Published in: UEA Papers in Linguistics, No 23, 1985, pp 21-38, Norwich, UK.

INDEXED GRAMMARS AND INTERSECTING DEPENDENCIES

S.G.Pulman,
Computer Laboratory,
University of Cambridge.

G.D.Ritchie,
Dept. of Artifial Intelligence,
University of Edinburgh.

Introduction

It has been pointed out many times (Chomsky 1963, Postal 1964, etc.) that a language that exhibits syntactic dependencies which are not properly nested, (i.e. take the form in 1), cannot be described adequately using simple context-free phrase structure grammars.

```
1  a1  a2  a3... an   b1  b2  b3... bn
   |   |   |     |    |   |   |     |
   -------------------    |   |     |
       |   |     |        |   |     |
       -------------------    |     |
           |     |            |     |
           -------------------      |
                 |                  |
                 -------------------
```

More recently, there has been renewed interest in this issue, since generalised context-free grammars are being proposed as adequate descriptive models of natural language syntax (see e.g. Gazdar 1982). However, Dutch is widely believed (Bresnan et al 1982 and refs there) to display exactly the kind of intersecting (crossed, or non-nested) dependencies sketched in 1, in the form of sequences of NP constituents followed by sequences of verbal items which pair up as in 2:

2 NP1 NP2 ... NPn V1 V2 ... Vn

Several Scandinavian languages also seem to allow for multiple intersecting dependencies caused by Wh-movement (Maling and Zaenen 1982) giving rise to sequences with similar formal properties.

Such dependencies can be handled by indexed grammars (Aho, 1968), and in the following sections we sketch how indexed grammars work, illustrating with an English example. Next we go on to suggest how a system of indexing could be incorporated into a generalised phrase structure grammar to cope with the intersecting dependencies found in Dutch. This analysis (as with the others) is intended to be illustrative and exploratory rather than definitive, and would require much further work before being offered as a serious account of the complex facts about these constructions.

Indexed Grammars

An indexed grammar consists of:

(a) a set N of non-terminal symbols

(b) a set T of terminal symbols

(c) a set I of indices

(d) a set P of basic rules, where each rule in P is of the form

A -> X

where X is a sequence of terminal symbols, non-terminal symbols and indices, such that each non-terminal symbol is immediately followed by zero or more indices, and no indices appear in X except in these clusters adjacent to non-terminal symbols. For example, if A,B,C,D are in N; a,b, c are in T and i,j,k,l are in I, then

C -> AijBljbcD

would be a permissible form for a basic rule.

(e) for each i in I, a set IRi of indexed rules, where each indexed rule is of the form

A -> X

where A is in N and X is a string of terminal and non-terminal symbols, possibly empty. In other words, indexed rules are like ordinary context-free rules, and in particular they cannot introduce indices.

(f) a particular non-terminal S in N is the start symbol.

The language generated by a given indexed grammar is the set of strings of terminal symbols which can be derived from the start symbol by repeated applications of rules, where rule application is defined in the following way. For both basic rules and indexed rules, application can be regarded as occurring in two stages - "expansion" and "distribution". Expansion is the simple rewriting of a single non-terminal symbol (on the left of the rule) as a sequence of symbols (the right hand side of the rule). Distribution is the insertion, within the newly rewritten sequence, of multiple copies of a cluster of indices which previously lay on the right of the expanded symbol.

For a basic rule, rule application is as follows. Suppose there is a basic rule

A -> X<1> y<1> ... X<n> y<n>

where each X<i> is either in N or T and each y<i> represents a sequence of indices. Any occurrence of A in a string can be replaced by the given sequence of X<i> y<i> s, but any sequence of indices which are situated immediately to the right of the occurrence of A must be appended to the right of every Y<i> which follows a non-terminal X<i> . That is,

Az

(where A is in N and z is the longest sequence of indices next to A) can be rewritten as

X<1> y<1>z X<2> y<2>z .... X<n> y<n>z

with the proviso that where X<j> is a terminal symbol, no indices follow it (i.e. Y<j> is empty and z is not appended).

An indexed rule can be applied only if its left hand side symbol occurs immediately before that index in a string. In that case, the symbol can be expanded as above, and any further indices which lay to the right of the one associated with the rule used must be distributed along the rewritten sequence, as above. That is, suppose that k is an index, and IR<k> contains the rule

B -> AjiBjjjcdDli

where A, B, D are in N, c,d are in T, and i,j,k,l in I. Then if a string contains the sequence

Bk

it can be expanded as AjiBjjjcdDli

Moreover, any sequence of adjacent indices lying immediately to the right of k before this expansion must be distributed along the rewritten sequence, in the manner described above. Hence

BkijjlDC

would rewrite as

AjiijjlBjjjijjlcdDliijjl

Another way to look at the process of rule application is to regard indices as being attached to non-terminals (although still as a strictly ordered sequence, not as an unordered set). Then the application of a basic rule is simply:

> Rewrite the left hand symbol as the right hand side sequence, attaching to each non-terminal therein the set of indices which were attached to the rewritten symbol.

And the application of an indexed rule is:

> A non-terminal symbol which has a sequence of indices i<1> ... i<m> attached to it may be rewritten using any rule associated with i<1>, and the index sequence i<2> ... i<m> is attached to each non-terminal in the rewritten form.

A useful way to think of the process informally is to imagine each non-terminal as having its own stack, or last-in-first-out store, for indices. Indices can only be added to the top of the stack, which is represented by the index immediately to the right of the non-terminal in the above examples and definitions, and indexed rules can only be applied by popping the top index of the stack, applying a rule from the corresponding subgrammar, and carrying along any remaining indices in that application of the rule, or of any subsequent basic rules.

Number Agreement

By way of illustration, we will give an indexed grammar to handle some sentences displaying agreement not just between subject and verb, but between subject and predicate: the ‘predicate nominal’ construction shown in:

```
3 a John is a doctor
  b *John is doctors
  c *Bill are doctors
  d *Those men are a doctor
  e Those men are doctors
```

We will assume that the facts of agreement here are no more complicated than this, which is of course not entirely true. It is also the case that the construction can be treated adequately by a less powerful type of grammar.

An indexed grammar which will account for examples like those in 3 is:

```
4 N = { E, S, NP, VP, V, N, Name, Det }

  T = { John, Bill, man, men, doctor, doctors, is, are }

  I = { s, p }

  P = { E-> Ss, E-> Sp, S-> NP VP, VP-> V NP, NP-> Det N, Name-> Bill,
        Name-> John }

 IR<s> = { V-> is, Det-> a, NP-> Name, N-> doctor, N-> man }

 IR<p> = { V->are, Det-> those, Det-> 0, N-> men, N-> doctors }

 Start symbol = E
```

Examples of the structures generated by this grammar are:

```
5 a         E
            |
            S.s
          /   \
        NP.s    VP.s
        /      /   \
    Name      V.s   NP.s
      |       |    /   \
      |       |  Det.s  N.s
      |       |   |      |
    John     is   a   doctor
```

```
5 b          E
             |
             S.p
           /     \
         NP.p      VP.p
         /  \       /   \
     Det.p  N.p    V.p   NP.p
       |     |      |    /   \
       |     |      |  Det.p  N.p
       |     |      |   |      |
    those  men    are   0    doctors
```

(Dots have been inserted between the non-terminals and their indices for readability - they are not part of the formalism.)

The indices denoting agreement are introduced by the expansions of the start symbol E, and passed down by the application of the basic rules for S, VP etc. from P. The pre-terminal or lexical category symbols are all rewritten by rules from the appropriate indexed subgrammar, consuming the index that appears on them.

Aside from such oddities as:

6 ?a man is Bill

the grammar in 4 will only generate good English: in particular, it ensures that everything that should agree does agree.

Intersecting dependencies

As a preliminary to the treatment of Dutch we will show how an indexed grammar can be used to generate the language consisting entirely of strings of the form XX, (i.e. some sequence of terminal symbols repeated exactly), since this is the standard example of a language which cannot be generated by a context-free grammar. Then we will combine the ideas from these illustrative examples to show how non-nested dependencies in Dutch or other languages could be generated.

Consider the indexed grammar:

```
7  N = { S, Y }
   T = { a, b }
```

```
   I = { i, j }
   P = { S-> Si, S-> Sj, S-> YY }
 IR<i> = { Y-> aY, Y-> a}
 IR<j> = { Y->bY,  Y-> b}
 Start Symbol = S
```

This grammar will generate the language consisting of any repeated sequence of (one or more) as and bs. The idea is to build up a pattern of i and j indices to the right of S, then expand S as YY, when each Y will inherit an identical pattern of indices. Subsequent expansions of each Y merely manifest the identical sequences of indices as identical strings of a and b symbols.

Some sample derivations permitted by this grammar are:

```
8 a     S                      b          S
        |                                 |
        S.j                               S.j
        |                                 |
        S.ij                             S.ij
       / \                                |
    Y.ij   Y.ij                          S.iij
    / \   / \                             |
   a Y.j a  Y.j                          S.jiij
      |     |                             /  \
      b     b                        Y.jiij   Y.jiij
                                      /   \     /  \
                                     b  Y.iij  b  Y.iij
                                          / \      / \
                                        a  Y.ij   a  Y.iij
                                           / \       / \
                                          a  Y.j    a  Y.j
                                               |         |
                                               b         b
```

These sample derivations illustrate one aspect of this style of grammar which is relevant to the fragment we will develop below. In using indexed grammars to encode dependencies, it is often the case that a number of basic rules have to be used purely in order to build up a sequence of indices, as in the derivations in 8a. and b. During this stage, no essential constituent structure is being generated, and it is inelegant and unnecessary to regard these rule-applications as producing a set of non-branching nodes all with the same non-terminal label. We can adopt a convention that any such non-branching sequence can be collapsed to a single pair of nodes, in which the lower (daughter) node has the full sequence of indices attached to it. If we adopted it for the above sample grammars, then, for example, the top of the tree in 8b. above would be regarded, for linguistic purposes, as having the form:

```
9           S
            |
          S.jiij
           / \
          /   \
      Y.jiij   Y.jiij
      /    \  /  \
    .... etc  ......
```

A further possible notational simplification would be to prune out the topmost of these two nodes with identical labels, leaving just the node with the index sequence. In effect, these conventions say that, if a grammar contains (basic) rules:

```
W -> Wi<1>,
......
W -> Wi<n>
```

for any non-terminal W and any set of indices i<1>,...i<n>, these rules can be thought of as a rule schema stating that any node labelled W can be have any sequence of indices from {i1,...in} attached to it.

Dutch

The facts of Dutch have been discussed extensively in recent years and there is much disagreement over the acceptability of crucial examples and over the constituent structures which should be assigned even to the undisputed examples. We will base our account on the discussion to be found in Bresnan, Kaplan, Peters, and Zaenen (1982) (henceforth BKPZ), and we will assume that the facts they present, and the structures assigned to their examples, are correct (with one exception to be mentioned later).

BKPZ are concerned to argue that, although the string set of Dutch might be generable by a context free grammar, making Dutch a weakly context free language, it is not a strongly context free language. That is, the trees assigned to the strings by such a grammar would not be trees that were consistent with the other syntactic properties of the sentences and arguably not such as to support a coherent semantic interpretation.

The examples at issue are sentences like:

10 a ... dat Jan de kinderen zag zwemmen
... that Jan saw the children swim

b ... dat Jan Piet de kinderen zag helpen zwemmen
... that Jan saw Piet help the children swim

c ... that Jan Piet Marie de kinderen zag helpen laten zwemmen
... that Jan saw Piet help Marie make the children swim

The verbs like 'help' that can appear in such frames can be repeated and thus a potentially infinite number of such sentences exists, subject to the usual caveats about implausibility, unprocessability etc.

However, in one respect this subset of Dutch is not strictly analogous to the artifial language illustrated by the grammar of 7. Given a string like

11 ... NP1 NP2 ... NPn NPn+1 V1 V2 ... Vn

there is a syntactic dependency between NP1 and V1, which must agree in number and person, and between NPn+1 and Vn: Vn must be the type of verb which can have NPn+1 as an argument. In the examples in 10 Vn is intransitive and there is no NPn+1 present. But if Vn were transitive there would have to be another NP after NPn for this requirement to be satisfied; thus, essentially, the requirement is that the subcategorisation properties of the last verb should be satisfied by the last group of NPs before the verb group. An example with a transitive verb is:

12 ... omdat ik Cecilia Henk de nijlpaarden zag helpen voeren
... because I saw Cecilia help Henk feed the hippos

(from Steedman 1983).

Apart from these requirements, however, there are no other formally marked dependencies between the Nps and the Vs taken individually: they can come in any order which respects the two formal dependencies. So a sentence like:

13 ... dat Jan Marie de nijlpaarden de kinderen zag laten helpen zwemmen

is just as acceptable as the others, in theory. There are of course pragmatic constraints on plausibility, but these are not encoded formally. Thus since in terms of the set of strings of terminal elements of these examples there are only two dependencies, a context free grammar, technically speaking, can generate them. BKPZ in fact provide one which does so, capturing the two dependencies using some of the apparatus made available in GPSG.

However, BKPZ argue that the tree structures to be associated with this type of sentence must have the properties illustrated by the one here:

```
14           S
          /        \
         /          \
     NP               VP
     |              / |     \
                  /   |         \
     |        NP      VP             V'
              /     /    \         /    \
     |       /    NP     VP     V          V'
            /    /       |                /  \
     |     |    /        NP     |      V        V'
                                      /       /     \
     |     |    |        |      |    |     V          V'
                                                      |
     |     |    |     /  |      |    |     |          V

   Jan  Piet Marie de kinderen zag helpen laten  zwemmen
```

rather than the type of tree given by the CFG they provide:

```
15
               S
            /  |  \
         NP    S    V
            /  |  \
          NP   S   V
             / | \
           NP  S  V
              / \
             NP  V
```

If their arguments in favour of such a structure are sound, then this fragment of Dutch cannot be strongly generated by a CFG. Although the tree in 13 can be generated by the simple grammar:

```
16  S -> NP VP
   VP -> (NP)(VP)(V')
   V' -> V (V')
```

it will also be possible to generate structures with an unmatched set of NPs and Vs, since the two recursive rules are independent of each other. This being so, there is no guarantee that each will be applied just the right number of times to match the output of the other. In BKPZ's framework this is of no consequence, for the constituent structures generated by such rules are all required to meet a principle of 'functional coherence', which in effect act to filter out derivations in which verbs cannot be matched up with the appropriate number and type of argument. Thus the grammar taken as a whole will not overgenerate, although the PS rules taken in isolation would.

In fact, the rules given by BKPZ will overgenerate in another way too: V' can occur inside both VP constituents, permitting derivations like:

```
17               S
               /     \
          NP         VP
                    /|  \
                   / |     \
                NP   VP      V'
                    / | \    |  \
                  NP VP V'   V   V'
                       / \       .
                       V  V'     .
                          .      .
                          .
                          .
```

Such structures would also, presumably, be ruled out by the various functional coherence requirements of their framework. Notice too that the VP rule in 15 has an extremely curious feature: one of the rules which it abbreviates allows for VPs which at no time need contain a V, thus, one assumes, violating every possible version of X bar theory.

In any phrase structure treatment of these examples from Dutch, then, there are three problems to be solved: ensuring agreement between the first NP and the first V; ensuring that the subcategorisation requirements of the last V are met; and ensuring that the number of NPs and Vs is correctly matched. In the analysis to be given, the first of these appears to follow directly from the existing GPSG machinery of feature handling, whereas possibly the second, and certainly the third require the machinery of an indexed grammar.

Let us assume that a GPSG treatment of the rest of Dutch is available (!). This will include rules expanding VPs whose verbs are those like 'make' and 'see' etc., which permit the construction in question, and those like 'feed' and 'swim', which will be assumed to behave otherwise like their English equivalents (obvious word order differences apart). The grammar will therefore contain rules like:

```
18 < y, VP -> V .... >
             [y]
```

where y stands for any of the rule numbers which the GPSG subcategorisation mechanism assigns as features to the verbs other than ‘make, see’ etc., and rules like:

```
19 < x, VP -> V S’ ... >
             [x]
```

for ‘make’ and ‘see’, etc, on the assumption that these can take sentential complements under normal circumstances. (Niceties concerning ID/LP, semantics, other features etc. are suppressed here).

The basic idea behind the indexed extension to GPSG that we will introduce is to enlarge the role of rule numbers in the overall grammar. These are already permitted to appear, by convention, as features on the verbs they introduce, in order to capture the subcategorisation properties of these verbs etc. We shall have them act also as indices of the type illustrated earlier, in a way which is closely linked to the VP rule that they label. That is to say, if there is a rule in the grammar, for example:

```
20 < a, VP  -> V   NP ... >
              [a]
```

then there is a corresponding index "a". Furthermore, we shall insert in the grammar a metarule which states that for every such rule-feature-cum-index there is a rule of the form:

```
21  S -> NP VPa
```

which will produce trees of the form:

```
22      S
      /   \
     NP   VPa
```

(Alternatively, a similar effect could be achieved by adding rules like VP -> VPa, etc. to the grammar, and relying on the convention introduced above for collapsing derivations into trees).

Of course, until we have an indexed subgrammar IR<a>, there will be no effect: the indices will be passed down the tree and disappear harmlessly when the preterminal symbols are expanded (recall that indices can only appear next to non-terminals). The form of the sets IR<a> is described later.

Given this basic mechanism, the treatment of intersecting dependencies proceeds as follows. For each rule in the grammar of the form:

```
23 < y, VP -> V     X >
             [+y]
```

where y as before stands for the rule numbers of any rule introducing verbs other than ‘make’, ‘see’, etc, there is to be a corresponding indexed rule set:

```
24  IR<y> = { (i) VP/V -> X, (ii) V' -> V    }
                                     [+y]
```

This does not require anything more powerful than the existing metarule formalism. Rule (i) of the pair in IR<y> introduces, via the variable X (instantiated in a real example, of course), the complement of $V_{[+y]}$, dominated by a new category VP/V - a VP missing a V. The second rule, (ii), merely reintroduces the original V as a member of the constituent V’.
So far, the rules in 24 are useless, as the occasion for them applying cannot yet arise.

We now add to the grammar a schema to give rules of the form:

```
25 VP -> VP/Vx V'x, VP -> VP/Vxx  V'xx, etc
```

i.e. a rule introducing a VP/V category with some number of x indices and a V’ category with the same number of x indices. Intuitively x corresponds to the subcategorisation feature/number that we assume to be shared by all the verbs like ‘make’ and ‘see’ etc. that can appear in this construction. The rules produced by this schema are what ultimately provide the environment for those in 24. Notice that such a schema, while clearly very powerful, is still not going beyond what is permitted by the indexed grammar formalism: indeed, it is simply a shorthand way of expressing what could be achieved by adding a pair of rules:

```
26 VP -> VPx,
   VP -> VP/V V'
```

to the grammar, and avoiding the redundancy of tree structure this would produce by relying on the conventions introduced at the end of the previous section for collapsing non-branching derivations.

The associated indexed rule set for the x index is:

```
27  IR<x> = { (i) VP/V -> NP VP/V,
             (ii) V' -> V   V'     }
                        [x]
```

This is not produced by a metarule, but is part of the basic grammatical description of the language.

Sample derivations produced by this extended GPSG are as follows: the first one is annotated to show which rules were involved. As in the earlier artificial examples, indices sufficient to generate the requisite dependencies are introduced and then consumed by the application of rules from the corresponding indexed sets.

```
28a        .
           .
           .
           S
       /       \
   NP           VPy (25)
                 /        \
   |            /          \
          VP/Vxy (27i)    V'xy (27ii)
   |        /   \           | \
           /     \          |  \
  Jan    NPy   VP/Vy (24i)  V    V'y (24ii)
                           [x]   |
        |       |           |    |
     Marie      0          saw   V
                                [y]
                                 |
                                swim
```

b

```
                  .
                  .
                  .
                  S
               /     \
      NP                 VPy
                        /     \
      |      VP/Vxxy              V'xxy
               /     \             |    \
      |   NPxy      VP/Vxy        V     V'xy
                     /   \       [x]   /  \
      |   |     NPy    VP/Vy      |   V      V'y
                 |       |           [x]     |
      |   |      |       NP       |   |      V
                                            [y]

      I  Marie  Henk  the hippos saw  help  feed
```

Several comments are needed: firstly, note that these structures are not quite the same as those in BKPZ, as the first VPy produces a binary, not a ternary branching structure. This is just to make matters more simple and uniform, and we take no stand on whether it is either necessary or correct: there seems to be immense disagreement over the correct constituent structure of these examples. Secondly, the node VP/Vy expands as null in 28a because ‘swim’ takes no obligatory complements in the VP rule from which this indexed set was produced. Probably the X in 22 and 24i should only range over obligatory complements, optional ones appearing instead in 24ii, after V. In 28b, on the other hand, since ‘feed’ is transitive, VP/Vy expands as its complement NP. Thirdly, some extraneous indices remain on NPs: these are artefacts of the formalism and can be assumed to be harmless, or stipulated away.

The indexed rule sets IRy derived from VP rules ensure that the subcategorisation facts are handled correctly, via the mediation of the rule schema in 25. This schema also ensures that the numbers of NPs and Vs matches up correctly via the matching sets of indices on the right hand side. The agreement which is found between the first NP and the first V, however, is captured automatically if we assume that the Control Agreement Principle and the Head Feature Convention (Gazdar and Pullum 1982) apply here. The effect of these two conditions is to ensure that the topmost VP agrees with the subject (CAP) and that the agreement features percolate down (HFC) to the first verb (even though the index gets spelled out as a feature on the final verb, via IRy), as illustrated:

29

```
             S
          /     \
    NP           VPy    ------>
 [z num] -CAP-> [z num]        |HFC
                 /   \     \    |
               NP    VP/V    V' |
                           [z num]
                     .     /  \
                     .    V      V'
                     . [z num]    \
                                   .
                                   .
                                   .
```

This completes our short demonstration of the applicability of indexed grammars to the description of natural languages. As stressed earlier, our goals have been illustrative rather than innovative; in particular, we are not offering the above as a serious analysis of Dutch intersecting dependencies. Nevertheless, it appears that GPSGs need to be augmented with something rather similar to the mechanisms made available by indexed grammars if they are to be able to describe such languages adequately, and the foregoing treatment may possibly suggest the lines a more complete analysis should take.

FOOTNOTE

This paper is very much a working paper and dates from Easter 1983: it takes no account of developments in GPSG since about 1982.

BIBLIOGRAPHY

Aho, A. 1968 'Indexed grammars - an extension of context free grammars', Journal of the ACM 15, 647-671

Bresnan, J., Kaplan, R., Peters, S., and Zaenen, A. 1982 'Cross-serial dependencies in Dutch', Linguistic Inquiry 13, 613-635.

Chomsky, N. 1963 'Formal properties of grammars', in Handbook of Mathematical Psychology vol II , ed R.D.Luce,R. Bush and

E. Galanter; New York, Wiley.

Gazdar, G. 1982 ‘Phrase structure grammar’ in Jacobson and Pullum

Gazdar, G. and Pullum, G.K. 1982 ‘Generalised Phrase Structure Grammar - a theoretical synopsis’, Indiana University Linguistics Club.

Jacobson, P. and Pullum G.K. (eds) 1982 The Nature Of Syntactic Representation, Dordrecht: D. Reidel Publishing.

Maling, J. and Zaenen, A. 1982 ‘Scandinavian Extraction Phenomena’, in Jacobson and Pullum

Steedman, M. 1983 ‘On the generality of the nested dependency constraint and the reason for an exception in Dutch’, in B. Butterworth, B. Comrie, and O. Dahl (eds) Explanations of Language Universals, Mouton Publishing, forthcoming.

Thompson, H 1983 ‘Crossed serial dependencies: a low power parseable extension to GPSG. DAI 193 technical report, Dept of Artificial Intelligence, Edinburgh.