English Auxiliaries without Trees or Transformations

Ian. C. Stirk

Introduction

This paper contains a mixture of the formal and the informal : in order to make it more readable, I have confined the most formal parts to an appendix entitled "Proofs". The reader is invited to look at the appendix at various points in the text. Acceptance of the invitation is of course voluntary.

My word processing program gets very crotchety over the matter of bars and double bars, but it will cheerfully do crossings out and double crossings out, hence the use of "~~N~~" for "N bar" and "₦" for "N double bar", etc.

Context freeness and Treelessness

In previous work (Stirk 1999) I showed that a certain set of languages, which I called "treeless", could be generated as the intersections of finite state languages and a finite number of simple context free languages which I designated "bracketing languages". These treeless languages seemed potentially useful from the linguistic point of view, as they include not only context free languages but also certain context sensitive ones, and some human languages may well be context sensitive.

Since then, I have become aware of work done at the Turku Centre for Computer Science, exploring the field of "mildly context sensitive languages" (Martin-Vide, Mateescu and Salomaa, 1999). On page 1 of this work, the three authors include the following condition for a family of languages $\mathbb{L}$ to be mildly context sensitive :

" $\mathbb{L}$ contains the following three non-context-free languages :

-multiple agreements :

$$L_1 = \{a^i b^i c^i \mid i \geq 0\},$$

-crossed agreements :

$$L_2 = \{a^i b^j c^i d^j \mid i, \ j \geq 0\}, \text{ and}$$

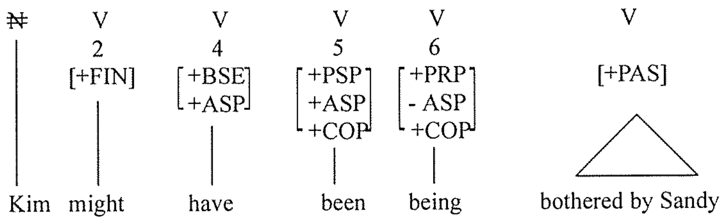
-duplication :

$$L_3 = \{ww \mid w \varepsilon \ \{a, b\}^*\}."$$

The family of treeless languages certainly includes multiple agreements and crossed agreements, but not duplication (Proof 1). Presumably, then, the treeless family is somewhat milder than mild. Nevertheless, it may have its uses.

In the present paper I want to make a start on showing how the context free syntactic rules of Generalised Phrase Structure Grammar may be translated into treeless form. It is most convenient to begin with an early form of GPSG, namely Gazdar, Klein, Pullum and Sag (1999; henceforth ARP). Here can be found a more straightforward and pure form of the basic ideas, and there is no need to disentangle the dreadful set-theoretic muddle of Gazdar, Klein, Pullum and Sag (1985) or Pollard and Sag (1994).

On page 601 of ARP is a tree to illustrate the operation of the phrase structure rules. If we just strip the leaves from the tree and forget about the rest, then presumably we will have the kind of structure that a treeless grammar should provide :



The next steps must be to show how to obtain such a flat structure by means no more powerful than a finite state grammar and some bracketing grammars, and also to replace the information lost in the felling of the original ARP tree.

It seems most convenient to continue to consider categories as no more than non-terminal symbols written in a more perspicuous manner. In that case, the leaf diagram above could be regarded as an expression of a derivation according to a finite state grammar as follows :

$$\text{₦} \longrightarrow \text{Kim} \begin{bmatrix} \text{V} \\ 2 \\ \text{+FIN} \end{bmatrix}$$

$$\begin{bmatrix} \text{V} \\ 2 \\ \text{+FIN} \end{bmatrix} \text{might} \begin{bmatrix} \text{V} \\ 4 \\ \text{+BSE} \\ \text{+ASP} \end{bmatrix}$$

and so on.

It would be even more straightforward to "generate" the leaf structure more directly by using statements like the following :

$$[₦] \quad < \quad \begin{bmatrix} \text{V} \\ 2 \\ \text{+FIN} \end{bmatrix}$$

$$\begin{bmatrix} \text{V} \\ 2 \\ \text{+FIN} \end{bmatrix} \quad < \quad \begin{bmatrix} \text{V} \\ 4 \\ \text{+BSE} \end{bmatrix}$$

The first of these is intended to mean that any category containing the features and values associated with "₦" can appear immediately to the left of one with the features and values going with "V", "2" and "+FIN". Doing this does not add anything that is not finite state in power (Proof 2). Also we need some "immediate dominance of terminal symbol" statements, which could be written as follows :

₦ | Kim, V [2] | might, etc.

Statements such as these would sanction leaf diagrams like the one above, again with no increase in power beyond finite state (Proof 2).

In fact, the immediate precedence statements above would be much too restrictive, for they could *only* be used in the generation of sentences like "Kim must have been being bothered by Sandy", sentences with a modal verb *and* perfect aspect and continuous forms. We get a lot further by loosening immediate precedence statements to such as the following :

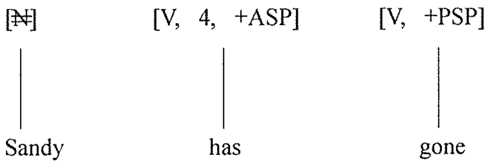
[+N] < [V]

[2, +FIN] < [+BSE]

[3,+FIN] < [+BSE, -AUX] etc.

Here I can write "etc" without qualms, as these immediate precedence statements can be read off from the rules given in ARP, in their Table 1 on page 599.

These looser statements will sanction many more leaf diagrams and sentences, such as the one below :



The precedence statements only mention certain features of categories : the others may be filled in *ad lib*, provided only that feature cooccurrence restrictions and defaults are observed. These latter two items can be carried over directly from phrase structure grammars to treeless ones. The extra looseness causes no problems for finite statehood : see Proof 2 once more.

Notice a great simplification made by treelessness here : simple immediate precedence takes over from the combination of rule and Head Feature Convention used in ARP and generally in phrase structure grammars. Of course there could not possibly be a Head Feature Convention in a treeless grammar, as the concept is meaningless without a tree structure. Heads can only be recognised, if it is necessary to do so, by the contents of a single category.

Schemata and Metarules

ARP tackles VP fronting on page 603, and this provides a good example for introducing a bracketing component into the treeless forms. The fronted verb form must have the feature values [-FIN, -INF, -ASP]. We are going to use the "bracketing features" y and y' for this fronting, so the "ordinary" FCR

[y] ⊃ [-FIN, -INF, -ASP]

will be needed. A different kind of FCR will also be necessary, for which I cannot as yet think of a suitable notation. Anyhow, the idea is to correlate the contents of two different categories, so that if one contains [y, αC] then the other contains [y', αC], where α ranges over values and C over features. The result will be to ensure that if there is a category containing y and some other stuff, there will be a category containing y' and the same other stuff. The following immediate precedence statement will then introduce fronted VPs :

[y] < [+N]

Or at least fronted verb forms. Sentences like the ARP page 603 example, "going he will be", will certainly work out as required :

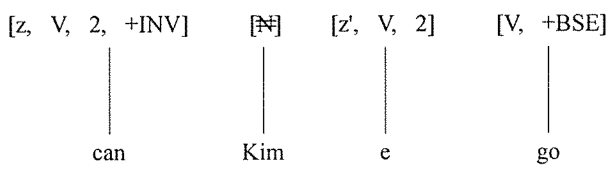
```
[y, V, +PRP]    [+N]    [V, 2, +FIN]    [V, 5, +BSE, +ASP, +COP]    [y', V, +PRP]
     |            |          |                     |                      |
   going          he        will                   be                     e
```

On the other hand, it is not so clear how sentences with VPs longer than a single item would be dealt with : "giving orders he will be", for instance. One drastic effect of treelessness is that the idea of a *phrase* becomes problematic. Tree structures provide an extra kind of glue for sticking words together which is no longer available. I will set aside this problem for further work. Meanwhile, Proof 3 shows that the new procedures above do not violate the basic principles of finite statehood or bracketing.

The treatment of subject-auxiliary inversion in ARP (page 608) is decidedly one of the most attractive points in that paper. Again it depends on being able to pick out whole phrases, and I am still unsure how it should best be tackled in a treeless framework. That it can be tackled there is no doubt. One possibility is to use the same trick as for VP fronting : arranging two categories, mostly similar, one containing z and the other z', the latter dominating nothing. There will need to be a FCR

$$[z] \equiv [+INV]$$

as well as a condition stating that if some category has [z, +INV, αC] then another will have [z', αC], where α and C range as before. In that case, sentences like "can Kim go" will appear in leaf form thus :



provided we arrange for a statement

$$[z] < [\bar{N}]$$

This is not tremendously beautiful, however, and I think a better solution may emerge with a more thorough investigation of the "precedes" relation. This is connected to all the above problems, as well as such things as "sentential adverb insertion", which ARP also tackles by metarule.

Conclusion

As with the frog climbing out of the well, one makes three steps upwards but slides two back. I think I have shown that treelessness has its advantages, however. As well as its potential to go beyond context freeness, it can simplify at least some kinds of rules. A more extensive treatment of precedence would almost certainly add to the attractiveness of the method, and I think I can achieve this in the near future.

A rather more nagging problem is that of the semantic component. I had hoped that this could be included just by conjoining two sentences, one in the natural language under investigation, English in this case, and the other in a formal language, intensional logic, let us say. The process of bracketing would link the two. Unfortunately this simple picture cannot work, as Proof 1 shows. For instance, producing the conjunction

John loves Jane ~ love' (john', jane')

means correlating "John" and "john'", "Jane" and "jane'" in the right order, a process akin to duplication, which this kind of grammar cannot handle.

On the other hand, the "milder than mild" nature of treelessness may have valuable consequences, especially in the drive towards "minimalist" theories. Just as certain detergents can be powerful on dirt, yet kind to the hands ... but many more investigations need to be carried out.

The Proofs

Proof 1

It is straightforward to show that multiple agreements are included in the treeless family. Consider the following finite state grammar :

$X_0 \rightarrow \lambda$

$X_0 \rightarrow xaX_0$

$X_0 \rightarrow xaX_1$

$X_1 \rightarrow x'ybX_1$

$X_1 \rightarrow x'ybX_2$

$X_2 \rightarrow y'cX_2$

$X_2 \rightarrow y'c$

The language generated by this consists of any number of xa's followed by any number of x'yb's followed by any number of y'c's. "Any number" includes none of anything, by virtue of the first rule.

There follow these two bracketing grammars :

$Y_1 \rightarrow \lambda$	$Y_2 \rightarrow \lambda$
$Y_1 \rightarrow Y_1Y_1$	$Y_2 \rightarrow Y_2Y_2$
$Y_1 \rightarrow xY_1x'$	$Y_2 \rightarrow yY_2y'$
$Y_1 \rightarrow a$	$Y_2 \rightarrow a$

$Y_1 \rightarrow b$	$Y_2 \rightarrow b$
$Y_1 \rightarrow y$	$Y_2 \rightarrow x$
$Y_1 \rightarrow y'$	$Y_2 \rightarrow x'$

The first of these generates random strings of terminals subject only to the condition that the number of x's is the same as a subsequent number of x'' s. The second behaves similarly, but controls y' s and y'' s instead. The finite state grammar ensures that every x comes with just one a, that every b is accompanied by one x' and one y, while the number of c' s which follow is the same as the number of y'' s. When the bracs and kets (defined in Stirk 1999) are eliminated, only sentences of the form $a^ib^ic^i$ remain.

Crossed agreements can be dealt with similarly. The following finite state and bracketing grammars will do :

$X_0 \rightarrow \lambda$

$X_0 \rightarrow xaX_0$

$X_0 \rightarrow xaX_1$

$X_1 \rightarrow ybX_1$

$X_1 \rightarrow ybX_2$

$X_2 \rightarrow x'cX_2$

$X_2 \rightarrow x'cX_3$

$X_3 \rightarrow y'dX_3$

$X_3 \rightarrow y'd$

$Y_1 \rightarrow \lambda$	$Y_2 \rightarrow \lambda$
$Y_1 \rightarrow Y_1 Y_1$	$Y_2 \rightarrow Y_2Y_2$

$Y_1 \rightarrow xY_1x'$	$Y_2 \rightarrow yY_2y'$
$Y_1 \rightarrow a$	$Y_2 \rightarrow a$
$Y_1 \rightarrow b$	$Y_2 \rightarrow b$
$Y_1 \rightarrow c$	$Y_2 \rightarrow c$
$Y_1 \rightarrow d$	$Y_2 \rightarrow d$
$Y_1 \rightarrow y$	$Y_2 \rightarrow x$
$Y_1 \rightarrow y'$	$Y_2 \rightarrow x'$

Here again the bracketing grammars just take care of the numbers of x, x' and y, y' (and, redundantly, the ordering of each pair). The finite state grammar fixes everything else, including, crucially, the order of the various elements.

This is essentially why the treeless family cannot include duplication. In this, a random string of a's and b's has to be followed by a copy of the same string. A finite state grammar can manage a finite number of orderings, as the one above does for crossed agreements, but it could not copy the infinite variety of a random string. The bracketing grammars can keep *numbers* of items under control, but cannot cope with the ordering of several kinds of bracs and kets. Notice that even if the powers of a bracketing grammar were increased to include more than one kind of bracketing, only *mirror images* of random strings could be generated, as in the following example :

$X_0 \rightarrow xaX_0$, $X_0 \rightarrow ybX_0$, $X_0 \rightarrow X_1$, $X_1 \rightarrow x'aX_1$, $X_1 \rightarrow y'bX_1$, $X_1 \rightarrow \lambda$

$Y \rightarrow YY$, $Y \rightarrow xYx'$, $Y \rightarrow yYy'$, $Y \rightarrow a$, $Y \rightarrow b$

Proof 2

A simple example will serve to illustrate the method of proof. Suppose there are just 3 features, A, B, and C, each of which can take just two values. That means a total of eight distinct categories, which we could imagine as standing for eight non-terminal symbols of a finite state grammar, X_1, X_2, ... , X_8. X_1 could be thought of as representing [-A, -B, -C], X_2 representing [-A , -B, +C], and so on, each X_i being matched with the binary number for i-1.

Imagine a grammar which contained all of the following rules :

$X_1 \rightarrow aX_3$

$X_1 \rightarrow aX_4$

$X_1 \rightarrow aX_7$

$X_1 \rightarrow aX_8$

$X_2 \rightarrow bX_3$

$X_2 \rightarrow bX_4$

$X_2 \rightarrow bX_7$

$X_2 \rightarrow bX_8$

$X_3 \rightarrow cX_3$

$X_3 \rightarrow cX_4$

$X_3 \rightarrow cX_7$

$X_3 \rightarrow cX_8$

$X_4 \rightarrow dX_3$

$X_4 \rightarrow dX_4$

$X_4 \rightarrow dX_7$

$X_4 \rightarrow dX_8$

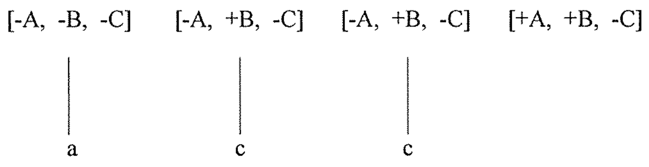
Comparing these to the categories, we see that in each rule the non-terminal symbol on the left hand side has the feature "A" with value "-", while the one on the right has the feature "B" with value "+". Using the symbol "<" for "can immediately precede" we can sum up the non-terminal part of the sixteen rules by the simple statement

[-A] < [+B]

Using the symbol "|" for "immediately dominates (a terminal symbol)", we add these dominance statements

[-A, -B, -C]	| a
[-A, -B, +C]	| b
[-A, +B, -C]	| c
[-A, +B, +C]	| d

These statements can clearly be used to sanction such leaf diagrams as



This corresponds to the following (partial) derivation of the finite state grammar :

X_1

aX_3

acX_3

$accX_7$

This simple model illustrates the principles which can be used to turn finite state grammars into statements suitable for translation into leaf diagram form. Clearly there would be no extra difficulty in doing the trick the other way round : extracting a finite state grammar from some immediate precedence and dominance statements.

Proof 3

It is generally convenient to organise the intersections of finite state and bracketing languages by using abstract markers like y, y' which disappear from the final sentence. But as always there is more than one way to skin a cat. Suppose for illustration that we have a rule

$X_9 \rightarrow ayX_{10}$

and another

$X_5 \rightarrow by'X_3$

The y and y' ensure that if the first of these rules is used in a derivation, the second will be also, because of the bracketing component. Another way of ensuring the same result would be to add a rule $X_i \rightarrow aX_{10}$ to the grammar, where X_i is a new non-terminal symbol. Similarly another rule, $X_j \rightarrow bX_3$, is added, where X_j is new. Rules in the grammar with X_9 on the right hand side are duplicated also, but this time with X_i instead of X_9. The same is done with X_5 and X_j.

Now, instead of using the y, y' symbols in the bracketing language, we use the non-terminals X_i and X_j of the finite state grammar, making the intersection apply to derivations rather than terminal strings. Thus if X_i appears somewhere in a derivation, X_j must appear later. The effect will be to generate the same language as before.

Since categories take the place of the regular non-terminal symbols, and extra features add to the number of categories, this is precisely the method used in the discussion above.

Bibliography

ARP : Gazdar, Pullum and Sag (1982)

G. Gazdar, E. Klein, G. K. Pullum and I. A. Sag (1985)
"Generalized Phrase Structure Grammar" (Basil Blackwell)

G. Gazdar, G. K. Pullum and I. A. Sag (1982)
"Auxiliaries and Related Phenomena in a Restricted Theory of Grammar"
(Language Vol 58 No 3)

C. Martin-Vide, A. Mateescu and A. Salomaa (1999)
"Sewing Contexts and Mildly Context-Sensitive Languages"
(Turku Centre for Computer Science TUCS Technical Report No 257)

C. Pollard and I. A. Sag (1994)
"Head-Driven Phrase Structure Grammar"
(University of Chicago Press)

Ian C. Stirk (1999) "Cut Down the Trees, and Save the Environment"
(大阪外大英米研究第23号)