Analogical grammars *are* mildly context sensitive

Ian C. Stirk

Introduction

In 1999 I published a paper in Eibeikenkyu called "Cut down the trees, and save the environment". I had hoped to include in it a proof that analogical grammars, or treeless grammars as I called them then, were mildly context sensitive. Unfortunately I could not work out a completely sound proof, even though Masaji Tajiri, who was editing that particular volume, kindly extended the deadline for me.

Now I have discovered a sound proof, and that is what I present here. I would like to dedicate it to the memory of Masaji Tajiri.

Analogical grammars

Consider the following examples of Basque sentences with English translations:

Mikel joango da	=	Mikel will go
Miren joango da	=	Miren will go
Itziar joango da	=	Itziar will go

Iñaki joango da	=	Iñaki will go
Margari joango da	=	Margari will go
Joseba joango da	=	Joseba will go
Maite joango da	=	Maite will go

We could try to generate the strings using a finite state (FS) grammar:

$X_0 \rightarrow$ Mikel X_1, $X_0 \rightarrow$ Miren X_1 etc, $X_1 \rightarrow$ joango da X_2, $X_2 \rightarrow =X_3$,
$X_3 \rightarrow$ Mikel X_4 etc, $X_4 \rightarrow$ will go

That grammar will certainly generate the example strings, but unfortunately it will also generate a whole lot of strings like the following:

Itziar joango da	=	Maite will go

since there is no mechanism for matching up the names in the Basque part and those in the English part. To avoid the problem, the FS grammar is amended as follows:

$X_0 \rightarrow$ Mikel $\{_0$ X_1, $X_0 \rightarrow$ Miren $\{_1$ X_1 etc, $X_1 \rightarrow$ joango da X_2, $X_2 \rightarrow =X_3$,
$X_3 \rightarrow$ Mikel $\}_0$ X_4 etc, $X_4 \rightarrow$ will go

Now each name in the Basque part is generated with a numbered left hand curly bracket, and the same name in the English part comes with a similarly numbered right hand bracket. We stipulate that every left hand bracket be paired with a right hand one bearing the same number. In that case, strings like

Itziar $\{_2$ joango da	=	Maite $\}_6$ will go

will not be accepted. In the end, the brackets are removed to give the final string of the analogical grammar.

Brackets can do far more than just pair off items in the Basque part and the English part of a string. If we add another sentence pair:

Mikelek Miren ikusiko du = Mikel will see Miren

then it looks as if the FS grammar could be adjusted as follows:

$X_0 \rightarrow$ Mikel $\{_0$ X_1, $X_0 \rightarrow$ Miren $\{_1$ X_1 etc, $X_1 \rightarrow$ ek $\{_7$ X_1, $X_1 \rightarrow$ joango da $\{_9$ X_2,
$X_1 \rightarrow$ Mikel $\{_0 \}_7$ $\{_8$ $\{_{11}$ X_2 etc, $X_2 \rightarrow$ ikusiko du $\{_{10} \}_8$ X_2, $X_2 \rightarrow = X_3$,
$X_3 \rightarrow$ Mikel $\}_0$ X_4 etc, $X_4 \rightarrow$ will go $\}_9$, $X_4 \rightarrow$ will see $\}_{10}$ X_4,
$X_4 \rightarrow$ Mikel $\}_0 \}_{11}$ etc.

Although some of the rules seem to be recursive, the brackets prevent them from being used so. Of course a whole lot of new sentence pairs will be generated by this, as well as the one pair that was added to prompt the new grammar. That is why such grammars are called "analogical".

Notice also that no special mention has been made of Basque's famous "ergative" case ending "-ek". The pairing of Basque and English takes care of the grammatical differences between the two languages without any extra fuss.

However it should be noted that the previous version of the grammar will generate some ungrammatical Basque. "Itziar" has "Itziarrek" in the ergative case rather than "Itziarek", while the names ending in a vowel add only "-k": "Iñakik" etc. The following amended grammar will take care of that problem:

$X_0 \rightarrow$ Mikel $\{_0\ X_1$, $X_0 \rightarrow$ Miren $\{_1\ X_1$ etc,

$X_0 \rightarrow$ Mikel $\{_0\ \{_{12}\ X_1$, $X_0 \rightarrow$ Miren $\{_1\ \{_{12}\ X_1$,

$X_0 \rightarrow$ Itziar $\{_2\ \{_{13}\ X_1$,

$X_0 \rightarrow$ Iñaki $\{_3\ \{_{14}\ X_1$ etc,

$X_1 \rightarrow$ ek $\{_7\ \}_{12}\ X_1$, $X_1 \rightarrow$ rek $\{_7\ \}_{13}\ X_1$, $X_1 \rightarrow$ k $\{_7\ \}_{14}\ X_1$,

$X_1 \rightarrow$ joango da $\{_9\ X_2$,

$X_1 \rightarrow$ Mikel $\{_0\ \}_7\ \{_8\ \{_{11}\ X_2$ etc, $X_2 \rightarrow$ ikusiko du $\{_{10}\ \}_8\ X_2$, $X_2 \rightarrow =X_3$,

$X_3 \rightarrow$ Mikel $\}_0\ X_4$ etc, $X_4 \rightarrow$ will go $\}_9$, $X_4 \rightarrow$ will see $\}_{10}\ X_4$,

$X_4 \rightarrow$ Mikel $\}_0\ \}_{11}$ etc.

In this grammar, the first line of rules covers every name. The second line involves just "Mikel" and "Miren", adding an extra bracket. The third line has a rule just for "Itziar", while the fourth has rules for "Iñaki", "Margari", "Joseba" and "Maite". The new brackets will link the names to the correct ending among the rules of the fifth line.

No doubt there are better ways of organising this kind of grammar, but I am only interested just now in illustrating the principle, and demonstrating the power of the bracketing system.

Analogical grammars are context-sensitive

The following CS grammar will generate strings of the language $a^n b^n c^n$, together with brackets.

1. $X_0 \rightarrow$ a $L_0\ X_0$
2. $X_0 \rightarrow$ b $R_0\ L_1\ X_1$
3. $X_1 \rightarrow$ b $R_0\ L_1\ X_1$

4. $X_1 \rightarrow c\ R_1\ X_2$
5. $X_2 \rightarrow c\ R_1\ X_2$
6. $X_2 \rightarrow c\ R_1$

7. $L_0\ a \rightarrow a\ L_0$
8. $L_0\ b \rightarrow b\ L_0$
9. $L_0\ L_1 \rightarrow L_1\ L_0$
10. $L_0\ \{_0\ \}_0 \rightarrow \{_0\ \}_0\ L_0$

11. $L_1\ b \rightarrow b\ L_1$
12. $L_1\ c \rightarrow c\ L_1$
13. $L_1\ \{_0\ \}_0 \rightarrow \{_0\ \}_0\ L_1$
14. $L_1\ \{_1\ \}_1 \rightarrow \{_1\ \}_1\ L_1$

15. $L_0\ R_0 \rightarrow \{_0\ \}_0$
16. $L_1\ R_1 \rightarrow \{_1\ \}_1$

The first six rules act like a finite state (FS) grammar, generating strings of a's, b's and c's together with the non-terminal symbols L_0, R_0, L_1 and R_1. Here is an example derivation:

$$X_0$$

.

.

$$a\ L_0\ a\ L_0\ a\ L_0\ b\ R_0\ L_1\ b\ R_0\ L_1\ b\ R_0\ L_1\ c\ R_1\ c\ R_1\ c\ R_1$$

The dots indicate various applications of rules 1 through 6. The next lot of rules, 7 through 10, move occurrences of L_0 to the right. Rule 15 determines

the fate of L_0 when it meets an R_0. Rules may of course be applied in any order, when the conditions of their left hand sides are met, but one possibility is to use just the rules 7-10 and 15 until we get

$$a\ a\ a\ b\ \{_0\ \}_0\ L_1\ b\ \{_0\ \}_0\ L_1\ b\ \{_0\ \}_0\ L_1\ c\ R_1\ c\ R_1\ c\ R_1$$

Then we could use 11-14 and 16 until we reach a final string T:

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

This is a final string, as the brackets are terminal symbols. It contains equal numbers of a's, b's and c's, as it should. Clearly if we had used rules 1-6 to generate different numbers of a's, b's and c's, then some L's or R's would be left over, unable to become left or right brackets. Such derivations do not result in terminal strings. Thus if we can just suppress the brackets, the language generated by this CS grammar is just $a^nb^nc^n$.

There might be a complaint that the final strings of this CS grammar are not precisely like those of an analogical grammar: the brackets all cluster in pairs, instead of appearing as in T^* below:

$$T^* \quad a\ \{_0 a\ \{_0 a\ \{_0\ b\ \}_0\ \{_1\ b\ \}_0\ \{_1\ b\ \}_0\ \{_1\ c\ \}_1\ c\ \}_1\ c\ \}_1$$

as they should. No problem! If we just add the following rules to the CS grammar:

17. $b\ \{_0 \rightarrow \{_0\ b$
18. $\}_0\ \{_0 \rightarrow \{_0\ \}_0$
19. $a\ \{_0\ \{_0 \rightarrow \{_0\ a\ \{_0$

20. c $\{_1$ → $\{_1$ c

21. $\}_1$ $\{_1$ → $\{_1$ $\}_1$

22. b $\}_0$ $\{_1$ $\{_1$ → $\{_1$ b $\}_0$ $\{_1$

the reader may easily check that they will shuffle T to T^*, and in general bring all final strings into analogical form. This kind of shuffling is a feature of CS grammars.

Suppressing the brackets to obtain a final string is not done in CS grammars however. There is always the danger that deletion may shift the language generated from being recursive to being merely computably enumerable (I use this term instead of "recursively enumerable", following Chaitin, 2006, p35). The difference is important. If a language is recursive, it is always possible to check whether or not a sentence is grammatical according to the grammar. It is only necessary to check the finite number of derivations that lead to sentences of that length. If the sentence is among those, it is grammatical, otherwise it is not. If the rules of the grammar involve deletions, however, it is impossible to be sure. Lines in the derivation may get longer than the sentence we are checking, but maybe deletions will bring them down to the required length in the end. Or maybe not. We can generate sentence after sentence using the grammar, but we can never be sure that a particular sentence will *not* be generated at some time. This is the idea of computable enumeration.

Now analogical grammars involve deletion, but they still generate recursive languages. This is because any terminal symbol can only be associated with a finite number of brackets, and there are only a finite number of different brackets. So if there is, say, a maximum of five brackets connected with any terminal symbol, and if there are n symbols in a particular string of terminals,

we only need to check outputs of the grammar up to at most 5n symbols long. If the string, plus brackets, is not among these, it is not grammatical.

Unfortunately there are still recursive languages which are not CS. A proof of this can be found in Salomaa (1973) and Stirk (1988). This prevented me, in Stirk (1999), from being able to state with certainty that analogical grammars (or treeless grammars, as I called them then) were mildly CS.

But now I realise that any analogical grammar can be replaced by a CS grammar that generates the same language but without any brackets. A clue to the reason why is contained in the example grammar above. The language $a^n b^n c^n$ can be generated by a very simple CS grammar such as the following:

1. $X_0 \rightarrow a\ X_0\ b\ c$
2. $X_0 \rightarrow a\ b\ c$
3. $c\ b \rightarrow b\ c$

It is easy to see that rules 1 and 2 generate strings with equal numbers of a's, b's and c's, but with the b's and c's mixed up. The third rule shuffles the b's and c's apart until everything is in the right order. Compare that to the CS grammar above, which generated $a^n b^n c^n$ in the same way as an anological grammar, with brackets to keep the numbers of symbols right. The simple CS grammar gets the numbers right by generating the related items in the same place. After that a rule moves those terminal symbols into the right places. Clearly the trick could always be done in this way. Instead of pairing distant terminal symbols by brackets, we generate them in the same place without brackets, and then move one of them by CS rules until it reaches its proper place. No doubt a formal proof could be given, but it would be tedious and I will not attempt it here.

Anyway, we can be satisfied that analogical languages are definitely CS.

The context-sensitivity is only mild

We can show that they are only mildly CS by finding CS languages which they cannot generate. I have already given more or less the same proof as the one that follows in Stirk (1999).

Consider the CS grammar above which imitated an analogical one. It began with some FS rules, which contained all the recursion necessary to generate an infinite number of sentences. In general, such a FS component may generate some number K of terminal symbols before any recursion happens. When a recursive rule or set of rules is used, the number of brackets generated may need to be balanced by brackets from other recursive rules, if a grammatical sentence is to result. Because of recursion, the same set of rules may apply again, once, twice, or any number of times. If one lot of brackets can be balanced by other rules, then so can two, three or any number. In general, we can see that for some number L, if K+L symbols make up a grammatical sentence (after the brackets have been removed), then so will K+2L, K+3L and so on. In the example grammar above, K=0 and L=3. The string abc is grammatical, with 3 symbols, so is aabbcc with 6 and so on.

Now consider this CS grammar:

1. $X_0 \rightarrow X_1\ b$
2. $X_1 \rightarrow X_1\ X_2$
3. $X_1 \rightarrow b\ X_3$
4. $X_3\ X_2 \rightarrow X_2\ X_3\ X_3$

5. X_3 b → b b
6. b X_2 → b b

The first three rules of it will generate strings like the following:

b X_3 X_2 . . . . . X_2 b

Each such string will contain n X_2's, where $n \geq 0$ (0 if rule 2 is not used). It is rule 4 which causes the fun. Each time it applies, one X_3 jumps over an X_2 and becomes two. If there are n X_2's, then after rule 4 has finished applying, there will be 2^n X_3's following them. Rules 5 and 6 change all the remaining non-terminal symbols into b's, so in the end there will 2 b's coming from rules 1 and 3, n b's from the n X_2's and 2^n from the 2^n X_3's, 2^n+n+2 b's altogether.

The shortest grammatical sentence will thus be bbb ($n=0$), followed by bbbbb ($n=1$). Following that will be one with 8 b's, one with 13 b's, and so on, the number increasing rapidly.

Now suppose that there is an analogical grammar which can generate this language. Of course this grammar (or its CS equivalent) may generate any finite number of sentences directly, without any recursion. But at some stage recursion must set in. Suppose that this occurs at some value n. So for some K and L and n and x,

$$K+L = 2^n+n+2$$
$$K+2L = 2^{n+x}+n+x+2$$

Subtracting shows that

$$L = 2^{n+x} - 2^n + x$$

Another subtraction gives

$$K = 2^{n+1} - 2^{n+x} + n - x + 2 = 2^{n+1}(1 - 2^{x-1}) + n - x + 2$$

Clearly if $x=0$, then $L=0$ also, and there would be no recursion. Also, if $x>1$, then K will clearly be a negative quantity, which is impossible. The only possibility is that
$x=1$, when $L=2^n+1$ and $K=n+1$. That gives

$K+L=2^n+n+2$, and $K+2L=2^{n+1}+n+3$

which are fine. However, $K+3L$ should also be grammatical, and that is

$3 \times 2^n+n+4$

Unfortunately that is too small: the next largest sentence, according to the original CS grammar, should have $2^{n+2}+n+4$ b's in it, and that can be written

$4 \times 2^n+n+4$

In general, the length of sentences given by the analogical grammar increases linearly, while the length of those from the CS grammar increases exponentially.

So there are CS languages which cannot be analogical. We have proved that analogical grammars are mildly context sensitive.

That is all to the good: there is a certain amount of evidence that human languages are not context free, but on the other hand, their grammars should not be much more powerful. See Stirk (1999) for more discussion of this.

Bibliography

Chaitin, Gregory (2006) Meta Math! (Vintage Books)

Salomaa, Arto (1973) Formal Languages (Academic Press)

Stirk, Ian C. (1988) Counting Languages (大阪外国語大学英米研究 15)

Stirk, Ian C. (1999) Cut Down the Trees, and Save the Environment (大阪外国語大学英米研究 23)