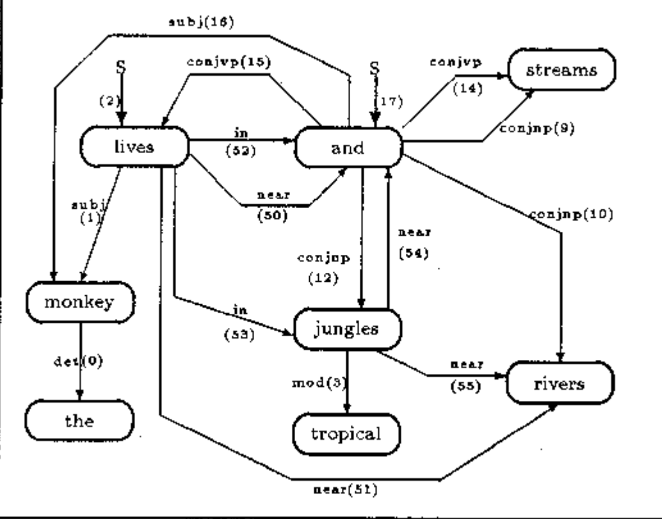
An exclusion matrix, (*Ematrix*), is an $N * N$ matrix where N is the number of triples. If $Ematrix(i, j) = 1$ then the i-th and j-th triple cannot co-occur in any reading. The exclusion matrix for Ex1 is shown in the right of Figure 1. In Ex1, the triples 5 and 9 cannot co-occur in any interpretation according to the matrix. Trivially exclusive triples which share the same third term are also marked in the matrix. It is very important to maintain the *Ematrix* because otherwise a syntactic graph generates more interpretations than actually result from the parsing grammar.

Syntactic graphs and the exclusion matrix are computed from the chart (or forest) formed by an all-paths chart parser. Grammar rules for the parse are in augmented phrase structure form, but are written to minimize their deviation from a pure context-free form, and thus, limit both the conceptual and computational complexity of the analysis system. Details of the graph form, the grammar, and the parser are given in (Seo and Simmons 1989).

COMPUTING SEMANTIC GRAPHS FROM SYNTACTIC GRAPHS

An important test of the utility of syntactic graphs is to demonstrate that they can be used directly to compute corresponding semantic graphs that represent the union of acceptable case analyses. Nothing would be gained, however, if we had to extract one reading at a time from the syntactic graph, transform it, and so accumulate the union of case analyses. But if we can apply a set of rules

[3]Once the phrase "on the hill" is attached to *saw*, "with a telescope" must be attached to either *hill* or *saw*, not *man*.



	0	1	2	3	9	10	12	14	15	16	17	50	51	52	53	54	55
0																	
1								1	1	1	1						
2								1	1	1	1						
3																	
9								1	1	1	1						
10							1	1	1	1	1		1	1			1
12						1		1	1	1	1	1	1		1	1	
14		1	1		1	1	1					1		1		1	
15		1	1		1	1	1					1		1		1	
16		1	1		1	1	1					1		1		1	
17		1	1		1	1	1					1		1		1	
50							1	1	1	1	1		1	1		1	1
51						1	1					1		1		1	1
52						1		1	1	1	1	1	1		1	1	
53							1							1			
54							1	1	1	1	1	1	1	1			1
55						1						1	1			1	

Figure 3: Functional Graph and Exclusion Matrix for *"The monkey lives in tropical jungles near rivers and streams."*

directly to the syntactic graph, mapping it into the semantic graph, then using the graph can result in a significant economy of computation.

We compute a semantic graph in a two-step process. First, we transform the labeled dependency triples resulting from the parse into functional notation, using labels such as *subject, object,* etc. and transforming to the canonical *active* voice. This results in a **functional graph** as shown in Figure 3. Second, the functional graph is transformed into the semantic graph of Figure 5. During the second transformation, filtering rules are applied to reduce the possible syntactic interpretations to those that are semantically plausible.

COMPUTING FUNCTIONAL GRAPHS

To determine SUB, OBJ and IOBJ correctly, the process checks the types of verbs in a sentence and its voice, active or passive. In this process, a syntactic triple is transformed into a functional triple: for example, **(snp X Y)** is transformed into **(subj X Y)** in an active sentence.

However, some transformation rules map several syntactic triples into one functional triple. For example, in a passive sentence, if three triples, **(voice X passive)**, **(vpp X by)**, and **(ppn by Y)**, are in a syntactic graph and they are not exclusive with each other, the process produces one functional triple **(subj X Y)**. Since prepositions are used as functional relation names, two syntactic triples for a prepositional phrase are also reduced into one functional triple. For example, **(vpp lives in)** and **(ppn in jungles)** are transformed into **(in lives jungles)**. These transformations are represented in Prolog rules based on general inference forms such as the following:

(stype X declarative) & **(voice X passive)** & **(vpp X by)** & **(ppn by Y)** ⇒ **(subject X Y)**

(vpp X P) & **(ppn P Y)** & **not(voice X passive)** ⇒ **(P X Y)**.

When the left side of a rule is satisfied by a set of triples from the graph, the exclusion matrix is consulted to ensure that those triples can all co-occur with each other.

This step of transformation is fairly straightfoward and does not resolve any syntactic ambiguities. Therefore, the process must carefully transform the exclusion matrix of the syntactic graph into the exclusion matrix of the functional graph so that the transformed functional graph has the same interpretations as the syntactic graph has[4].

Intuitively, if a functional triple, say F, is produced from a syntactic triple, say T, then F must be exclusive with any functional triples produced from the syntactic triples which are exclusive with T. When more than one syntactic triple, say $T_i's$ are involved in producing one functional triple, say F_1, the process marks the exclusion

[4]At a late stage in our research we noticed that we could have written our grammar to result directly in syntactic-functional notation; but one consequence would be increasing the complexity of our grammar rules, requiring frequent tests and transformations, thus increasing conceptual and computational complexities.

```
N : the implausible triple which will be removed.
The process starts by calling remove-all-Dependent-arcs([N]).

remove-all-dependent-arcs(Arcs-to-be-removed)
    for all Arc in Arcs-to-be-removed do
    begin
        if Arc is not removed yet
            then
                find all arcs pointing to the same node as Arc; call them Alt-arcs
                find arcs which are exclusive with every arc in Alt-arcs, call them Dependent-arcs
                remove Arc
                remove entry of Arc from the exclusion matrix
                remove-all-Dependent-arcs(Dependent-arcs)
    end
```

Figure 4: Algorithm for Finding Dependent Relations

matrix so that F_1 can be exclusive with all functional triples which are produced from the syntactic triples which are exclusive with any of T_i'*s*.

The syntactic graph in Figure 2 has five possible syntactic interpretations and all and only the five syntactic-functional interpretations must be contained in the transformed functional graph with the new exclusion matrix in Figure 3. Notice that, in the functional graph, there is no single, functional triple corresponding to the syntactic triples, *(4)-(8)*, *(11)* and *(13)*. Those syntactic triples are not used in one-to-one transformation of syntactic triples, but are involved in many-to-one transformations to produce the new functional triples, *(50)-(55)*, in the functional graph.

COMPUTING SEMANTIC GRAPHS

Once a functional graph is produced, it is transformed into a semantic graph. This transformation consists of the following two subtasks: given a functional triple (i.e., an arc in Figure 3), the process must be able to (1) check if there is a semantically meaningful relation for the triple (i.e., co-occurrence constraints test), (2) if the triple is semantically implausible, find and remove all functional triples which are dependent on that triple.

The co-occurrence constraints test is a matter of deciding whether a given functional triple is semantically plausible or not.[5] The process uses a type hierarchy for real world concepts and rules that state possible relations among them. These relations are in a case notation such as **agt** for *agent*, **ae** for *affected-entity*, etc. For example, the *subject(1)* arc between *lives* and *monkey* numbered *(1)* in Figure 3 is semantically plausible since **animal** can be an agent of **live** if the **animal** is a *subj* of the **live**. However, the *subject* arc between *and* and *monkey* numbered *(16)* in Figure 3 is semantically implausible, because the relation *conjvp* connects *and* and *streams*, and *monkey* can not be a subject of the verb *streams*. In our knowledge base, the legitimate *agent* of the verb *streams* is a **flow-thing** such as a river.

[5] Eventually we will incorporate more sophisticated tests as suggested by Hirst(1987) and others, but our current emphasis is on the procedures for transforming graphs.

When a given arc is determined to be semantically plausible, a proper case relation name is assigned to make an arc in the semantic graph. For example, a case relation *agt* is found in our knowledge base between *monkey* and *lives* under the constraint *subject*.

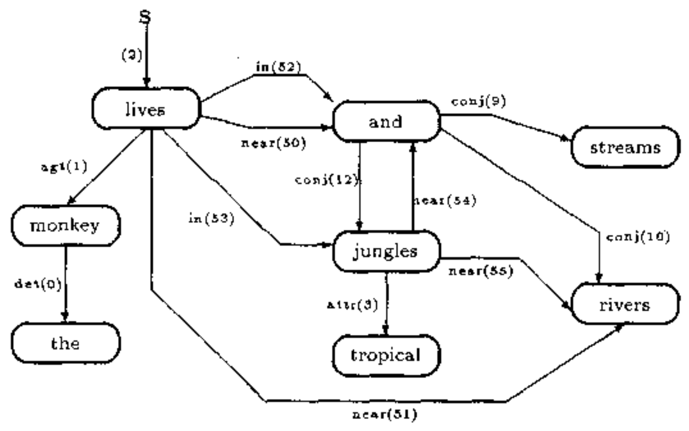
If a triple is determined to be semantically implausible, then the process removes the triple. Let us explain the following definition before discussing an interesting consequence.

Definition 1 *A triple, say T_1, is* **dependent on** *another triple, say T_2, if every interpretation which uses T_1 always uses T_2.*

Then, when a triple is removed, if there are any triples which are dependent on the removed triple, those triples must also be removed. Notice that the **dependent on** relation between triples is transitive.

Before presenting the algorithm to find dependent triples of a triple, we need to discuss the following property of a functional graph.

Property 1 *Each semantic interpretation derived from a functional graph must contain every node in each position once and only once.*



	0	1	2	3	9	10	12	50	51	52	53	54	55
0													
1													
2													
3													
9													
10							1		1	1			1
12						1		1	1		1	1	
50							1		1	1		1	1
51						1	1	1		1		1	1
52						1		1	1		1	1	
53							1			1			
54							1	1	1	1			1
55						1		1	1			1	

Figure 5: Semantic Graph and Exclusion Matrix for *"The monkey lives in tropical jungles near rivers and streams."*

Here the position means the position of a word in a sentence. This property ensures that all words in a sentence must be used in a semantic interpretation once and only once.

The next property follows from Property 1.

Property 2 *If a triple is determined to be semantically implausible, there must be at least one triple which shares the same modifier-word. Otherwise, the sentence is syntactically or semantically ill-formed.*

Lemma 1 *Assume that there are n triples, say $T_1, \ldots, T_n$, sharing a node, say N, as a modifier-word (i.e. third term) in a functional graph. If there is a triple, say T, which is exclusive with $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n$ and is not exclusive with T_i, T is dependent on T_i.*

This lemma is true because T cannot co-occur with any other triples which have the node N as a modifier-word except T_i in any interpretation. By Property 1, any interpretation which uses T must use one triple which has N as a modifier-word. Since there is only one triple, T_i that can co-occur with T, any interpretations which use T use T_i.□

Using the above lemma, we can find triples which are dependent on a semantically implausible triple directly from the functional graph and the corresponding exclusion matrix. An algorithm for finding a set of dependent relations is presented in Figure 4.

For example, in the functional graph in Figure 3, since *monkey* cannot be an **agt** of *streams*, the triple *(16)* is determined to be semantically implausible. Since there is only one triple, *(1)*, which shares the same modifier-word, *monkey*, the process finds triples which are exclusive with *(1)*. Those are triples numbered *(14)*, *(15)*, *(16)*, and *(17)*. Since these triples are dependent on *(16)*, these triples must also be removed when *(16)* is removed. Similarly, when the process removes *(14)*, it must find and remove all dependent triples of *(14)*. In this way, the process cascades the remove operation by recursively determining the dependent triples of an implausible triple.

Notice that when one triple is removed, it removes possibly multiple ambiguous syntactic interpretations—two interpretations are removed by removing the triple *(16)* in this example, but for the sentence, *It is transmitted by eating shellfish such as oysters living in infected waters, or by drinking infected water, or by dirt from soiled fingers*, 189 out of 378 ambiguous syntactic interpretations are removed when the semantic relation (**mod water drinking**) is rejected.[6] This saves many operations which must be done in other approaches which check syntactic trees one by one to make a semantic structure. The resulting semantic graph and its exclusion matrix derived from the functional graph in Figure 3 have three semantic interpretations and are illustrated in Figure 5. This is a reduction from five syntactic interpretations as a result of filtering out the possibility, **(agt streams monkey)**.

There is one arc in Figure 5, labeled *near(51)*, that proved to be of considerable interest to us.

[6]In "infected drinking water", (**mod water drinking**) is plausible but not in "drinking infected water".

If we attempt to generate a complete sentence using that arc, we discover that we can only produce, "The monkey lives in tropical jungles near rivers." There is no way that that a generation with that arc can include "and streams" and no sentence with "and streams" can use that arc. The arc, *near(51)*, shows a failure in our ability to rewrite the exclusion matrix correctly when we removed the interpretation "the monkey lives ... and streams." There was a possibility of the sentence, "the monkey lives in jungles, (lives) near rivers, and (he) streams." The redundant arc was not dependent on *subj(16)* (in Figure 3) and thus remains in the semantic graph. The immediate consequence is simply a redundant arc that will not do harm; the implication is that the exclusion matrix cannot filter certain arcs that are indirectly dependent on certain forbidden interpretations.

DISCUSSION AND CONCLUSION

The utility of the resultant semantic graph can be appreciated by close study of Figure 5. The graph directly answers the following questions, (assuming they have been parsed into case notation):

- Where does the monkey live?
  1. in tropical jungles near rivers and streams,
  2. near rivers and streams,
  3. in tropical jungles near rivers,
  4. in tropical jungles.
- Does the monkey live in jungles? Yes, by *agt(1)* and *in(53)* which are not exclusive with each other.
- Does the monkey live in rivers? No, because *in(52)* is exclusive with *conj(10)*, and *in(53)* is pointing to *jungles* not *rivers*.
- Does the monkey live near jungles? No, because *near(50)* and *conj(12)* are exclusive, so no path from *live* through *near(50)* can go through *conj(12)* to reach *jungle*, and the other path from *live* through *near(51)* goes to *rivers* which has no exiting path to *jungle*.

Thus, by matching paths from the question through the graph, and ensuring that no arc in the answering path is forbidden to co-occur with any other, questions can be answered directly from the graph.

In conclusion, we have presented a computational method for directly computing semantic graphs from syntactic graphs. The most crucial and economical aspect of the computation is the capability of applying tests and transformations directly to the graph rather than applying the rules to one interpretation, then another, and another, etc. When a semantic filtering rule rejects one implausible relation, then pruning all dependent relations of that relation directly from the syntactic graph has the effect of excluding substantially many syntactic interpretations from further consideration. An algorithm for finding such dependent relations is presented.

In this paper, we did not consider the multiple word senses which may cause more semantic ambiguities than we have illustrated. Incorporating and minimizing word sense ambiguities is part of our continuing research. We are also currently investigating how to integrate semantic graphs of previous sentences with the current one, to maintain a continuous context whose ambiguity is successively reduced by additional incoming sentences.

References

[1] Alfred V. Aho, and Jeffrey D. Ullman, *The Theory of Parsing, Translation and Compiling*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[2] G. Edward Barton and Robert C. Berwick, "Parsing with Assertion Sets and Information Monotonicity," *Proceedings of IJCAI-85*: 769-771, 1985.

[3] Jay Early, "An Efficient Context-free Parsing algorithm," *Communications of the ACM*, Vol. 13, No. 2: 94-102, 1970.

[4] Graeme Hirst, *Semantic Interpretation and the Resolution of Ambiguity*, Cambridge University Press, Cambridge, 1987.

[5] Mitchell P. Marcus, *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, 1980.

[6] Elain Rich, Jim Barnett, Kent Wittenburg and David Wroblewski, "Ambiguity Procrastination," *Proceedings of AAAI-87*: 571-576, 1987.

[7] Jungyun Seo and Robert F. Simmons, "Syntactic Graphs: A Representation for the Union of All Ambiguous Parse Trees," *Computational Linguistics*, Vol. 15, No. 1: 19-32, 1989.

[8] Masaru Tomita, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, 1985.