More on analogical grammars

Ian C. Stirk

Introduction

In a previous paper in this Journal (Stirk 2008), I set out an outline proof that analogical grammars are mildly context-sensitive. As part of that proof, I suggested (p 74) a way of generating any analogical language using a CS grammar. I have since found a better way of doing this, and since it throws some light on analogical languages in general, and thus on human languages, I think it is a good idea to present the method in detail.

Analogical languages and grammars

It is best to introduce the general principles with an example. Consider the CS language $a^n b^n c^n$, where $n \geq 1$, and the following finite state (FS) grammar:

(1) $X_0 \rightarrow \{_0 \{_1 a X_0$

$X_0 \rightarrow \{_0 \{_1 a X_1$

$X_1 \rightarrow \}_0 b X_1$

$X_1 \rightarrow \}_0 b X_2$

$X_2 \rightarrow \}_1 c X_2$

$X_2 \rightarrow \}_1 c$

That grammar will generate any number of a's, each preceded by $\{_0 \{_1$, followed by any number of b's preceded by $\}_0$, and ending up with any number of c's preceded by $\}_1$, for example

$\{_0 \{_1 a \{_0 \{_1 a \}_0 b \}_0 b \}_1 c \}_1 c$

Clearly if the number of a's and b's and c's is to be the same, the brackets $\{_0$ $\}_0$ and $\{_1 \}_1$ have to be properly paired, and the numbers of each pair have to be the same. There is a slight complication in that one member of a pair may sometimes be enclosed in another pair: in the above example, for instance, a $\{_1$ appears between a $\{_0$ and a $\}_0$. That complication might be avoided in simple cases like the present example, but we should consider the general case. In order to get the pairings and the numbers of pairs correct, we arrange that the final language should be the intersection of the FS language and the two context-free (CF) languages generated by the following grammars:

(2) $Y_0 \rightarrow Y_0 Y_0$
$Y_0 \rightarrow Y \{_0 Y Y_0 Y \}_0 Y$
$Y_0 \rightarrow Y \{_0 Y \}_0 Y$
$Y \rightarrow a, b, c, \{_1, \}_1$

(3) $Z_0 \rightarrow Z_0 Z_0$
$Z_0 \rightarrow Z \{_0 Z Z_0 Z \}_0 Z$
$Z_0 \rightarrow Z \{_0 Z \}_0 Z$
$Z \rightarrow a, b, c, \{_0, \}_0$

Grammar (2) generates proper pairs of $\{_0\ \}_0$ interspersed with random strings of other symbols. The last rule means that Y may be rewritten as a or b or c and so on. Grammar (3) does the same work for the pairs $\{_1\ \}_1$. Clearly the language of the intersection will be $a^n b^n c^n$ together with brackets. Thus in general analogical grammars consist of a FS grammar and a number of CF grammars like (2) and (3), one for each pair of brackets involved. The analogical language is the intersection of the languages generated by these grammars, with the brackets removed.

This deletion of brackets is the main problem when we come to proving that analogical languages are mildly context-sensitive. Deletion is always a problem for such proofs, and it is even worse when we do not know exactly what is being deleted in every case. There may be any number of bracket pairs.

The simplest way round the problem is to generate analogical languages using a CS grammar right from the start.

Context-sensitive grammars for analogical languages

The CS grammar will be based on the FS grammar which is part of the analogical one. In what follows I will concentrate on the simple analogical grammar described above, and indicate how it may be extended to the general case. The first set of rules of the CS grammar is

(4) $X_0 \rightarrow P_0\, R_0\, X_1\, Q$

$X_1 \rightarrow A_1\, X_1$

$X_1 \rightarrow A_1\, X_2$

$X_2 \rightarrow B\, X_2$

$X_2 \rightarrow B\, X_3$

$X_3 \rightarrow C\ X_3$

$X_3 \rightarrow C$

Compared to the original FS rules, it will be seen that these produce similar strings, but where all the terminal symbols have been replaced by non-terminals, a by A1 and so on. Also strings generated by these rules will be preceded by P_0 R_0 and will end in Q. The reason for the subscript 1 on A_1 is that in the original grammar a was accompanied by two brackets, $\{_0$ and $\{_1$. The terminals b and c were only accompanied by one each, so B and C have no subscript. (I suppose strictly speaking they should have a subscripted 0, but no subscript at all saves typing!) Anyway, we can more easily see the action of the next sets of rules by using a sample derivation from those in (4), perhaps

$P_0\ R_0\ A_1\ A_1\ A_1\ B\ B\ B\ C\ C\ C\ Q$

The next of rules will be

(5) $R_0\ \alpha \rightarrow \alpha\ R_0$ (That is actually a rule schema: α is to range over all symbols except A1 and Q)

$R_0\ A_1 \rightarrow A\ S_0$

$S_0\ \beta \rightarrow \beta\ S_0$ (β ranges over all symbols except B and Q)

$S_0\ B \rightarrow b\ T_0$

$T_0\ \gamma \rightarrow \gamma\ T_0$ (γ ranges over all symbols except Q and P with any subscript)

$T_0\ Q \rightarrow U_0\ Q$

The effect of those rules on our sample string will be that the first A_1 becomes A, the first B becomes b, and a U_0 will end up just to the left of Q:

$P_0\,R_0\,A\,A_1\,A_1\,b\,B\,B\,C\,C\,C\,U_0\,Q$

Notice that the effect has been, in a way, to check a $\{_0\ \}_0$ bracketing that would have appeared in the analogical grammar. Of course we are using a sample string that will eventually lead to a correct derivation, but that is not guaranteed by the FS rules of (4). If there had been no B's in our sample string, then S_0 would have moved right as far as Q, after which no further rule of the grammar would have applied, and the derivation would have become stuck without ever leading to a terminal string.

Anyway, the next step is to move U_0 to the left, and start the process again:

(6) $\gamma\ U_0 \rightarrow U_0\ \gamma$

$P_0\,U_0 \rightarrow P_0\,R_0$

The rules in (6) and (5) will bring our sample string to

$P_0\,A\,A\,A\,b\,b\,b\,C\,C\,C\,R_0\,Q$

R_0 will have moved right as far as Q, since at the end there will be no A_1's to stop it. Notice here that if there had been more B's than A_1's they would remain as B's, that is, as non-terminal symbols, even though R_0 still reached Q. They will remain non-terminals too, as no further rules will apply to them, so no derivation containing them will reach a terminal string, whatever the effect of other rules. That ensures proper pairings like the analogical grammar does: if there are too many left-hand brackets, as it were, then a non-terminal symbol like S_0 will reach Q and block the derivation, if there are too many right-hand brackets, at least one non-terminal like B will be left stranded.

We deal with the R_0 just to the left of Q as follows:

(7) $R_0\,Q \rightarrow V_0\,Q$

$\gamma\ V_0 \rightarrow V_0\ \gamma$

$P_0\,V_0 \rightarrow P_1\,R_1$

Now the idea is that R_1 will be the trigger for checking the second lot of brackets that were present in the analogical grammar. The rules involving it and other symbols will of course be similar to those in (5) and (6):

(8) $R_1\ \alpha \rightarrow \alpha\ R_1$ (That is actually a rule schema: α is to range over all symbols except A and Q)

$R_1\,A \rightarrow a\,S_1$

$S_1\ \beta \rightarrow \beta\ S_1$ (β ranges over all symbols except C and Q)

$S_1\,C \rightarrow c\,T_1$

$T_1\ \gamma \rightarrow \gamma\ T_1$ (γ ranges over all symbols except Q and P with any subscript)

$T_1\,Q \rightarrow U_1\,Q$

$\gamma\ U_1 \rightarrow U_1\ \gamma$

$P1\ U_1 \rightarrow P_1\,R_1$

Clearly those rules will operate on our sample string to end up with

$P_1\,a\,a\,a\,b\,b\,b\,c\,c\,c\,R_1\,Q$

Again derivations without the proper number of C's would fail – too few C's would cause $S_1\,Q$ to appear, too many would leave some over. We can change R_1 to V_1 and send it back left as we did to R_0:

(9) $R_1\ Q \rightarrow V_1\ Q$

$\gamma\ V_1 \rightarrow V_1\ \gamma$

but in this particular case, there are no more bracket pairs left to go. The derivation could be brought to an end with these rules:

(10) $R_1\ V_1\ \gamma \rightarrow \gamma\ R_1\ V_1$

$R_1\ V_1\ Q \rightarrow ttt$

where t is a new terminal symbol, necessary because CS grammars cannot delete any symbols. The t's do not cause any trouble: they can be ignored when considering sentences of the analogical language, and added when we want to check whether or not a particular string can be generated by the CS grammar.

It should be pretty clear that the example above can be generalised to suit other analogical grammars. Non-terminal symbols are chosen for each terminal, the number of brackets associated with each is registered as a subscript, and so on. But there is one potential problem which I will explore below.

A potential difficulty

In the example above, one terminal symbol of the analogical grammar had two left-hand brackets associated with it, while the others just had one right-hand bracket each. It is important to be sure that the method of constructing a CS grammar will still work if terminals are associated with a mixture of left- and right-handed brackets. The following analogical FS grammar will put this to the test:

(11) $X_0 \rightarrow \{_0\ a\ X_0$

$X_0 \rightarrow \{_0\ a\ X_1$

$X_1 \rightarrow \{_1\ \}_0\ b\ X_1$

$X_1 \rightarrow \{_1\ \}_0\ b\ X_2$

$X_2 \rightarrow \}_1\ c\ X_2$

$X_2 \rightarrow \}_1\ c$

The language generated will be the same as before, and the same pair of CF grammars will provide the other languages for intersection. Notice that I still have not neatly nested one bracket pair inside the other. Associated with b, though, is a left-hand and a right-hand bracket. According to the procedure developed before, the first set of rules of the CS grammar should be

(12) $X_0 \rightarrow P_0\ R_0\ X_1\ Q$

$X_1 \rightarrow A\ X_1$

$X_1 \rightarrow A\ X_2$

$X_2 \rightarrow B_1\ X_2$

$X_2 \rightarrow B_1\ X_3$

$X_3 \rightarrow C\ X_3$

$X_3 \rightarrow C$

A sample string might be

$P_0\ R_0\ A\ A\ A\ B_1\ B_1\ B_1\ C\ C\ C\ Q$

The other rules will be adapted as follows:

(13) $R_0\ \alpha \rightarrow \alpha\ R_0$ (That is actually a rule schema: α is to range

over all symbols except A1 and Q)

$R_0\,A \rightarrow a\,S_0$

$S_0\,\beta \rightarrow \beta\,S_0$ (β ranges over all symbols except B and Q)

$S_0\,B_1 \rightarrow B\,T_0$

$T_0\,\gamma \rightarrow \gamma\,T_0$ (γ ranges over all symbols except Q and P with any subscript)

$T_0\,Q \rightarrow U_0\,Q$

$\gamma\,U_0 \rightarrow U_0\,\gamma$

$P_0\,U_0 \rightarrow P_0\,R_0$

So far so good: the sample string will become

$P_0\,a\,a\,a\,B\,B\,B\,C\,C\,C\,R_0\,Q$

and any mismatch in numbers will block the derivations as before. The remaining rules can be easily adapted, it seems:

(14) $R_0\,Q \rightarrow V_0\,Q$

$\gamma\,V_0 \rightarrow V_0\,\gamma$

$P_0\,V_0 \rightarrow P_1\,R_1$

$R_1\,\alpha \rightarrow \alpha\,R_1$ (That is actually a rule schema: α is to range over all symbols except B and Q)

$R_1\,B \rightarrow b\,S_1$

$S_1\,\beta \rightarrow \beta\,S_1$ (β ranges over all symbols except C and Q)

$S_1\,C \rightarrow c\,T_1$

$T_1\,\gamma \rightarrow \gamma\,T_1$ (γ ranges over all symbols except Q and P with any subscript)

$T_1\,Q \rightarrow U_1\,Q$

$\gamma\ U_1 \rightarrow U_1\ \gamma$

$P_1\ U_1 \rightarrow P_1\ R_1$

Those will certainly turn our sample string into

$P_1\ a\ a\ a\ b\ b\ b\ c\ c\ c\ R_1\ Q$

and the rules in (9) and (10) will turn that into the terminal string we expect:

a a a b b b c c c t t t

The other kind of bracketing does not cause any difficulty, fortunately.

Conclusion

I am relieved to make it absolutely certain that analogical languages are definitely context-sensitive, and thus mildly context-sensitive, as demonstrated in my (2008).

The result is important because I am convinced that human languages are analogical, and that this fact explains how human infants are able to learn them, and also explains why human languages have the characteristics that they do have.

Reference

Ian C. Stirk "Analogical grammars are mildly context sensitive" (Journal of Anglo-American Studies, 英米研究, 2008)