HEURISTICS FOR BROAD-COVERAGE NATURAL LANGUAGE PARSING

Michael C. McCord

IBM T. J. Watson Research Center
POB 704 Yorktown Heights, NY 10598

ABSTRACT

The Slot Grammar system is interesting for natural language applications because it can deliver parses with deep grammatical information on a reasonably broad scale. The paper describes a numerical scoring system used in Slot Grammar for ambiguity resolution, which not only ranks parses but also contributes to parsing efficiency through a parse space pruning algorithm. Details of the method of computing parse scores are given, and test results for the English Slot Grammar are presented.

1. INTRODUCTION

As everyone who has tried it knows, the hardest part of building a broad-coverage parser is not simply covering all the constructions of the language, but dealing with ambiguity.

One approach to ambiguity resolution is to "understand" the text well enough – to have a good semantic interpretation system, to use real-world modeling, inference, etc. This can work well in small domains, and it is, in this author's opinion, ultimately necessary for the highest quality of natural language processing in any domain; but it is probably not feasible on a broad scale today. So some kind of heuristic method is needed for disambiguation, some way of ranking analyses and choosing the best. Even in the ideal model of human language processing (which would use a great deal of knowledge representation and inference), ranking heuristics seem appropriate as a mechanism since humans must work with incomplete knowledge most of the time.

Two major questions that can be asked about a heuristic method for ambiguity resolution are these:

1. What level of representation is used for disambiguation and is involved in the statements of the heuristic rules – lexical/morphological, surface syntactic, deep syntactic, or logical/semantic?
2. Where do the heuristic rules come from? Are they largely created through human linguistic insight, or are they induced by processing corpora?

This paper describes the heuristic method used in the Slot Grammar (SG) system [10, 11, 13, 16, 17] for ambiguity resolution – the SG parse scoring system. This scoring system operates during parsing (with a bottom-up chart parser), assigning real number scores to partial analyses as well as to analyses of the complete sentence. The scores are used not only for ranking the final analyses but also for pruning the parse space during parsing, thus increasing time and space efficiency.

The level of representation being disambiguated is thus the level of SG parses. SG parse structures are dependency- or head-oriented, and include, in a single tree, both surface structure and deep syntactic information such as predicate-argument structure, remote dependencies, control information, and unwinding of passives. [1]

SG parse structures also include a choice of *word senses*. The extent to which these represent semantic sense distinctions depends on the lexicon. The SG system is set up to deal with semantic word-sense distinctions and to resolve them by doing semantic type-checking during parsing. However, in the lexicon for **ESG** (English Slot Grammar), nearly all word sense distinctions are a matter of part of speech or syntactic slot frame. Some semantic types are shown in the lexicon and are used in parsing, but generally very few. Thus one would say that **ESG** parse structures are basically *syntactic* structures, although the deep information like argument structure, passive unwinding, etc., counts for "semantics" in some people's books.

Where do the SG scoring rules come from – human linguistic insight or induction from corpus processing? The score of an SG parse, which will be described in Section 4, is the sum of several components. Most of these come completely from human linguistic insight, though some of them get their numeric values from corpus processing. In the tests reported in the final section, only the "linguistic-insight" rules are used. Some previous tests using the corpus-based heuristic rules together with the main SG heuristic rules showed that the former could improve the parse rate by a few percentage points. It is definitely worth pursuing both approaches, and more work will be done with a combination of the two.

[1] No attempt is made to resolve quantifier scoping in SG parses, although there is a post-processing system that produces a logical form with scope resolution for quantifiers and other "focalizers"[12]. Anaphora resolution [8, 9] is also done by post-processing SG parses.

In the next section we give a brief overview of Slot Grammar. In Section 3 we describe the scoring system generally and its use in parse space pruning, and in Section 4 we give some details of the computation of scores. Finally, Section 5 presents the results of some tests of **ESG**.

2. OVERVIEW OF SLOT GRAMMAR

The *slots* that figure in Slot Grammar rules and parsing come in two varieties: *complement slots* and *adjunct slots*. Analysis is word-oriented, and slots are associated with word senses. The complement slots for a word sense are associated with it in the lexicon. The adjunct slots depend only on the part of speech of the word sense and are listed for that part of speech in the grammar. Slots have names like *subj* and *obj* and should be thought of basically as *syntactic* relations, though complement slot frames in the lexicon can be viewed as corresponding to arguments in logical form.

The notion that a phrase *fills* a slot of a word (sense) is primitive in the grammar, and the conditions under which this can happen are given by the *slot-filler* rules. Grammatical analysis of a phrase consists basically of choosing, for each word of the phrase, (1) a word sense, (2) a feature structure, and (3) filler subphrases for its slots. A slot is *obligatory* or *optional* according as it must be, or need not be, filled in order for the analysis to be complete. Adjunct slots are normally optional. A complement slot can be filled at most once, but adjunct slots can, by default, be filled multiply.

The parser works bottom-up, beginning with one-word phrases and attaching other phrases as left and right modifiers as they can fill slots. As a phrase is built up in this way, it retains a distinguished head word, and the slots associated with this head word are considered slots of the phrase.

An example of a slot-filler rule is the following for the subject slot (in simplified form):

$$subj \Longrightarrow f(noun(nom, Num)) \,\&\, hf(verb(fin(Num))).$$

A goal $f(Feat)$ on the right hand side of a filler rule tests that the feature structure of the filler phrase matches $Feat$. A goal $hf(Feat)$ tests the feature structure of the *higher* phrase – the phrase (with possibly other modifiers attached) with which the slot is associated. The SG formalism includes a rich set of special predicates like f and hf that can be used for examining any aspects of the filler phrase and higher phrase for a slot filling.

Slot-filler rules normally do not constrain left-to-right ordering of the phrases involved. Instead, there are modularly stated ordering rules, which are applied as constraints in parsing after slot-filler rules apply.

Generally, there is a modular treatment of different grammatical phenomena in a Slot Grammar. There are separate rule systems not only for slot-filling and ordering, but also for coordination, unbounded dependencies, obligatory slots, adjunct slot declaration, "or-slots", punctuation, and parse scoring. All these rule types make use of the same system of special predicates (mentioned above for slot-filler rules) for examining the phrases involved in slot filling. Modularization of the rule system makes large grammars more manageable and also makes it easier to adapt a grammar for one language to another language.

There are currently Slot Grammars (in various states of completeness) for English, German, Spanish, Danish, Norwegian, and Hebrew. A great deal of attention has been paid to the development of a large, language-universal component of the system, the Slot Grammar *shell*. For a particular language, the shell represents roughly 65% of the rules/facts, not counting lexicons. All of the rule types mentioned above have part of their treatment in the shell, but there are especially large language-universal components for coordination, unbounded dependencies, punctuation, and parse scoring. Nevertheless, for all of these, there can be rules in the language-specific grammar that override or augment the language-universal rules.

The lexicon for **ESG** consists of a hand-coded portion for approximately 6,000 lemmas (basically most frequent words), plus a large back-up lexicon of approximately 60,000 lemmas derived from UDICT [1, 7] and other sources. Mary Neff is working on improvements of the large **ESG** lexicon through extraction from standard dictionaries.

Slot Grammars are used for source analysis in the MT system **LMT** [14, 15].

For a more detailed description of current version of the SG system, see [16, 17, 18]. In this paper we concentrate on the scoring system, in its latest form.

3. SCORING AND PARSE SPACE PRUNING

During parsing, each analysis P of a subphrase is assigned a real number $score(P)$. A larger number represents a worse score. As described in the next section, most of the ingredients that go into scores are positive numbers that are like penalties for unusual structure, and total scores are normally positive.

Parse space pruning involves comparison of scores of partial analyses and pruning away analyses that have relatively bad scores; but the comparisons are made only within certain equivalence classes of analyses. Two partial analyses are *equivalent* when they have the same boundaries, the same head word, and the same *basic feature*. For most categories, the basic feature is just the part of speech, but for verbs a finer distinction is made according to the inflection type (finite,

infinitive, etc.) of the verb. The notion of equivalence is loosened in certain ways for coordinated phrases that will not be described here.

Pruning is done as follows. Suppose P is a new, candidate partial analysis obtained in parsing. Let $compar(P)$ denote the set of existing partial analyses that are equivalent to P (not including P itself). Because of previous pruning, all members of $compar(P)$ have the same score; call this number $scompar(P)$. (If $compar(P) = \emptyset$ then consider $scompar(P) = +\infty$.) The system stores this best score $scompar(P)$ for the equivalence class of P in a way that can immediately be computed from P without searching the chart.

Now three things can happen: (1) If $score(P) > scompar(P)$, then P is discarded. (2) If $score(P) = scompar(P)$, then P is simply added to the chart. (3) If $score(P) < scompar(P)$, then P is added to the chart and all members of $compar(P)$ are discarded.

This parse space pruning can be turned on or off at run time, but scoring is done in any case, and final parses are ranked by the scoring system whether pruning is on or off. Generally, parse space pruning is crucial to the running of the system for large sentences because of space and time problems if it is not used. When pruning is turned on, there are generally very few final parses obtained for a sentence – on average about 1.3 (per successfully parsed sentence).

When parsing of a sentence fails, the system pieces together a "fitted parse" in a manner somewhat similar to that in [5]. The scores obtained for partial analyses figure heavily in choosing the pieces for this result.

4. COMPUTATION OF SCORES

The score of a partial analysis is obtained incrementally in building up the analysis. The initial score, for a one-word analysis, is associated with the word sense, and the score is incremented whenever a slot is filled or a coordinate structure is formed. All in all, there are eight main components of the total score (the score is the sum of them). We first list them with a brief description and then discuss them individually. The list is in decreasing order of "importance" – the amount of effort put into the rules/data for the component and roughly the current contribution of the component to successful parsing. Components 1, 2, 3, 4, 5, 8 are totally "human-coded", and components 6, 7 get their data from corpus processing.

Most of the heuristics described in [4] are covered by these rules (and were developed independently).

1. *SlotPref.* Computed when a modifier is attached by filling a slot. Measures the preference for using that slot *vs.* other slots for attaching the given modifier to the given higher phrase.
2. *ParallelCoord.* Favors parallelism in coordination.
3. *CloseAttach.* Favors close attachment.
4. *PhrasePref.* Tests the characteristics of a "completed" phrase – a phrase that becomes a modifier or is taken as an analysis of the complete input segment. Similar to *SlotPref*, but independent of the slot.
5. *WordSensePref.* Favors one sense of a word over other senses.
6. *HeadSlotFiller.* Used, like *SlotPref*, when a slot is filled, but tests for specific choices for the head words of the higher phrase and the filler phrase, as well their parts of speech, and tests only these things.
7. *POSPref.* Similar to *WordSensePref*, but tests only for part of speech.
8. *LexSlotPref.* Another score associated with filling a given slot, but coded in the lexicon in a given slot frame and can test semantic type conditions on the filler.

In the following more detailed description of the scoring components, we will use "XSG" to refer to the language-specific part of the Slot Grammar of language X. Thus the rules for grammar of X reside in both the SG shell and XSG.

SlotPref The rules for *SlotPref* are coded in both the shell and XSG. The default score, given in the shell, is +1 for an adjunct slot and 0 for a complement slot, so that complement slots are preferred over adjuncts.

For an example, consider the sentence *John sent the file to Bill.* The PP *to Bill* can attach to *file* by an adjunct slot or to *sent* by a complement slot (its indirect object), but the default *SlotPref* score is 1 for the adjunct and 0 for the complement, [2] so the analysis with the complement wins and the other is pruned away. [3]

Slot-scoring rules in XSG can override the default. Currently, out of a total of 678 rules of various types in **ESG**, 216 are slot-scoring rules. Most of the day-to-day effort in improving **ESG** consists of work on these scoring rules.

A slot-scoring rule is of the form:

$$Slot + E \ (\leftarrow Body).$$

E is normally a real number and is the contribution of this rule to *SlotPref.* The *Body*, if present, can contain special predicates like those mentioned in Section 2 for slot-filler

[2] Actually, a slot-scoring rule in **ESG** gives the adjunct a score of 2 in this instance.

[3] The *CloseAttach* component by itself favors the closer attachment of the PP to *file*, but this score component is dominated by *SlotPref.*

rules that test any characteristics of the filler phrase and the higher phrase.

Two examples of slot-scoring rules in **ESG** are as follows (given in simplified form).

$$ndet + 0 \leftarrow hf(noun(T, *, *))\&(T = cn|T = gerund).$$

$$vadv + 0 \leftarrow f(noun(*, *, *))\&st(tm).$$

The first says that the determiner slot for nouns is rewarded if the higher noun is a common noun or a gerund. The second says that the *vadv* slot for verbs is rewarded when it is filled by a time noun phrase. The special goal $st(Type)$ says that $Type$ is a semantic type of the (sense of) the filler phrase, and tm is a semantic type used in **ESG** for time words.

A slot-scoring rule might be used to penalize the use of a complement slot under certain conditions, by assigning it a score higher than the default 0. An example of this in **ESG** is

$$comp(binf) + 1 \leftarrow headf(noun(*, *, *))\&hrmods(nil).$$

Here $comp(binf)$ is a verb complement slot filled by a bare infinitive VP. This is penalized if the head word of the filler is *also* a noun and the higher verb has (so far) no right modifiers. This is to discourage use of the $comp(binf)$ analysis when the phrase may really be an NP, maybe an object. Several of the slot-scoring rules in **ESG** involve checks on existing *alternative* analyses of words, as in this example. It is quite useful to have such heuristics because of the great ambiguity of English in part of speech of words.

Slot-scoring may use conditions on punctuation. For example, for the slot *nprep* that finds adjunct PP modifiers of nouns, we might have:

$$nprep + 2 \leftarrow \neg sep(nil)\&\neg hrmodf(*, prep(*, *, *)).$$

This penalizes *nprep* if the separator is not nil (say, if there is a comma separator) and there is no other PP postmodifier already. Thus, in an example like *John noticed his neighbor, from across the street*, there will be a preference for the PP to modify *noticed* instead of *neighbor*.

ParallelCoord Most of the rules for this score component are in the shell. Parallelism in coordinate structures is measured by similarity of the conjuncts with respect to several different characteristics. Explicitly, when a coordinate structure is formed, the increment in the total score due to the *ParallelCoord* component is currently given by a formula (slightly simplified):

$$PFea + PFrame + PSense + PMods + PLen + PConj + PXSG.$$

Here the first five ingredients measure similarity of the conjuncts with respect to the following: (1) $PFea$: feature structures; (2) $PFrame$: complement slot frames; (3) $PSense$: word senses; (4) $PMods$: modifier list lengths; (5) $PLen$: word list lengths.

The ingredient $PConj$ tests for certain characteristics of the conjunction itself (which can include punctuation).

The ingredient $PXSG$ represents a contribution from language-specific coordination rules in XSG.

CloseAttach This score is essentially the same as that developed by Heidorn [3], and is designed to favor close attachment, although *SlotPref* and *ParallelCoord* can easily override it.

For a phrase P, the default for $CloseAttach(P)$ is defined recursively as the sum of all terms

$$0.1 * (CloseAttach(Mod) + 1),$$

where Mod varies over the modifiers of P. (One need not state the base of this recursive formula separately, since one arrives eventually at phrases with no modifiers, and then the sum over the empty list is understood to be zero.) The factor 0.1 used in the recursive formula is the default, and it can be overridden by an option in slot-scoring rules. Also, a slot-scoring rule can change the basic formula applied, in a way that will not be described here.

The combination of *SlotPref* and *CloseAttach* is closely related to preference rules discussed in [19].

PhrasePref Some rules for this component are coded in the shell and have to do with preferences for the feature structure of the analysis of a complete input phrase, for example preference of finite clauses over noun phrases (except in certain environments).

Phrase-scoring rules in XSG contribute to *PhrasePref*, and are of a form similar to slot-scoring rules – without mentioning a slot:

$$+E \leftarrow Body.$$

The real number E is added to the total score whenever a phrase satisfying *Body* fills any slot, or is used as a conjunct in a coordinate structure, or is taken as a top-level analysis.

A sample phrase-scoring rule in **ESG** is

$$+1 \leftarrow f(noun(*, *, *))\&\neg lmod(ndet, *)\& lmod(nadj, P)\&headst(P, quantadv).$$

This penalizes (by +1) a complete noun phrase that has no determiner but does have an adjective modifier which has *some* analysis with the feature *quantadv*. This rule penalizes for example the analysis of *even Bill* in which *even* is an adjective.

WordSensePref All of the rules for this component are coded in the lexicon. An example is a lexical entry for the word *man*:

$$man < n(human\&male, nil) < v(ev(2), *, obj1).$$

The first lexical analysis element shows *man* as a noun with features *human* and *male*. The second analysis shows a verb word sense with a *WordSensePref* penalty of +2.

These scores for word sense choices can also be coded *conditionally* on subject area codes, and there is an if-then-else formalism for expressing this.

The *WordSensePref* score is added when an initial (one-word) phrase analysis is formed.

HeadSlotFiller Following a method due largely to Ido Dagan [2], counts of head-slot-filler occurrences are obtained by parsing a corpus with **ESG**. Actually parts of speech are stored along with the head words of the higher and modifier phrases, so the counts are of quintuples:

$$(HWord, HPOS, Slot, MWord, MPOS).$$

These counts are then used (with certain coefficients) to add a reward (a negative number) to the score each time a modifier is attached with a match to a stored quintuple.

POSPref Using an idea of Ido Dagan and Herbert Leass, **ESG** corpus parsing is used to obtain counts of occurrences of pairs

$$(Word, PartOfSpeech).$$

When an initial (one-word) phrase analysis is formed, and the word and its part of speech match an entry in the table just mentioned, then the count, with a certain negative coefficient, is added as the *POSPref* contribution to the phrase score. This is of course similar to *WordSensePref*, taken from the lexicon, and there is an overlap.

LexSlotPref Rules for this component are coded in the lexicon. A slot appearing in a slot frame in a lexical entry can have an associated semantic type test on its filler. For example consider the following entry for *give* (not an actual entry for **ESG**):

$$give < v(obj\ .\ iobj{:}human).$$

Here the *iobj* slot requires a filler with the semantic type *human*. (In general, any Boolean combination of type tests can be coded.) If this analysis is used, then a *LexSlotPref* score of -1 is added. As it is stated, this semantic type requirement is absolute. But if one writes

$$give < v(obj\ .\ iobj{:}pref(human)).$$

then the test is not an absolute requirement, but merely gives a score increment of -1 if it is satisfied. In both the absolute and the "failsoft" forms of semantic type tests, the formalism allows one to specify arbitrary score increments.

5. TESTS OF ESG

Three recent tests of **ESG** coverage will be described, two on computer manual text and one on *Wall Street Journal (WSJ)* text. In all of the tests, there were no restrictions placed on vocabulary or length of test segments. Only the *first* parse given by **ESG** for each segment was considered. [4]

For each segment, parse output was rated with one of three categories – *p*: perfect parse, *pa*: *approximate parse*, or *bad*: not *p* or *pa*. To get a *p* rating, all of the SG structure had to be correct, including for example slot labels; so this is a stricter requirement than just getting surface structure or bracketing correct. An *approximate parse* is a non-perfect one for which nevertheless all the feature structures are correct and surface structure is correct except for level of attachment of modifiers. In MT applications, one can often get reasonable translations using approximate parses.

This way of rating parses is not an ideal one, because a parse for a very long sentence can be rated *bad* even when it has a single word with a wrong feature or slot. A combination of measures of partial success, such as those obtained by counting bracketing crossings, would be reasonable, since partially correct parses may still be useful. I can make up for this partially by reporting results as a function of segment length.

Test 1 This was done using a set of approximately 88,000 segments from computer manuals on which no training of **ESG** had been done. Half of the corpus, simply consisting of the odd-numbered segments, was used for some lexical training. Slava Katz's terminology identification program [6] was run on this portion as well as a program that finds candidate terms by looking (roughly) for sequences of capitalized words. About one day was spent editing this auxiliary multiword lexicon; the edited result consisted of 2176 entries. Then 100 segments were selected (automatically) at random from the (blind) even-numbered segments. The segments ranged in token list length from 2 to 38. The following table shows rating percentages for the segments of token list length $\leq N$ for selected N.

N	% *p*	% *p* or *pa*
10	75	75
17	71	79
25	66	76
38	61	73

[4] This first parse had the best score, but when more than one parse had the best, only the first one output by the system was used.

Test 2 From a set of about 2200 computer manual segments, 20% had been selected automatically at random, removed, and kept as a blind test set, and some **ESG** grammatical and lexical work had been done on the remaining. The test was on 100 of the blind test sentences, which happened to have the same range in token list length, 2 to 38, as in the preceding test. The following table, similar in form to the preceding, shows results.

N	% p	% p or pa
10	72	75
17	74	84
25	70	80
38	67	80

Test 3 This used a corpus of over 4 million segments from the *WSJ*. No attempt was made to isolate a blind test set. However, little work on **ESG** has been done for *WSJ* text – maybe looking at a total of 500 sentences over the span of work on **ESG**, with most of these obtained in other ways (I do not know if they were in the corpus in question). At any rate, automatic random choice from the 4M-segment corpus presumably resulted in segments that **ESG** had never seen in its life.

Prior to selection of the test set, Katz's terminology identification was run on approximately 40% of the corpus. A portion of the results (based on frequency) underwent about a day's worth of editing, giving an auxiliary multiword lexicon with 1513 entries.

Then 100 segments were selected at random from the 4M-segment corpus. They ranged in token list length from 6 to 57. **ESG** was run, with the following results, shown again as percentages for segments of length $\leq N$:

N	% p	% p or pa
10	75	75
17	48	56
25	45	55
38	33	48
57	29	45

ESG delivered some kind of analysis for all of the segments in the three tests, with about 11% fitted parses for the computer manual texts, and 26% fitted parses for the *WSJ*. The average parse time per segment was 1.5 seconds for the computer manuals and 5.6 seconds for the *WSJ* – on an IBM mainframe with a Prolog interpreter (not compiled).

References

1. Byrd, R. J. "Word Formation in Natural Language Processing Systems," *Proceedings of IJCAI-VIII*, 1983, pp. 704-706.
2. Dagan, I. and Itai, A. "Automatic Acquisition of Constraints for the Resolution of Anaphoric References and Syntactic Ambiguities," *Proceedings of Coling-90*, vol. 3, 1990, pp. 162-167.
3. Heidorn, G. E. "Experience with an Easily Computed Metric for Ranking Alternative Parses," *Proceedings of the 20th Annual Meeting of the ACL*, 1982, pp. 82-84.
4. Hobbs, J. R. and Bear, J. "Two Principles of Parse Preference," *Proceedings of Coling-90*, vol. 3, 1990, pp. 162-167.
5. Jensen, K. and Heidorn, G. E. "The Fitted Parse: 100% Parsing Capability in a Syntactic Grammar of English," Research Report RC9729, 1982, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.
6. Justeson, J.S. and Katz, S.M. "Technical Terminology: Its Linguistic Properties and an Algorithm for Identification in Text" (to appear).
7. Klavans, J. L. and Wacholder, N. "Documentation of Features and Attributes in UDICT," Research Report RC14251, 1989, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.
8. Lappin, S. and McCord, M.C. "A Syntactic Filter on Pronominal Anaphora in Slot Grammar" in *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 1990, pp. 135-142.
9. Lappin, S. and McCord, M.C. "Anaphora Resolution in Slot Grammar," *Computational Linguistics* 16, 1990, pp. 197-212.
10. McCord, M. C. "Slot Grammars," *Computational Linguistics*, vol. 6, 1980, pp. 31-43.
11. McCord, M. C. "Using Slots and Modifiers in Logic Grammars for Natural Language," *Artificial Intelligence*, vol. 18, 1982, pp. 327-367.
12. McCord, M. C. "Natural Language Processing in Prolog," in Walker, A. (Ed.), McCord, M., Sowa, J. F., and Wilson, W. G., *Knowledge Systems and Prolog: A Logical Approach to Expert Systems and Natural Language Processing*, Addison-Wesley, Reading, Mass., 1987.
13. McCord, M. C. "A New Version of Slot Grammar," Research Report RC14506, 1989, IBM T.J. Watson Research Center, Yorktown Heights, NY.
14. McCord, M. C. "Design of **LMT**: A Prolog-based Machine Translation System," *Computational Linguistics*, 15, 1989, pp. 33-52.
15. McCord, M. C. "**LMT**," *Proceedings of MT Summit II*, 1989, pp. 94-99, Deutsche Gesellschaft für Dokumentation, Frankfurt.
16. McCord, M. C. "Slot Grammar: A System for Simpler Construction of Practical Natural Language Grammars," In R. Studer (ed.), *Natural Language and Logic: International Scientific Symposium*, Lecture Notes in Computer Science, Springer Verlag, Berlin, 1990, pp. 118-145.
17. McCord, M. C. "The Slot Grammar System," Research Report RC17313, 1991, IBM T.J. Watson Research Center, Yorktown Heights, NY. To appear in J. Wedekind and C. Rohrer (Eds.), *Unification in Grammar*, MIT Press.
18. McCord, M. C., Bernth, A., Lappin, S., and Zadrozny, W. "Natural Language Processing within a Slot Grammar Framework," *International Journal on Artificial Intelligence Tools*, vol. 1, 1992, pp. 229-277.
19. Wilks, Y., Huang, X-M., and Fass, D. "Syntax, Preference and Right-Attachment," *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 1985, pp. 779-784.