Crossed Serial Dependencies: A low-power parseable extension to GPSG

Henry Thompson
Department of Artificial Intelligence
and
Program in Cognitive Science
University of Edinburgh
Hope Park Square, Meadow Lane
Edinburgh EH8 9NW
SCOTLAND

ABSTRACT

An extension to the GPSG grammatical formalism is proposed, allowing non-terminals to consist of finite sequences of category labels, and allowing schematic variables to range over such sequences. The extension is shown to be sufficient to provide a strongly adequate grammar for crossed serial dependencies, as found in e.g. Dutch subordinate clauses. The structures induced for such constructions are argued to be more appropriate to data involving conjunction than some previous proposals have been. The extension is shown to be parseable by a simple extension to an existing parsing method for GPSG.

I. INTRODUCTION

There has been considerable interest in the community lately with the implications of crossed serial dependencies in e.g. Dutch subordinate clauses for non-transformational theories of grammar. Although context-free phrase structure grammars under the standard interpretations are weakly adequate to generate such languages as a^nb^n, they are not capable of assigning the correct dependencies - that is, they are not strongly adequate.

In a recent paper (Bresnan Kaplan Peters and Zaenen 1982) (hereafter BKPZ), a solution to the Dutch problem was presented in terms of LFG (Kaplan and Bresnan 1982), which is known to have considerably more than context-free power. (Steedman 1983) and (Joshi 1983) have also made proposals for solutions in terms of Steedman/Ades grammars and tree adjunction grammars (Ades and Steedman 1982; Joshi Levy and Yueh 1975). In this paper I present a minimal extension to the GPSG formalism (Gazdar 1981c) which also provides a solution. It induces structures for the relevant sentences which are non-trivially distinct from those in BKPZ, and which I argue are more appropriate. It appears, when suitably constrained, to be similar to Joshi's proposal in making only a small increment in power, being incapable, for instance, of analysing $a^nb^nc^n$ with crossed dependencies. And it can easily be parsed by a small modification to the parsing mechanisms I have already developed for GPSG.

II. AN EXTENSION TO GPSG

II.1 Extending the syntax

GPSG includes the idea of compound non-terminals, composed of pairs of standard category labels. We can extend this trivially to finite sequences of category labels. This in itself does not change the weak generative capacity of the grammar, as the set of non-terminals remains finite. GPSG also includes the idea of rule schemata - rules with variables over categories. If we further allow variables over sequences, then we get a real change.

At this point I must introduce some notation. I will write

$$[a,b,c]$$

for a non-terminal label composed of the categories a, b, and c. I will write

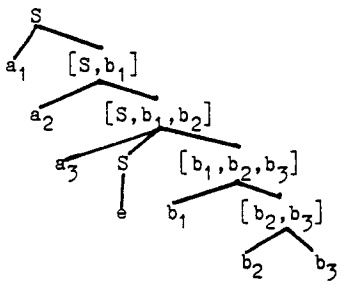
$$Z \blacklozenge b^*$$

to indicate that the schematic variable Z ranges over sequences of the category b. We can then give the following grammar for a^nb^n with crossed

dependencies:

S -> e
S|Z -> a S|Z|b (1)
S|Z -> a S Z|b (2)
b|Z -> b Z (3),

where we allow variables over sequences to appear not only alone, but in simple, that is with constant terms only, concatenation, notated with a vertical bar (|). This grammar gives us the following analysis for a^3b^3, where I have used subscripts to record the dependencies, and the marginal numbers give the rule which admits the adjacent node:



(1)
(1)
(2)
(3)
(3)

With the aid of this example, we see that rule 1 generates a's while accumulating b's, rule 2 brings this process to an end, and rule 3 successively generates the accumulated b's, in the correct, 'crossed', order. This is essentially the structure we will produce for the Dutch examples as well, so it is important to point out exactly how the crossed dependencies are captured. This must come out in two ways in GPSG - subcategorisation restrictions, and interpretation. That the subcategorisation is handled properly should be clear from the above example. Suppose that the categories a and b are pre-terminals rather than terminals, and that there are actually three sorts of a's and three sorts of b's, subcategorised for each other. If one used the standard GPSG mechanism for recording this dependency, namely by providing three rules, whose rule number would then appear as a feature on those pre-terminals appearing in them directly, we would get the above structure, where we can reinterpret the subscripts as the rule numbers so introduced, and see that the dependencies are correctly reflected.

II.2 Semantic interpretation

As for the semantics no actual extension is required - the untyped lambda calculus is still sufficient to the task, albeit with a fair amount of work. We can use what amounts to a packing and unpacking approach. The compound b nodes have compound interpretations, which are distributed appropriately higher up the tree. For this, we need pairs and sequences of interpretations. Following Church, we can represent a pair <1,r> as $\lambda f[f(1)(r)]$. If P is such a pair, then P_0 = $P(\lambda x\lambda y[x])$ and P_1 = $P(\lambda x\lambda y[y])$. Using pairs we can of course produce arbitrary sequences, as in Lisp. In what follows I will use a Lisp-based shorthand, using CAR, CDR, CONS, and so on. These usages are discharged in Appendix I.

Using this shorthand, we can give the following example of a set of semantic rules for association with the syntactic rules given above, which preserves the appropriate dependency, assuming that the b'(a',S') is the desired result at each level:

CONS(CADR(Q')(a')(CAR(Q')),CDDR(Q')) (1)
where Q' is short for S|Z|b',
CONS(CAR(Q')(a')(S'),CDR(Q')) (2)
where Q' is short for Z|b',
ADJOIN(Z',b'). (3)

These rules are most easily understood in reverse order. Rule 3 simply appends the interpretation of the immediately dominated b to the sequence of interpretations of the dominated sequence of b's. Rule 2 takes the first interpretation of such a sequence, applies it to the interpretations of the immediately dominated a and S, and prepends the result to the unused balance of the sequence of b interpretations. We now have a sequence consisting of first a sentential interpretation, and then a number of b interpretations. Rule 1 thus applies the second (b type) element of such a sequence to the interpretation of the immediately dominated a, and the first (S type) element of the sequence. The result is again prepended to the unused balance, if any. The patient reader can satisfy himself that this will produce the following (crossed) interpretation:

$$b'_1(a'_1,b'_2(a'_2,b'_3(a'_3,e'))))).$$

II.3 Parsing

As for parsing context-free grammars with the non-terminals and schemata this proposal allows, very little needs to be added to the mechanisms I have provided to deal with non-sequence schemata in GPSG, as described in (Thompson 1981b). We simply treat all non-terminals as sequences, many of only one element. The same basic technique of a bottom-up chart parsing strategy, which substitutes for matched variables in the active version of the rule, will do the job. By restricting only one sequence variable to occur once in each non-terminal, the task of matching is kept simple and deterministic. Thus we allow e.g. S|Z|b but not Z|b|Z. The substitutions take place by concatenation, so that if we have an instance of rule (1) matching first [a] and then [S,b,b,b] in the course of bottom-up processing, the Z on the right hand side will match [b,b], and the resulting substitution into the left hand side will cause the constituent to be labeled [S,b,b].

In making this extension to my existing system, the changes required were all localised to that part of the code which matches rule parts against nodes, and here the price is paid only if a sequence variable is encountered. This suggests that the impact of this mechanism on the parsing complexity of the system is quite small.

III. APPLICATION TO DUTCH

Given the limited space available, I can present only a very high-level account of how this extension to GPSG can provide an account of crossed serial dependencies in Dutch. In particular I will have nothing to say about the difficult issue of the precise distribution of tensed and untensed verb forms.

III.1 The Dutch data

Discussion of the phenomenon of crossed serial dependencies in Dutch subordinate clauses is bedeviled by considerable disagreement about just what the facts are. The following five examples form the core of the basis for my analysis:

1) omdat ik probeer Nikki te leren Nederlands te spreken

2) omdat ik probeer Nikki Nederlands te leren spreken

3) omdat ik Nikki probeer te leren Nederlands te spreken

4) omdat ik Nikki Nederlands probeer te leren spreken

5) * omdat ik Nikki probeer Nederlands te leren spreken.

With the proviso that (1) is often judged questionable, at least on stylistic grounds, this pattern of judgements seems fairly stable among native speakers of Dutch from the Netherlands. There is some suggestion that this is <u>not</u> the pattern of judgements typical of native speakers of Dutch from Belgium.

III.2 Grammar rules for the Dutch data

This pattern leads us to propose the following basic rules for subordinate clauses:

A) S' -> omdat NP VP
B) VP -> V VP (probeer)
C) VP -> NP V VP (leren)
D) VP -> NP V (spreken).

Taken straight, these give us (1) only. For (2) - (4), we propose what amounts to a verb lowering approach, where verbs are lowered onto VPs, whence they lower again to form compound verbs. (5) is ruled out by requiring that a lowered verb must have a target verb to compound with. The resulting compound may itself be lowered, but only as a unit. This approach is partially inspired by Seuren's transformational account in terms of predicate raising (Seuren 1972).

So the interpretation of the compound labels is that e.g. [V,V] is a compound verb, and [VP,V,V] is a VP with a compound verb lowered onto it. It follows that for each VP rule, we need an associated compound version which allows the lowering of (possibly compound) verbs from the VP onto the verb, so we would have e.g.

Di) VP|Z -> NP Z|V,

where we now use Z as a variable over sequences of Vs. The other half of the process must be

reflected in rules associated with each VP rule which introduces a VP complement, allowing the verb to be lowered onto the complement. As this rule must also expand VPs with verbs lowered onto them, we want e.g.

Cii) VP|Z -> NP VP|Z|V.

Rather than enumerate such rules, we can use metarules to conveniently express what is wanted:

I) VP -> ... V ... ==> VP|Z -> ... Z|V ...

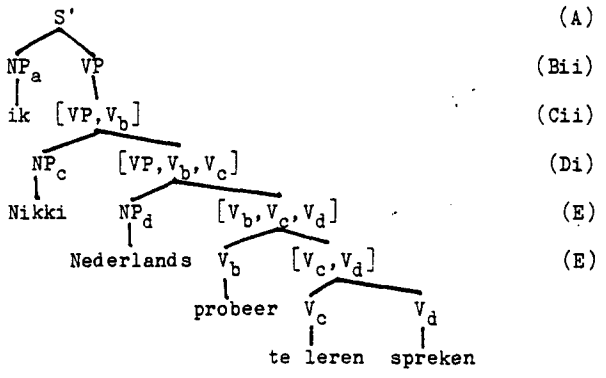
II) VP -> ... V VP ==> VP|Z -> ... VP|Z|V.

(I) will apply to all three of (B) - (D), allowing compound verbs to be discharged at any point. (II) will apply to (B) and (C), allowing the lowering (with compounding if needed) of verbs onto complements. We need one more rule, to unpack the compound verbs, and the syntactic part of our effort is complete:

E) W|Z -> W Z,

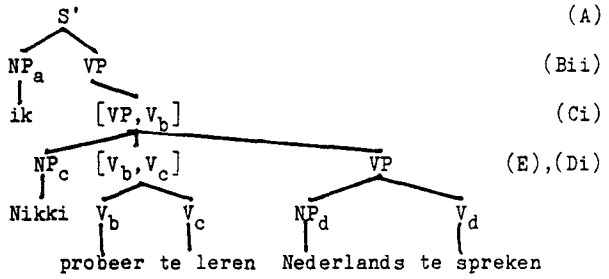
where W is an ordinary variable whose range consists of V. This slight indirection is necessary to insure that subcategorisation information propagates correctly.

By suitably combining the rules (A) - (E), together with the meta-generated rules (Bi) - (Di), (Bii) and (Cii), we can now generate examples (2) - (4). (4), which is fully crossed, is very similar to the example in section II.1, and uses meta-generated expansions for all its VP nodes:



Once again I include the relevant rule name in the margin, and indicate with subscripts the rule name feature introduced to enforce subcategorisation.

Sentences (2) and (3) each involve two meta-generated rules and one ordinary one. For reasons of space, only (3) is illustrated below. (2) is similar, but using rules (B), (Cii), and (Di).



III.3 Semantic rules for the Dutch data

The semantics follows that in section II.2 quite closely. For our purposes simple interpretations of (B) - (D) will suffice:

B') V'(VP')
C') V'(NP',VP')
D') V'(NP').

The semantics for the metarules is also reasonably straightforward, given that we know where we are going:

I') F(V') ==> CONS(F(CAR(Z|V')),CDR(Z|V'))
II') F(V',VP') ==> CONS(F(CADR(Q'),CAR(Q')),
CDR(Q')),

where Q' is short for VP|Z|V'. (I') will give semantics very much like those of rule (2) in section II.2, while (II') will give semantics like those of rule (1). (E') is just like (3):

E') ADJOIN(Z',W')

It is left to the enthusiastic reader to work through the examples and see that all of sentences (1) - (4) above in fact receive the same interpretation.

III.4 Which structure is right - evidence from conjunction

The careful reader will have noted that the structures proposed are not the same as those of BKPZ. Their structures have the compound verb depending from the highest VP, while ours depend from the lowest possible. With the exception of BKPZ's example (13), which none of my sources judge grammatical with the 'voor Marie' as given, I

believe my proposal accounts for all the judgements cited in their paper. On the other hand, I do not believe they can account for all of the following conjunction judgement, the first three based on (4), the next two on (3), whereas under the standard GPSG treatment of conjunction they all fall out of our analysis:

6) omdat ik Nikki Nederlands wil leren spreken en Frans wil laten schrijven

because I want to teach Nikki to speak Dutch and let [Nikki] write French

7) * omdat ik Nikki Nedrelands wil leren spreken en Frans laten schrijven

8) omdat ik Nikki Nederlands wil leren spreken en Carla Frans wil laten schrijven

because I want to teach Nikki to speak Dutch and let Carla write French.

9) omdat ik Nikki wil leren Nederlands te spreken en Frans te schrijven

because I want to teach Nikki to speak Dutch and to write French

10) * omdat ik Nikki wil leren Nederlands te spreken en Carla Frans te schrijven
or
... en Frans (te) laten schrijven

(6) contains a conjoined [VP,V,V], (8) a conjoined [VP,V], and (7) fails because it attempts to conjoin a [VP,V,V] with a [VP,V]. (9) conjoins an ordinary VP inside a [VP,V], and (10) fails by trying to conjoin a VP with either a non-constituent or a [VP,V].

It is certainly not the case that adding this small amount of 'evidence' to the small amount already published establishes the case for the deep embedding, but I think it is suggestive. Taken together with the obvious way in which the deep embedding allows some vestige of compositionality to persist in the semantics, I think that at the very least a serious reconsideration of the BKPZ proposal is in order.

IV. CONCLUSIONS

It is of course too early to tell whether this augmentation will be of general use or significance. It does seem to me to offer a reasonably concise and satisfying account of at least the Dutch phenomena without radically altering the grammatical framework of GPSG.

Further work is clearly needed to exactly establish the status of this augmented GPSG with respect to generative capacity and parsability. It is intriguing to speculate as to its weak equivalence with the tree adjunction grammars of Joshi et al. Even in the weakest augmentation, allowing only one occurence of one variable over sequences in any constituent of any rule, the apparent similarity of their power remains to be formally established, but it at least appears that like tree adjunction grammars, these grammars cannot generate $a^n b^n c^n$ with both dependencies crossed, and like them, it can generate it with any one set crossed and the other nested. Neither can it generate WW, although it can with a sequence variable ranging over the entire alphabet. If it can be shown that it is indeed weakly equivalent to TAG, then strong support will be lent to the claim that an interesting new point on the Chomsky hierarchy between CFGs and the indexed grammars has been found.

ACKNOWLEDGEMENTS

The work described herein was partially supported by SERC Grant GR/B/93086. My thanks to Han Reichgelt, for renewing my interest in this problem by presenting a version of Seuren's analysis in a seminar, and providing the initial sentential data; to Ewan Klein, for telling me about Church's 'implementation' of pairs and conditionals in the lambda calculus; to Brian Smith, for introducing me to the wonderfully obscure power of the Y operator; and to Gerald Gazdar, Aravind Joshi, Martin Kay and Mark Steedman, for helpful discussion on various aspects of this work.

APPENDIX I
SEQUENCES IN THE UNTYPED LAMBDA CALCULUS

To imbed enough of Lisp in the lambda calculus for our needs, we require not just pairs, but NIL and conditionals as well. Conditionals are implemented similarly to pairs - "if p then q else

r" is simply p applied to the pair <q,r>, where TRUE and FALSE are the left and right pair element selectors respectively. In order to effectively construct and manipulate lists, some method of determining their end is required. Numerous possibilities exist, of which we have chosen a relatively inefficient but conceptually clear approach. We compose lists of triples, rather than pairs. Normal CONS pairs are given as <TRUE,car,cdr>, while NIL is <FALSE,,>.

Given this approach, we can define the following shorthand, with which the semantic rules given in sections II.2 and III.3 can be translated into the lambda calculus:

```
TRUE - λx.[λy.[x]]
FALSE - λx.[λy.[y]]

NIL - λf.[f(FALSE)(λp.[p])(λp.[p])]
CONS(A,B) - λf.[f(TRUE)(A)(B)]

CAR(L) - L(λx.[λy.[λz.[y]]])
CDR(L) - L(λx.[λy.[λz.[z]]])

CONSP(L) - L(λx.[λy.[λz.[x]]])

CADR(L) - CAR(CDR(L))

ADJOINFORM - λa.[λL.[λN.[
                CONSP(L)(CONS(CAR(L),
                              a(CDR(L))(N)))
                        (CONS(N,NIL))]]]

Y - λf.[λx.[f(x(x))](λx.[f(x(x))])]

ADJOIN(L,N) - Y(ADJOINFORM)(L)(N)
```

Note that we use Church's Y operator to produce the required recursive definition of ADJOIN.

REFERENCES

Ades, A. and Steedman, M. 1982. On the order of words. Linguistics and Philosophy. to appear.

Bresnan, J.W., Kaplan, R., Peters, S. and Zaenen, A. 1982. Cross-serial dependencies in Dutch. Linguistic Inquiry 13.

Gazdar, G. 1981c. Phrase structure grammar. In P. Jacobson and G. Pullum, editors, The nature of syntactic representation. D. Reidel, Dordrecht.

Joshi, A. 1983. How much context-sensitivity is required to provide reasonable structural descriptions: Tree adjoining grammars. version submitted to this conference.

Joshi, A.K., Levy, L.S. and Yueh, K. 1975. Tree adjunct grammars. Journal of Computer and System Sciences.

Kaplan, R.M. and Bresnan, J. 1982. Lexical-functional grammar: A formal system of grammatical representation. In J. Bresnan, editor, The mental representation of grammatical relations. MIT Press, Cambridge, MA.

Seuren, P. 1972. Predicate Raising in French and Sundry Languages. ms., Nijmegen.

Steedman, M. 1983. On the Generality of the Nested Dependency Constraint and the reason for an Exception in Dutch. In Butterworth, B., Comrie, E. and Dahl, O., editors, Explanations of Language Universals. Mouton.

Thompson, H.S. 1981b. Chart Parsing and Rule Schemata in GPSG. In Proceedings of the Nineteenth Annual Meeting of the Association for Computational Linguistics. ACL, Stanford, CA. Also DAI Research Paper 165, Dept. of Artificial Intelligence, Univ. of Edinburgh.