A GENERALIZATION OF THE OFFLINE PARSABLE GRAMMARS

Andrew Haas

BBN Systems and Technologies, 10 Moulton St., Cambridge MA. 02138

ABSTRACT

The offline parsable grammars apparently have enough formal power to describe human language, yet the parsing problem for these grammars is solvable. Unfortunately they exclude grammars that use x-bar theory - and these grammars have strong linguistic justification. We define a more general class of unification grammars, which admits x-bar grammars while preserving the desirable properties of offline parsable grammars.

Consider a unification grammar based on term unification. A typical rule has the form

$t_0 \rightarrow t_1 \ldots t_n$

where t_0 is a term of first order logic, and $t_1 \ldots t_n$ are either terms or terminal symbols. Those t_i which are terms are called the top-level terms of the rule. Suppose that no top-level term is a variable. Then erasing the arguments of the top-level terms gives a new rule

$c_0 \rightarrow c_1 \ldots c_n$

where each c_i is either a function letter or a terminal symbol. Erasing all the arguments of each top-level term in a unification grammar G produces a context-free grammar called the *context-free backbone* of G. If the context-free backbone is finitely ambiguous then G is *offline parsable* (Pereira and Warren, 1983; Kaplan and Bresnan, 1982). The parsing problem for offline parsable grammars is solvable. Yet these grammars apparently have enough formal power to describe natural language - at least, they can describe the crossed-serial dependencies of Dutch and Swiss German, which are presently the most widely accepted example of a construction that goes beyond context-free grammar (Shieber 1985a).

Suppose that the variable M ranges over integers, and the function letter "s" denotes the successor function. Consider the rule

1 $p(M) \rightarrow p(s(M))$

A grammar containing this rule cannot be offline parsable, because erasing the arguments of the top-level terms in the rule gives

2 $p \rightarrow p$

which immediately leads to infinite ambiguity. One's intuition is that rule (1) could not occur in a natural language, because it allows arbitrarily long derivations that end with a single symbol:

$p(s(0)) \rightarrow p(0)$
$p(s(s(0))) \rightarrow p(s(0)) \rightarrow p(0)$
$p(s(s(s(0)))) \rightarrow p(s(s(0))) \rightarrow p(s(0)) \rightarrow p(0)$
....

Derivations ending in a single symbol can occur in natural language, but their length is apparently restricted to at most a few steps. In this case the offline parsable grammars exclude a rule that seems to have no place in natural language.

Unfortunately the offline parsable grammars also exclude rules that do have a place in natural language. The excluded rules use x-bar theory. In x-bar theory the major categories (noun phrase, verb phrase, noun, verb, etc.) are not primitive. The theory analyzes them in terms of two features: the phrase types noun, verb, adjective, preposition, and the bar levels 1,2 and 3. Thus a noun phrase is major-cat(n,2) and a noun is major-cat(n,1). This is a very simplified account, but it is enough for the present purpose. See (Gazdar, Klein, Pullum, and Sag 1985) for more detail.

Since a noun phrase often consists of a single noun we need the rule

3 major-cat(n,2) → major-cat(n,1)

Erasing the arguments of the category symbols gives

4 major-cat → major-cat

and any grammar that contains this rule is infinitely ambiguous. Thus the offline parsable grammars exclude rule (3), which has strong linguistic justification.

One would like a class of grammars that excludes the bad rule

$p(s(N)) \rightarrow p(N)$

and allows the useful rule

major-cat(n,2) → major-cat(n,1)

Offline parsable grammars exclude the second rule because in forming the context-free backbone they erase too much information - they erase the bar levels and phrase types, which are needed to guarantee finite ambiguity. To include x-bar grammars in the class of offline parsable grammars we must find a different way to form the backbone - one that does not require us to erase the bar levels and phrase types.

One approach is to let the grammar writer choose a finite set of features that will appear in the backbone, and erase everything else. This resembles Shieber's method of restriction (Shieber 1985b). Or following Sato et.al. (1984) we could allow the grammar writer to choose a maximum depth for the terms in the backbone, and erase every symbol beyond that depth. Either method might be satisfactory in practice, but for theoretical purposes one cannot just rely on the ingenuity of grammar writers. One would like a theory that decides for every grammar what information is to appear in the backbone.

Our solution is very close to the ideas of Xu and Warren (1988). We add a simple sort system to the grammar. It is then easy to distinguish those sorts S that are recursive, in the sense that a term of sort S can contain a proper subterm of sort S. For example, the sort "list" is recursive because every non-empty list contains at least one sublist, while the sorts "bar level" and "phrase type" are not recursive. We form the *acyclic backbone* by erasing every term whose sort is recursive. This preserves the information about bar levels and phrase types by using a general criterion, without requiring the grammar writer to mark these features as special. We then use the acyclic backbone to define a class of grammars for which the parsing problem is solvable, and this class includes x-bar grammars.

Let us review the offline parsable grammars. Let G be a unification grammar with a set of rules R, a set of terminals T, and a start symbol S. S must be a ground term. The *ground grammar* for G is the four-tuple (L,T,R′,S), where L is the set of ground terms of G and R′ is the set of ground instances of rules in R. If the ground grammar is finite it is simply a context-free grammar. Even if the ground grammar is infinite, we can define the set of derivation trees and the language that it generates just as we do for a context-free grammar. The language and the derivation trees generated by a unification grammar are the ones generated by its ground grammar. Thus one can consider a unification grammar as an abbreviation for a ground grammar. The present paper excludes grammars with rules whose right side is empty; one can remove this restriction by a straightforward extension.

A ground grammar is *depth-bounded* if for every $L > 0$ there is a $D > 0$ such that every parse tree for a string of length L has a depth $< D$. In other words, the depth of a parse tree is bounded by the length of the string it derives. By definition, a unification grammar is depth-bounded iff its ground grammar is depth-bounded. One can prove that a context-free grammar is depth-bounded iff it is finitely ambiguous (the grammar has a finite set of symbols, so there is only a finite number of strings of given length L, and it has a finite number of rules, so there is only a finite number of possible parse trees of given depth D).

Depth-bounded grammars are important because the parsing problem is solvable for any depth-bounded unification grammar. Consider a bottom-up chart parser that generates partial parse trees in order of depth. If the input α is of length L, there is a depth D such that all parse trees for any substring of α have depth less than D. The parser will eventually reach depth D; at this depth there are no parse trees, and then the parser will halt.

The essential properties of offline parable grammars are these:

Theorem 1. It is decidable whether a given unification grammar is offline parsable.

Proof: It is straightforward to construct the context-free backbone. To decide whether the backbone is finitely ambiguous, we need only decide whether it is depth-bounded. We present an algorithm for this problem.

Let C_n be the set of pairs [A,B] such that $A \stackrel{+}{\Rightarrow} B$ by a tree of depth n. Clearly C_1 is the set of pairs [A,B] such that $(A \rightarrow B)$ is a rule of G. Also, C_{n+1} is the set of pairs [A,C] such that for some B, $[A,B] \in C_n$ and $[B,C] \in C_1$. Then if G is depth-bounded, C_n is empty for some $n > 0$. If G is not depth-bounded, then for some non-terminal A, $A \stackrel{+}{\Rightarrow} A$.

The following algorithm decides whether a cfg is depth-bounded or not by generating C_n for successive values of n until either C_n is empty, proving that the grammar is depth-bounded, or C_n contains a pair of the form [A, A], proving that the grammar is not depth-bounded. The algorithm always halts, because the grammar is either depth-bounded or it is not; in the first case $C_n = \varnothing$ for some n, and in the second case $[A,A] \in C_n$ for some n.

Algorithm 1.
n := 1;
C_1 := {[A,B] | (A → B) is a rule of G }
while true do
[if $C_n = \emptyset$ then return true;
  if ($\exists$ A . [A,A] $\in C_n$) then return false;
  C_{n+1} := {[A,C] | ($\exists$ B . [A,B] $\in C_n$
                    $\wedge$ [B,C] $\in C_1$)};
  n := n+1;
]

Theorem 2. If a unification grammar G is offline parsable, it is depth-bounded.

Proof: The context-free backbone of G is depth-bounded because it is finitely ambiguous. Suppose that the unification grammar G is not depth-bounded; then there is a string α of symbols in G such that α has arbitrarily deep parse trees in G. If t is a parse tree for α in G, let t′ be formed by replacing each non-terminal $f(x_1...x_n)$ in t with the symbol f. t′ is a parse tree for α in the context-free backbone, and it has the same depth as t. Therefore α has arbitrarily deep parse trees in the context-free backbone, so the context-free backbone is not depth-bounded. This contradiction shows that the unification grammar must be depth-bounded.

Theorem 2 at once implies that the parsing problem is solvable for offline parsable grammars.

We define a new kind of backbone for a unification grammar, called the acyclic backbone. The acyclic backbone is like the context-free backbone in two ways: there is an algorithm to decide whether the acyclic backbone is depth-bounded, and if the acyclic backbone is depth-bounded then the original grammar is depth-bounded. The key difference between the acyclic backbone and the context-free backbone is that in forming the acyclic backbone for an x-bar grammar, we do not erase the phrase type and bar level features. We consider the class of unification grammars whose acyclic backbone is depth-bounded. This class has the desirable properties of offline parsable grammars, and it includes x-bar grammars that are not offline parsable.

For this purpose we augment our grammar formalism with a sort system, as defined in (Gallier 1986). Let S be a finite, non-empty set of sorts. An *S-ranked alphabet* is a pair (Σ,r) consisting of a set Σ together with a function $r : \Sigma \rightarrow S^* \times S$ assigning a *rank* (u,s) to each symbol f in Σ. The string u in S^* is the *arity* of f and s is the *sort* of f. Terms are defined in the usual way, and we require that every sort includes at least one ground term.

As an illustration, let S = { phrase, person, number }. Let the function letters of Σ be { np, vp, s, 1st, 2nd, 3rd, singular, plural }. Let ranks be assigned to the function letters as follows, omitting the variables.

r(np) = ([person, number],phrase)
r(vp) = ([person, number],phrase)
r(s) = (e,phrase)
r(1st) = (e,number)
r(2nd) = (e,number)
r(3rd) = (e,number)
r(singular) = (e,person)
r(plural) = (e,person)

We have used the notation [a,b,c] for the string of a, b and c, and e for the empty string. Typical terms of this ranked alphabet are np(1st,singular) and vp(2nd, plural).

A sort s is *cyclic* if there exists a term of sort s containing a proper subterm of sort s. If not, s is called *acyclic*. A function letter, variable, or term is called cyclic if its sort is cyclic, and acyclic if its sort is acyclic. In the previous example, the sorts "person","number", and "phrase" are acyclic. Here is an example of a cyclic sort. Let S = {list,atom} and let the function letters of Σ be { cons, nil, a, b, c }. Let

r(a) = (e,atom)
r(b) = (e,atom)
r(c) = (e,atom)
r(nil) = (e,list)
r(cons) = ([atom,list],list)

The term cons(a,nil) is of sort "list", and it contains the proper subterm nil, also of sort "list". Therefore "list" is a cyclic sort. The sort "list" includes an infinite number of terms, and it is easy to see that every cyclic sort includes an infinite number of ground terms.

If G is a unification grammar, we form the acyclic backbone of G by replacing all cyclic terms in the rules of G with distinct new variables. More exactly, we apply the following recursive transformation to each top-level term in the rules of G.

transform($f(t_1...t_n)$) =
  if the sort of f is cyclic
  then new-variable()
  else f(transform(t_1)...transform(t_n))

where "new-variable" is a function that returns a new variable each time it is called (this new variable must be of the same sort as the function letter f). Obviously the rules of the acyclic backbone subsume the original rules, and they contain no cyclic function letters. Since the

acyclic backbone allows all the rules that the original grammar allowed, if it is depth-bounded, certainly the original grammar must be depth-bounded.

Applying this transformation to rule (1) gives

$$p(X) \rightarrow p(Y)$$

because the sort that contains the integers must be cyclic. Applying the transformation to rule (3) leaves the rule unchanged, because the sorts "phrase type" and "bar level" are acyclic. In any x-bar grammar, the sorts "phrase type" and "bar level" will each contain a finite set of terms; therefore they are not cyclic sorts, and in forming the acyclic backbone we will preserve the phrase types and bar levels. In order to get this we result we need not make any special provision for x-bar grammars - it follows from the general principle that if any sort s contains a finite number of ground terms, then each term of sort s will appear unchanged in the acyclic backbone.

We must show that it is decidable whether a given unification grammar has a depth-bounded acyclic backbone. We will generalize algorithm 1 so that given the acyclic backbone G' of a unification grammar G, it decides whether G' is depth-bounded. The idea of the generalization is to use a set S of pairs of terms with variables as a representation for the set of ground instances of pairs in S. Given this representation, one can use unification to compute the functions and predicates that the algorithm requires. First one must build a representation for the set of pairs of ground terms [A,B] such that $(A \rightarrow B)$ is a rule in the ground grammar of G'. Clearly this representation is just the set of pairs of terms [C,D] such that $(C \rightarrow D)$ is a rule of G'.

Next there is the function that takes sets S_1 and S_2 and finds the set link(S_1,S_2) of all pairs [A,C] such that for some B, [A,B] $\in S_1$ and [B,C] $\in S_2$. Let T_1 be a representation for S_1 and T_2 a representation for S_2, and assume that T_1 and T_2 share no variables. Then the following set of terms is a representation for link(S_1,S_2):

$$\{\ s([A,C])\ |\ (\exists\ B,B'.\ [A,B] \in T_1 \wedge [B',C] \in T_2 \wedge s \text{ is the most general unifier of B and B}')\ \}$$

One can prove this from the basic properties of unification.

It is easy to check whether a set of pairs of terms represents the empty set or not - since every sort includes at least one ground term, a set of pairs represents the empty set iff it is empty. It is also easy to decide whether a set T of pairs with variables represents a set S of ground pairs that includes a pair of the form [A,A] - merely check whether A unifies with B for some pair [A,B] in T. In this case there is no need for renaming, and once again the reader can show that the test is correct using the basic properties of unification.

Thus we can "lift" the algorithm for checking depth-boundedness from a context-free grammar to a unification grammar. Of course the new algorithm enters an infinite loop for some unification grammars - for example, a grammar containing only the rule

1 $p(M) \rightarrow p(s(M))$

In the context-free case the algorithm halts because if there are arbitrarily long chains, some symbol derives itself - and the algorithm will eventually detect this. In a grammar with rules like (1), there are arbitrarily long chains and yet no symbol ever derives itself. This is possible because a ground grammar can have infinitely many non-terminals.

Yet we can show that if the unification grammar G contains no cyclic function letters, the result that holds for cfgs will still hold: if there are arbitrarily long chain derivations, some symbol derives itself. This means that when operating on an acyclic backbone, the algorithm is guaranteed to halt. Thus we can decide for any unification grammar whether its acyclic backbone is depth-bounded or not.

The following is the central result of this paper:

Theorem 3. Let G' be a unfication grammar without cyclic function letters. If the ground grammar of G' allows arbitrarily long chain derivations, then some symbol in the ground grammar derives itself.

Proof: In any S-ranked alphabet, the number of terms that contain no cyclic function letters is finite (up to alphabetic variance). To see this, let C be the number of acyclic sorts in the language. Then the maximum depth of a term that contains no cyclic function letters is C+1. For consider a term as a labeled tree, and consider any path from the root of such a tree to one of its leaves. The path can contain at most one variable or function letter of each non-cyclic sort, plus one variable of a cyclic sort. Then its length is at most C+1. Furthermore, there is only a finite number of function letters, each taking a fixed number of arguments, so there is a finite bound on the

number of arguments of a function letter in any term. These two observations imply that the number of terms without cyclic function letters is finite (up to alphabetic variance).

Unification never introduces a function letter that did not appear in the input; therefore performing unifications on the acyclic backbone will always produce terms that contain no cyclic function letters. Since the number of such terms is finite, unification on the acyclic backbone can produce only a finite number of distinct terms.

Let D_1 be the set of lists (A,B) such that (A $\rightarrow$ B) is a rule of G'. For $n > 0$ let D_{n+1} be the set of lists $s((A_0,...A_n,B))$ such that $(A_0,...A_n) \in D_n$, $(A',B) \in D_1$, and s is the most general unifier of A_n and A' (after suitable renaming of variables). Then the set of ground instances of lists in D_n is the set of chain derivations of length n in the ground grammar for G'. Once again, the proof is from basic properties of unification.

The lists in D_n contain no cyclic function letters, because they were constructed by unification from D_1, which contains no cyclic function letters. Let N be the number of distinct terms without cyclic function letters in G' - or more exactly, the number of equivalence classes under alphabetic variance. Since the ground grammar for G' allows arbitrarily long chain derivations, D_{N+1} must contain at least one element, say $(A_0,...A_{N+1})$. This list contains two terms that belong to the same equivalence class; let A_i be the first one and A_j the second. Since these terms are alphabetic variants they can be unified by some substitution s. Thus the list $s((A_0,...A_{N+1}))$ contains two identical terms, $s(A_i)$ and $s(A_j)$. Let s' be any subsitution that maps $s((A_0,...A_{N+1}))$ to a ground expression. Then $s'(s((A_0,...A_{N+1})))$ is a chain derivation in the ground grammar for G'. It contains a sub-list $s'(s(A_i,...A_j))$, which is also a chain derivation in the ground grammar for G'. This derivation begins and ends with the symbol $s'(s(A_i)) = s'(s(A_j))$. So this symbol derives itself in the ground grammar for G', which is what we set out to prove.

Finally, we can show that the new class of grammars is a superset of the offline parsable grammars.

Theorem 4. If G is a typed unification grammar and its context-free backbone is finitely ambiguous, then its acyclic backbone is depth-bounded.

Proof: Asssume without loss of generality that the top-level function letters in the rules of G are acyclic. Consider a "backbone" G' formed by replacing the arguments of top-level terms in G with new variables. If the context-free backbone of G is finitely ambiguous, it is depth-bounded, and G' must also be depth-bounded (the intuition here is that replacing the arguments with new variables is equivalent to erasing them altogether). G' is weaker than the acyclic backbone of G, so if G' is depth-bounded the acyclic backbone is also depth-bounded.

The author conjectures that grammars whose acyclic backbone is depth-bounded in fact generate the same languages as the offline parsable grammars.

Conclusion

The offline parsable grammars apparently have enough formal power to describe natural language syntax, but they exclude linguistically desirable grammars that use x-bar theory. This happens because in forming the backbone one erases too much information. Shieber's restriction method can solve this problem in many practical cases, but it offers no general solution - it is up to the grammar writer to decide what to erase in each case. We have shown that by using a simple sort system one can automatically choose the features to be erased, and this choice will allow the x-bar grammars.

The sort system has independent motivation. For example, it allows us to assert that the feature "person" takes only the values 1st, 2nd and 3rd. This important fact is not expressed in an unsorted definite clause grammar. Sort-checking will then allow us to catch errors in a grammar - for example, arguments in the wrong order. Robert Ingria and the author have used a sort system of this kind in the grammar of BBN Spoken Language System (Boisen et al., 1988). This grammar now has about 700 rules and considerable syntactic coverage, so it represents a serious test of our sort system. We have found that the sort system is a natural way to express syntactic facts, and a considerable help in detecting errors. Thus we have solved the problem about offline parsable grammars using a mechanism that is already needed for other purposes.

These ideas can be generalized to other forms of unification. Consider dag unification as in Shieber (1985b). Given a set S of sorts, assign a sort to each label and to each atomic dag. The arity of a label is a set of sorts (not a sequence of sorts as in term unification). A dag is well-formed if whenever an arc labeled l leads to a node n,

either n is atomic and its sort is in the arity of l, or n has outgoing arcs labeled $l_1...l_n$, and the sorts of $l_1...l_n$ are in the arity of l. One can go on to develop the theory for dags much as the present paper has developed it for terms.

This work is a step toward the goal of formally defining the class of possible grammars of human languages. Here is an example of a plausible grammar that our definition does not allow. Shieber (1986) proposed to make the list of arguments of a verb a feature of that verb, leading to a grammar roughly like this:

vp → v(Args) arglist(Args)
v(cons(np,nil)) → [eat]
arglist(nil) → *e*
arglist(cons(X,L)) → X arglist(L)

Such a grammar is desirable because it allows us to assert once that an English VP consists of a verb followed by a suitable list of arguments. The list of arguments must be a cyclic sort, so it will be erased in forming the acyclic backbone. This will lead to loops of the form

arglist(X) → arglist(Y)

Therefore a grammar of this kind will not have a depth-bounded acyclic backbone. This type of grammar is not as strongly motivated as the x-bar grammars, but it suggests that the class of grammars proposed here is still too narrow to capture the generalizations of human language.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the support of the Office of Naval Research under contract number N00014-85-C-0279.

REFERENCES

Boisen, Sean; Chow, Yen-lu; Haas, Andrew; Ingria, Robert; Roucos, Salim; Stallard, David; and Vilain, Marc. (1989) Integration of Speech and Natural Language Final Report. Report No. 6991, BBN Systems and Technologies Corporation. Cambridge, Massachusetts.

Bresnan, Joan, and Kaplan, Ronald. (1982) LFG: A Formal System for Grammatical Representation. in *The Mental Representation of Grammatical Relations*. MIT Press.

Gallier, Jean H. (1986) Logic for Computer Science. Harper and Row, New York, New York.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey; and Sag, Ivan. (1985) Generalized Phrase Structure Grammar. Oxford: Basil Blackwell.

Pereira, Fernando, and Warren, David H. D. (1983) Parsing as Deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, Massachusetts.

Sato, Taisuke, and Tamaki, Hisao. (1984) Enumeration of Success Patterns in Logic Programs. Theoretical Computer Science 34, 227-240.

Shieber, Stuart. (1985a) Evidence against the Context-freeness of Natural Language. Linguistics and Philosophy 8(3), 333-343.

Shieber, Stuart. (1985b). Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, 145-152. University of Chicago, Chicago, Illinois.

Shieber, Stuart. (1986) An Introduction to Unification-Based Approaches to Grammar. Center for the Study of Language and Information.

Xu, Jiyang, and Warren, David S. (1988) A Type System for Prolog. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, 604-619. MIT Press.