SYNTACTIC PREFERENCES FOR ROBUST PARSING WITH SEMANTIC PREFERENCES

JIN WANG
Computing Research Laboratory
New Mexico State University
Las Cruces, NM 88003
E-mail: wjin@nmsu.edu

Abstract

Using constraints in robust parsing seems to have what we call "robust parsing paradox". Preference Semantics and Connectionism both offered a promising approach to this problem. However, Preference Semantics has not addressed the problem of how to make full use of syntactic constraints, and Connectionism has some inherent difficulties of its own which prevent it producing a practical system. In this paper we are proposing a method to add syntactic preferences to the Preference Semantics paradigm while maintaining its fundamental philosophy. It will be shown that syntactic preferences can be coded as a set of weights associated with the set of symbolically manipulatable rules of a new grammar formalism. The syntactic preferences such coded can be easily used to compute with semantic preferences. With the help of some techniques borrowed from Connectionism, these weights can be adjusted through training.

1. Introduction

Robust parsing faces what seems to be a "paradox". On one hand, it needs to tolerate input sentences which more or less break syntactic and semantic constraints; that is, given an ill-formed sentence, the parser should attempt to interpret it somehow rather than simply reject it because it violates some constraints. In order to do this it allows syntactic and semantic constraints to be relaxed. On the other hand, robust parsing still need to be able to disambiguate. For the sake of disambiguation, stronger syntactic and semantic constraints are required, since the stronger these constraints are, the greater the number of unlikely readings that can be rejected, and the more likely the correct reading is selected.

A lot of work related to this problem has been done. The most commonly used strategy is to use the strongest constraints first to parse the input. If the parser fails, the violated constraints are relaxed in order to recover the failed process and arrive at a successful parse (Carbonell & Hayes, 1983) (Huang, 1988) (Kwasny & Sondheimer, 1981) (Weischedel & Sondheimer, 1983). The major problem with this approach is that it is difficult to tell which constraint is actually being violated and, therefore, needs to be relaxed when the parser fails. This problem is more serious with the parser using backtracking.

Another strategy is based on Preference Semantics (Wilks, 1975, 1978) (Fass & Wilks, 1983). The idea is that all the possible sentence readings can (though not necessarily) be produced. All the "readings are scored according to how many preference satisfactions they contain", and "the best reading (that is, the one with the most preference satisfactions) is taken, even if it contains preference violations." Selfridge (1986) and Slator (1988) also use this strategy. One important advantage of this approach is that within a uniform mechanism, the semantic constraints can both be used maximally for the sake of disambiguation and be gracefully relaxed when necessary for the sake of robustness. However, how to extend the preference philosophy to syntactic constraints have not been addressed. There are two frequently used approaches to incorporate syntactic constraints in systems using semantic preferences. The first ,as in (Slator, 1988), is to use a weak version of a rather typical syntactic module. The problem with this approach is that the syntactic constraints here still suffer from the problem of "robust parsing paradox". Another problem with this approach is that it shifts more burdens of disambiguation to semantic preferences because the syntactic constraints need to be weak enough in order not to reject too many noisy inputs. The second approach, as in (Selfridge, 1986), is to try all the possible combinations without any structural preference. The problem here is computational complexity especially with long sentences which have complex or recursive structures in them.

Connectionism has also shown some appealing results (Dyer, 1991) (Waltz & Pollack, 1985) (Lehnert, 1991) on robustness. However, there are some very difficult problems which they pose, such as the difficulty with complex structures (especially recursive structures), the difficulty with variables, variable bindings and unification, and the difficulty with schemes and their instantiations. These problems need to be solved before such an approach can be used in practical natural language processing systems

especially those which involve syntactic processing (Dyer, 1991).

In this paper we will propose a framework of representing syntactic preferences[1] that will keep all the virtues of robustness Preference Semantics suggests. Furthermore, such preferences can be learned.

2. Sketch of Our Approach

As we seen the idea of Preference Semantics about robustness is to enable the system to 1) generate (potentially) all the possible readings for all the possible inputs, 2) find out among all the possible readings which one is the most appropriate one and generate it first. To meet this goal for syntactic constraints we want to accomplish the following:

1. A formalism with a practically feasible amount of symbolic manipulatable rules which will have enough generative power to accept any input and produce all the possible structures of it (section 3.1).
2. A scheme to associate weights with each of these rules such that the weights of all the rules that are used to produce a structure will reflect how preferable (syntactically speaking) the whole structure is (section 2.1).
3. A algorithm that will incorporate these rules and weights with semantic preferences so that the overall best reading will be generated first. (section 2.2).
4. A method to train the weights (section 2.1 and 3.3).

2.1. Coding Syntactic Constraints

The most popular method to encode syntactic constraints is probably phrase structure grammar. One way to make it robust is to assign for all the grammar rules a cost. So, a rule designed for well formed inputs will have less cost than the rules designed for less well formed inputs; that is to say that a higher cost grammar rule will be less preferred than a lower cost grammar rule. However there are two serious problems with this naive method. First, if all the ill-formedness will have a set of special grammar rules designed for them, to capture a reasonable variety of ill-formedness, we need an unreasonably amount of grammar rules. Second, it is not clear how we can find the costs for all these rules. Next we will show how we will solve these two problems.

To solve the first problem, our approach abandons the use of phrase structure grammar, and instead a new grammar formalism is used. The idea of this new formalism is to reduce syntactic constructions to a relative small set of more primitive rules (P-rules), so that each syntactic construction can be produced by the application of a particular sequence of these P-rules. Section 3.1 will give more details about this formalism, and we will just list some of its relevant characteristics here: 1. There are no nonterminals in this formalism. 2. Each P-rule take only three parameters to specify a pattern on which its action will be fired. All these three parameters will be one of the parts of speech (or word types), such as noun, verb, adverb, etc. For example: (noun, verb, noun) ==> action3 is a P-rule. 3. Each rule has one action. There are only three possible actions.

Since the number of parts of speech is generally limited[2], the total rule space (the number of all the possible P-rules) is rather small[3]. So, it is possible for us to keep all of the possible P-rules in the grammar rule base of the parser. The output of the parser is a parsing tree in which each node corresponds to a word in the input sentence and the edge represents the dependency or the attachment relation between the words in the sentence. One important property of this formalism is that given any input sentence, for all of the normal parsing trees that can be built on it (the definition of normal parsing tree is given in next section), there exists a sequence of P-rules, such that the tree will be derived by applying this rule sequence. This means that we can now use this small P-rule set to guide the parsing[4] without pre-excluding any kinds of ill-formed inputs for which a normal parsing tree does exist.

---

[1] We use syntactic preferences in a broader sense which not only includes syntactic feature agreements but also the order of the constituents in the sentence and the way in which different constituents are combined. On the other hand, as you will see, the absence of nonterminals and syntactic derivation trees will also distance us from a strong syntax theory.

[2] For example in Longman Dictionary of Contemporary English (1978) there are only 14 different parts of speech. Given that *interjection* can be taken care by a preprocessor and the complex parts like *v adv, v adv prep, v adv;prep, v prep* can be represented as *v* plus some syntactic features, this number can be further reduced to 9.

[3] If there are 10 different parts of speech, the total P-rule space is smaller than 10x10x10x3 = 3000.

[4] Because of the searching algorithm we used, a lot of searching paths sanctioned by p-rules will be pruned by semantic preferences. In this sense the

To solve the second problem, we will use some techniques borrowed from the connectionist camp. As we have mentioned before each rule takes three parameters to specify a pattern. Each of these parameters will associate with a vector of syntactic feature[5] roles. So, every P-rule will have a feature role vector:

$$(F_{11},\ldots, F_{1n}, F_{21},\ldots, F_{2n}, F_{31},\ldots, F_{3n})$$

Each feature role is in turn associated with a weight. So each P-rule also has a vector of weights:

$$(W_{11},\ldots, W_{1n}, W_{21},\ldots, W_{2n}, W_{31},\ldots, W_{3n})$$

If a rule is applicable at a certain situation, the feature roles will be filled by the corresponding feature values given by the language objects matched out by the pattern of the rule. The value of each feature can be either 1 or 0. Let F'_{ij} denotes the value filling role F_{ij} at one particular application. The cost of applying that rule at that situation will be:

$$-\sum W_{ij} \bullet F'_{ij} \qquad (*)$$

The weights can be trained by a training algorithm similar to that used in the single layer perceptron network.

To summarize, the syntactic constraints are encoded partly as P-rules and partly as the weights associated with each P-rule. The P-rules are used to guide the parsing and tell which constituent should combined with which constituent. There are two types of syntactic preferences encoded in the weights: the preferences of the language to different P-rules and the preferences of each P-rule to different syntactic features. The P-rule is still symbolic and working on a stack. So, the ability of recursion is kept.

2.2. Organization of the Parser

At the topmost level, the parser is a standard searching algorithm. Due to its efficiency, A* algorithm (Nilsson, 1980) is used. However, it is worth noting that any other standard searching algorithm can also be used. For any searching algorithm, the following things need to be specified: 1. what the states in the searching space are; 2. what the initial state is; 3. what the action rules are and taking one state how they create new states; 4. what the cost of creating a new node is (please note that the cost is actually charged to the edge of the searching graph); 5. what the final states are.

In the parser, the searching states are pairs like:

(<partial result>,<unparsed sentence>)

The partial result is represented as one or more parsing trees which are kept on a working stack. Details of this representation are given in next section. The initial state is a pair of an empty stack and a sentence to be parsed. The action of the searching process is to look at the next read-in word and the roots of the top two trees on the working stack, which represent the two most recent found active constituents. The searching will then search for all the applicable P-rules based on what it sees. All P-rules it has found will then be fired. The action part of these P-rules will decide how to manipulate the trees on the stack and the current read-in. It will also decide whether it needs to read in next word. Therefore, for each of these P-rules being fired, a new state is created. The cost of creating this new state is following: the cost of applying the P-rule on its father state; the degree of violation of any new local preference just being found in the newly built trees; the cost of contextual preference[6] associated with the new consumed read in sense. All these costs are normalized and added to yield the total cost of creating this new state. The reason for normalization is that, for example, the cost of applying P-rule (see section 2.1) needs to be normalized to be positive as it is required by A* algorithm. The relative magnitudes of different types of costs need also be balanced, so that one kind of costs will not overwhelm the others. The final states of the searching are the states where the unparsed sentence is nil and the working stack has only one complete parsing tree on it. Obviously the output of this searching algorithm is the reading (represented as the tree) of the input sentence which violates the least amount of syntactic and semantic preferences.

So far the heuristics used in the A* searching is simply taken to be:

parsing is actually guided by both syntactic preferences and semantic preferences.

[5] The use of term "syntactic feature" is only to make it distinct from semantic preference and semantic type . We, by no means, try to exclude the use of those features which are generally considered as semantic features but which will help to make the right syntactic choice.

[6] For the idea of contextual preferences see (Slator, 1988).

αlc

where α is a constant. l is the length of the unparsed sentence. c is the average cost for parsing each word.

It is needed to mention that the input sentence we talked above is actually a list of lexical frames. Each lexical frame contains the following information about the word it stands for: the part of speech, syntactic features, local preferences, contextual preferences, etc. Since words can have more than one senses and one lexical frame will only represent one sense, for each read-in word, the parser will have to work for each of its senses and produce all of their children.

3. Some Details

3.1. P-rules, Parsing Trees, and P-parsers.

Given an input string α on a vocabulary Σ[7], a parsing tree of the input is a tree[8] with all its nodes corresponding to one of the words in the input string. For example, one of the parsing trees for input *abcde* is:

(e (a) (c (b) (d)))

Here the head of each list is the root of the tree or the subtree, and the tail of the list are its children. Intuitively the parenthood relation reflects the attachment or dependency relation in the input expression. For example, given an input expression *a small dog*, the *dog* will be the father of both *a* and *small*. The parsing tree is more formally defined as follows:

Definition (Parsing Tree): Given an input string α, the parsing tree on α is recursively defined as following:

1. (a) is a parsing tree on α, if a is in α.
2. $(a\ T_1\ T_2\ .....\ T_i)$ is a parsing tree on α, if a is in α and T_k $(1 \le k \le i)$ is also a parsing tree on α.

Definition (Complete Parsing Tree): Suppose T is a parsing tree on α. If for all the a in α, a is also in T, then T is a complete parsing tree on α.

To reduce the computational complexity, we will limit our attentions to only one special type of parsing trees, namely the normal parsing tree.

[7] In the actual parser Σ is the set of all the parts of speech.

[8] The order of children for any node is significant here, and we will use LISP convention to represent the parsing tree in this paper.

It should not be difficult to see and prove from the following definition that normal parsing trees are simply the trees which do not have "crossing dependencies" (Maling & Zaenen, 1982).

Definition: (Normal Parsing Tree): Suppose T is a parsing tree on α. T is a normal parsing tree on α iff for all the nodes in T, if they have children, say $T_1,.....,T_k$, then all the nodes in T_i must be appeared before all nodes in T_j in the input string α, where $1 \le i < j \le k$.

The P-parser has an input tape, and the reading head is always moving forward. The P-parser also keeps a stack of parsing trees which will represent the parsing result of the part of the input which has already been read. Besides there is a working register in the P-parser to store the current read in word. As you will see later, the read-in word is actually transformed into a tree before it being stored in the working register. The configuration of the P-parser is thus defined as a triple [<the stack>,<content of working register>,<unparsed input>]. The P-rule is used to specify how the P-parser works. If the input is of a vocabulary Σ, the P-rules on it can be defined as follows:

Definition (P-rule): A P-rule on Σ is a member of set $\Sigma' \times \Sigma' \times \Sigma' \times \Lambda$, where Σ' is $\Sigma \cup \{nil\}$ and Λ is the set of actions defined below.

Definition (Action): Actions are defined as functions of type $\Xi \rightarrow \Xi$, where Ξ is the set of configurations. There are total three different actions used in P-rules, and they are listed below:

δ_1[S,C,R]= [(cons C S), (list (car R)), (cdr R)]
δ_2[S,C,R]= [(cdr S), (cons (car C) (cons (car S) (cdr C))), R]
δ_3[S,C,R]= [(cons (append (car S) (list (cadr S))) (cddr S)), C, R]

Action 1 simply pushes the current read-in on the stack and then reads the next word from the input. Action 2 attaches top of the stack as the first child of the current read-in stored in the working register. Action 3 pops the top of the stack first and then attaches it to the second top of the stack as its last child.

The initial configuration for the P-parser is [nil, (list (car α)), (cdr α)], where the α is the input string to be parsed. There is a set of P-rules in each P-parser to specify its behavior. The P-parser will work non-deterministically. A P-rule can be fired iff its three parameters match the roots of the top two parsing trees on the stack and the root of the tree in the working register

respectively. Note the P-rule does not care about the unparsed input part in the configuration triple. A configuration is a final configuration iff the unparsed input string and the working register are all empty and the stack only has one parsing tree on it. A input is grammatical if and only if the P-parser can reach from the initial configuration to a final configuration by applying a sequence of P-rules taken from its grammar set. If there are more than one possible final states for a given input string and the parsing trees produced at these states are different, then we say that the input string is ambiguous. Here is a simple example to show how the P-parser works. You may need a pen and a piece of paper to work it out. Given input *abcdc* (*a good friend of mine*, for example), the parsing tree *(c (a) (b) (d (c)))* can be constructed by applying the following sequence of rules:

(nil,nil,a)=>1	(a,nil,b)=>1	(b,a,c)=>2
(a,nil,c)=>2	(nil,nil,c)=>1	(c,nil,d)=>1
(d,c,c)=>1	(c,d,nil)=>3	(d,c,nil)=>3

Most properties of this formalism will not be of the interests of this paper, except this theorem:

Theorem: A P-parser with all the P-rules on Σ as its grammar set can produce all the complete normal parsing trees for any input string on Σ.
PROOF: It is easy to prove this by induction on the length of the input string. □

The theorem tells that a P-parser with all the possible P-rules as its rule set can produce for any input string all of its possible parsing trees which have no crossing dependencies. This alone may not be very useful since this P-parser will also accept all the strings over Σ. However, as we have shown in the last section, with a proper weighting scheme, this P-parser offers a suitable framework for coding syntactic preferences.

3.2. Syntactic Preferences in the Weights of P-rules

Each lexical item will have a vector (of a fixed length) containing syntactic features. The value of the each feature is either 1 or 0. Each of the three parameters in the P-rule is associated with a vector (of the same length) of syntactic feature roles. Each of these syntactic feature roles is associated with a weight. Each weight thus encodes a preference of the P-rule toward the particular syntactic feature it corresponds to. A higher weight means the P-rule will be more sensitive to the appearance of this syntactic feature, and the result of applying this rule therefore will be more competitive when it does appear. The preferences of the language to different P-rules are also reflected in weights. Instead of being reflected in each individual weight, they are reflected in the distribution of weights in P-rules. A P-rule with a higher average weight is generally more favored than a P-rule with a lower average weight, since the higher the average weight is, the lower the cost of applying the P-rule tends to be. It is also necessary to emphasize that these two types of preferences are closely integrated.

3.3. Weight Learning

The weights of all the P-rules will be trained. The training is supervised. There are two different methods to train the weights. The first method takes an input string and a correct parsing tree for that string. We will use an algorithm to computer a sequence of P-rules that will produce the given parsing tree from the given input string. This algorithm is not difficult to design, and due to the space limits we will not present it here. After the P-rule sequence is produced, each of P-rule in the sequence will get a bonus δ. The bonus will further be distributed to the weights in the P-rule in the same fashion as that in the single layer perceptron network, that is:

$$\Delta W_{ij} = \eta \delta F'_{ij}$$

where η is the learning factor.

The second method requires less interventions. The parser is let to work on its own. The user only need to tell the parser whether the output it gives is correct. If the result is correct, all the P-rules used to construct this output will get a bonus and all the rules used during the parsing which did not contribute to the output will get a punishment. If the answer is wrong, all the P-rules used to construct this answer will get a punishment, and the parser will continue to search for a second best guess. The bonus and punishment will be distributed to the weights of the P-rule in the same manner as that in the first method.

The first method is more appropriate at the early stage of the training when most of the rules have about the same weights. It will take much longer for the parser to find the correct result on its own at this time. On the other hand, the second method needs less interventions. So, it will be more appropriate to be used whenever it can work reasonably well.

It is well known that the single layer perceptron net can not be trained to deal with exclusive-or. The same situation will also happen here. Since exclusive-or relations do exist between syntactic

features, we need to solve this problem. The simplest solution is to make multiple copies for the P-rule, and, hopefully, each copy will converge to each side of the exclusive-or relation.

3.4. Measuring Preference Violations

The syntactic preference violation is measured by formula (*) in section 2. Both action 2 and action 3 of P-rule actions make an attachment, and some semantic preferences may be either violated or satisfied by the attachment. So, after each application of such P-rule actions, the parser needs to check whether there are some new preferences being violated or satisfied. If there are, it will computer the cost of these violations and report it to the searching process. Similarly, each action 1 will cause a new contextual preference being reported. The measurement for the violation degree of both local preferences and contextual preferences is basically taken from PREMO (Slator, 1988), which we will not repeat here.

4. Conclusion and Comparisons

Preference Semantics offers an excellent framework for robust parsing. However, how to make full use of syntactic constraints has not been addressed. Using weights to code syntactic constraints on a relatively small set of P-rules (from which all the possible syntactic structures can be derived) enables us to expand the philosophy of Preference Semantics from semantic constraints to syntactic constraints. The weighting system not only reflects the preferences of the language, say English, to different P-rules but also reflects the preferences of each P-rule to each syntactic feature. Besides of this, it also offers a nice interface so that we can integrate the application of these syntactic preferences nicely with the application of both local semantic preferences and contextual preferences by using a highly efficient searching algorithm.

This project has also shown that some of the techniques commonly associated with the connectionism, such as coding information as weights, training the weights, and so on, can also be used to benefit symbolic computing. The result is gaining the robustness and adaptability while not losing the advantages of symbolic computing such as recursion, variable binding, etc.

The notion 'syntactic preference' has been used in (Pereira, 1985) (Frazier & Fodor, 1978) (Kimball, 1973) to describe the preference between Right Association and Minimal Attachment. Our approach shares some similarities with (Pereira, 1985), in that MA and RA simply "corresponds to two precise rules on how to choose between alternative parsing actions" at certain parsing configurations. However, he did not offer a framework for how one of them will be preferred. According to our model the preference between them will be based on the weights associated with the two rules, the syntactic features of the words involved and the semantic preferences found between these words. Besides, the idea of syntactic preferences in this paper is more general than the one used in their work, since it includes not only the preference between MA or RA but other kinds of syntactic preferences as well.

Wilks, Huang and Fass (1985) showed that prepositional phrase attachments are possible with only semantic information. In their approach syntactical preferences are limited to the order of matchings and the default attaching. Their attaching algorithm can be seen as a special case of the model we proposed here, in that, if they are correct, the preferences between the sequences of rules used for RA and MA would turn out to be very similar so that the semantic preferences involved would generally over shadow their effects.

There are some significant differences between our approach and some hybrid systems (Kwasny & Faisal, 1989) (Simmons & Yu, 1990). First, our approach is not a hybrid approach. Everything in our approach is still symbolic computing. Second, in our approach the costs of the applications of syntactic constraints are passed to a global search process. The search process will consider these costs along with the costs given by other types of constraints and make a decision globally. In a hybrid system, the syntactic decision is made by the network locally, and there is no intervention from the semantic processing. Third, our parser is nondeterministic while the parsers in hybrid systems are deterministic since there is no easy way to do backtracking. It is also easy to see that without the intervention of semantic processing, lacking the ability of backtracking is hardly an acceptable strategy for a practical natural language parser. Finally, in our approach each P-rule has its own set of weights, while in the hybrid systems all the grammar rules share a common network, and it is quite likely that this net will be overloaded with information when a reasonably large grammar is used.

Acknowledgment

All the experiments for this project are carried on the platform given by PREMO which was designed and implemented by Brian Slator when he was here in CRL. The author also wants to thank Dr Yorick Wilks, Dr David Farwell, Dr John Barnden and one referee for their comments. Of course, the author is solely responsible for all the mistakes in the paper.

References

1. J. G. Carbonell and P. J. Hayes, "Recovery Strategies for Parsing Extragrammtical Language," *American Journal of Computational Linguistics*, vol. 9(3-4), pp. 123-146, 1983.
2. D. Fass and Y. Wilks, "Preference Semantics, Ill-Formedness, and Metaphor," *American Journal of Computational Linguistics*, vol. 9(3-4), pp. 178-187, 1983.
3. M. G. Dyer, "Symbolic NeuroEngineering and natural language processing: a multilevel research approach.," in *Advances in Connectionist and Neural Computation Theory, Vol. 1.*, ed. J.A. Barnden and J.B. Pollack, pp. 32--86, Ablex Publishing Corp. , Norwood, N.J., 1991.
4. L. Frazier and J. D. Fodor, "The Sausage Machine: A New Two-Stage Parsing Model," *Cognition*, vol. 6, pp. 291-325, 1978.
5. X. Huang, "XTRA: The design and Implementation of A Fully Automatic Machine Translation System (Doctoral Dissertation)," *Memoranda in Computer and Cognitive Science*, vol. MCCS-88-121, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003, USA, 1988.
6. J. Kimball, "Seven Principles of Surface Structure Parsing in Natural Language," *Cognition*, vol. 2, pp. 15-47, 1973.
7. S. C. Kwasny and N. K. Sondheimer, "Relaxation theories for parsing ill-formed input," *American Journal of Computational Linguistics*, vol. 7, no. 2, pp. 99-108, 1981.
8. S. C. Kwasny and K. A. Faisal, "Competition and Learning in a Connectionist Deterministic Parser," *Procs. 11th Annual Conf. of the Cognitive Science Society*, pp. 635-642., Lawrence Erlbaum, Hillsdale, N.J., 1989.
9. W. G. Lehnert, "Symbolic/Subsymbolic Sentence Analysis: Exploiting the best of two worlds.," in *Advances in Connectionist and Neural Computation Theory, Vol. 1.*, ed. J.A. Barnden and J.B. Pollack, pp. 32--86, Ablex Publishing Corp. , Norwood, N.J., 1991.
10. J. Maling and A. Zaenen, "A Phrase Structure Account of Scandinavian Extraction Phenomena," in *The Nature of Syntactical Representation*, ed. P. Jacobson and G. K. Pullum, pp. 229--282, Reidel Publishing Company, Holland, 1982.
11. N. Nilsson, *Principles of AI*, Tioga pulishing co., Menlo Park, 1980.
12. F. C. N. Pereira, "A New Characterization of Attachment Preference," in *Natural Language Parsing*, ed. D. R. Dowty, L. Karttunen & A. M. Zwicky, pp. 307-319, Cambridge University Press, Cambridge, 1985.
13. M. Selfridge, "Integrated Processing Produces Robust Understanding," *Computational Linguistics*, vol. 12(2), pp. 89-106, 1986.
14. R. F. Simmons and Y. H. Yu, "Training a Neural Network to be a Context Sensitive Grammar," *Proceedings of the 5th Rocky Mountain Conference on AI*, pp. 251-256, Las Cruces, NM., 1990.
15. B. M. Slator, "Lexical Semantics and Preference Semantics Analysis (Doctoral Dissertation)," *Memoranda in Computer and Cognitive Science*, vol. MCCS-88-1, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003, USA, 1988.
16. D. L. Waltz and J. B. Pollack, "Massive Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation," *Cognitive Science*, vol. 9, pp. 51-74, 1985.
17. R. M. Weischedel and N. K. Sondheimer, "Meta-rules as a basis for processing ill-formed output," *American Journal of Computational Linguistics*, vol. 9, no. 3-4, pp. 161-177, 1983.
18. Y. Wilks, "A Preferential Pattern-Seeking Semantics for Natural Language Inference," *Artificial Intelligence*, vol. 6, pp. 53-74, 1975.
19. Y. Wilks, "Making Preferences More Active," *Artificial Intelligence*, vol. 11, pp. 197-223, 1978.
20. Y. A. Wilks, X. Huang and D. Fass, "Syntax, preference and right attachment," *IJCAI-85*, pp. 635-642., 1985