RESTRICTIONS ON TRANSFORMATIONAL GRAMMARS

Ian C. Stirk

Introduction

In a previous contribution (Stirk, 1990), I presented a scheme for a transformational grammar that would generate the same language as an arbitrary unrestricted rewriting (UR) grammar. I present another one below, based on a different approach. The new one has the mathematical virtue of greater simplicity, and is much more conservative in the forms of transformation employed. More importantly, the only recursive symbol used in the base rules of this scheme is the initial symbol S itself. In the main part of the paper, I discuss certain restrictions on transformational grammars, whose properties are analysed by Peters and Ritchie (1973) and by Peters (1973). The proofs I present all assume that S shall be the only recursive symbol in the phrase structure component.

Transformational Grammars for UR Languages

The base rules of these grammars take the following form:

(1) $S \rightarrow \#^{i}S$ where $1 \leqq i \leqq n+3$

(2) $S \rightarrow a_{j}S$ where $1 \leqq j \leqq k$

(3) $S \rightarrow O\#O\#O\#O\#N$

(4) $N \rightarrow \#$

(1) and (2) are not individual rules, but schemata. The rules in (1) generate strings of i #'s followed by S. The maximum value of i is n+3, where n is the number of rules in a UR grammar generating the UR language. The a_j in (2) represent the k terminal symbols of the UR language.

The linearly ordered obligatory rules of the transformational component are now presented, in their order of application, interspersed with notes explaining their function.

T (1)

	#	−	[$_S$ X_1	−	0	−	#	−	0 − X_2 − 0 − N − X_3] $_S$
SD :	1		2	3	4	5	6	7	8 9
SC :	1		2	3	4	5	6	7	8 + 4 9

T (2)

	# #	−	[$_S$ X_1	−	0	#	−	0 − X_2 − 0 − N − X_3] $_S$
SD :	1		2	3	4	5	6	7 8
SC :	1		2	3	4	5	6	7 + 4 8

T (i + 2) where $1 \leqq i \leqq k$

	a_i	−	[$_S$ X_1	−	0	#	0 − X_2 − 0 − N − X_3] $_S$	
SD :	1		2	3	4	5	6	7
SC :	1		2	3	4	5	6 + 1	7

The third item here is of course a rule schema, covering k rules, one for each member of the terminal vocabulary of the UR language. The effect of these k+2 rules, in total, is to build up a random string of O's and #'s and terminal symbols just after the N in the innermost

subsentence. The string is random because of the arbitrary input from the PS component, but may be a line in some derivation of some sentence in the language we are interested in generating. The following rule, T(k+3), will end the formation of the random string.

T(k+3)

	### − [$_S$ X_1 − 0 − # − 0 − X_2 − 0							− N − X_3]$_S$	
SD :	1	2	3	4	5	6	7	8	9
SC :	1	2	3	4	5	6	7 + 4	8	9

This transformation will apply if the PS rules provide ### in the sentence being cycled over, and by adding a # between O and N, it blocks any further application of the preceding rules. In the next cycle, any of the transformations in the following schema may apply:

T(k+3+i) where $1 \leqq i \leqq n$

	$\#^{i+3}$ − [$_S$ X_1 − 0 # 0 − X_2 − P^i − X_3 − 0 − # −								N − X_4 − Q^i − X_5]$_S$			
SD :	1	2	3	4	5	6	7	8	9	10	11	12
SC :	1	2	3	4	5	6	7	8 + 8	9	10	11	12

Conditions: $X_2 = X_4$, $X_3 = X_5$.

There are n rules in this schema, and we have supposed that a grammar for the language in question has n rules, which can schematically be represented by the n expressions $P_1 \rightarrow Q_1$ to $P_n \rightarrow Q_n$. Suppose also that the UR grammar has r+1 non-terminal symbols, so that its non-terminal vocabulary may be expressed as the set $\{Y_0, Y_1, \ldots, Y_r\}$, with Y_0 as the initial symbol. The P^i and Q^i are strings

identical to P_i and Q_i respectively, except that each non-terminal symbol Y_j in V_N is replaced by # O^{j+1} #, that is, by j+1 O's between two boundary symbols. This encoding of non-terminal symbols is described in more detail in my (1990). If the subsentence being cycled over begins with a string of i+3 #'s, then the appropriate transformation in this schema will check for the presence of P¦ and Q¦ in two strings of symbols which must otherwise be identical, one before and one after the occurrence of O#N. If the two strings are present in the appropriate environment, O # N becomes O # # N. Otherwise there is no change, and it will be seen that no other transformation in the cycle can apply.

Recall that the random string of symbols built up to the right of O # N is a possible line in a derivation, containing the UR grammar's non-terminal symbols in encoded form. The string of symbols following O#O to the left of O#N is a possible previous line. The successful application of one of the transformations in the schema above ensures that the two strings have the necessary charcteristics of successive lines in a derivation: a rule of the UR grammar has in essence applied, replacing P¦ by Q¦ and leaving everything else unchanged. Initially, the PS rules provide the string #O# between O#O and ON in the lowest subsentence. Thus the first time that a transformation in the schema applies, X_2 and X_3 must be null, while P¦ must be #O#, the encoded form of the initial symbol Y_0 of the UR grammar. The effect is to commence a process which imitates correct derivations according to that UR grammar.

The next transformations are

T (k+n+4)

	X_1	− 0 −	#	0 − 0 −	X_2	− 0	# # N −	X_3
SD :	1	2	3	4	5	6		7
SC :	1	2	3	ϕ	5	6		7

T (k+n+5)

	X_1	− 0 −	#	− 0 −	#	− X_2	− 0	# # N −	X_3
SD :	1	2	3	4	5	6	7		8
SC :	1	2	3	4	ϕ	6	7		8

and a schema

T (k+n+5+i) where $1 \leqq i \leqq k$

	X_1 − a_i −	X_2 − 0	# 0 −	a_i −	X_3 − 0	# # N −	X_4	
SD :	1	2	3	4	5	6	7	8
SC :	1	2	3	4	ϕ	6	7	8

The presence of O # # N in the SD of each of these rules ensures that none of them can apply until the cycle after one of the T(k+3+i) has applied. The effect of these latest rules is to delete, under condition of identity, a symbol occurring between O # O and O # # N in the lowest subsentence. More than one of them can apply in a single cycle, and they will apply in successive cycles, until the entire string between O # O and O # # N has disappeared. Once that string has gone, the following rule will apply, which could not before:

T (2k+n+6)

	X_1	−	0	#	0	0	#	−	#	−	N	−	X_2
SD :	1			2			3		4		5		
SC :	1			2		3 + 3			4		5		

Now transformations in the following schema can apply:

T (2k+n+6+i) where $1 \leqq i \leqq k+2$

	X_1	−	0	#	0	−	X_2	−	0	#	#	#	N	−	ξ_i	−	X_3
SD :	1			2		3				4				5		6	
SC :	1			2		3				5 + 4				ϕ		6	

(Here $\xi_1 = \#$, $\xi_2 = 0$, $\xi_j = a_{j-2}$)

In successive cycles, these transformations will move the string of symbols from the right side of O # # # N to the left, to replace the previously deleted string. In effect, a line in a derivation which has been checked and found correct is deleted and replaced with the succeeding line.

The next step, of course, is to repeat the entire process, by generating another possible derivation line, checking if it is indeed correct, and then substituting it for its predecessor. The repetition can easily be commenced just by deleting the three #'s in the fragment O # # # N, for then the structural descriptions of the first rules in the cycle may again be met.

However, it is possible that the derivation has already come to an end: the string now between O # O and O # # # N may consist only of terminal symbols, or in other words, may be a sentence of the

language. The following transformation allows for this possibility:

T (3k+n+9)

	$X_1-0\#0-X_2-\#-X_3-0-\#-\#-\#-N$									
SD :	1	2	3	4	5	6	7	8	9	10
SC :	1	2	3	4	5	6	ϕ	ϕ	ϕ	10

The three #'s are only deleted if there is a boundary symbol to be found somewhere between O#O and O###N. This will only be the case if that part of the string contains the encoded form of a non-terminal symbol, so that the derivation has not terminated in a sentence. If the transformation does not apply, there is a sentence, and it will be necessary to remove all the unwanted material from the phrase marker:

T (3k+n+9+i) where $1 \leqq i \leqq k$

	$X_1-a_i-X_2-0\#0-X_3-a_i-X_4-0\#\#\#N$							
SD :	1	2	3	4	5	6	7	8
SC :	1	ϕ	3	4	5	6	7	8

The transformations in this schema delete unwanted terminal symbols, provided only that they are identical to symbols in the sentence being generated. More than one of these transformations can apply in a single cycle.

T (4k+n+10)

	$X_1-\#\#\#-X_2-0\#0-X_3-0-\#\#\#-N$							
SD :	1	2	3	4	5	6	7	8
SC :	1	ϕ	3	4	5	6	7	8

As well as terminal symbols, a whole collection of boundary symbols will need to be removed. The above transformation performs this task, removing them in blocks of three, on a condition of identity will the block of three between O and N. All the unwanted boundary symbols to the left may be removed in this way, for since the PS rules can provide an extra # or # #, the number of them may always be a multiple of three.

On the very last cycle, the SD of this final transformation may be met:

T (4k+n+11)

	#	− 0 −	#	− 0 −	X_1	− 0 −	#	− # −	#	− #
SD :	1	2	3	4	5	6	7	8	9	10
SC :	ϕ	ϕ	ϕ	ϕ	5	ϕ	ϕ	ϕ	ϕ	ϕ

which deletes everything except the final sentence. The first # in the SD comes from the topmost sentence, while the last one is the one dominated by N.

It is clear that nothing can be generated by this grammar free of boundary symbols except for sentences of the UR language. The only items not deleted under a condition of identity belong to the set {O, #}.

In their original proof that transformational grammars can generate any UR language, Peters and Ritchie (1971) employ a PS component rather different from the one used above. Every structure generated by it contains a string comprising all the terminal symbols

of the language. The ones which do not appear in a sentence generated by the grammar are all deleted. The designated set of terminal symbols which may be deleted thus contains the entire terminal vocabulary! This somehow violates the spirit, if not the letter, of Chomsky's restrictions on deletion. The PS component I have introduced above introduces some random selection of terminal symbols into its structures. If any of them are not incorporated into the sentence being generated, then no string free of boundary symbols is obtained.

Non-filtering and Local Filtering Languages

Filtering is used extensively in the grammar above, and it prompts one to speculate what kind of languages could be generated if there was no filtering whatsoever. The following theorem is rather easy to prove, at least informally:

Theorem 1 For any infinite non-filtering language L, there are constants M and x such that for any sentence of length $y \geqq x$, there is another sentence of length not greater than My.

Proof: Suppose that in a set of phrase structure rules, which form part of a non-filtering transformational grammar, any S on the right hand side of a rule is replaced by a new symbol S^{1}. In that case, there could be no recursion, and only a finite number of trees, or labelled bracketings, could be derived. Schematically, they might be represented as follows:

$[_s - - S' - S' - - -]_s$

Depending on the original rules, there may be any finite number of occurrences of S' in the bracket, but only two are depicted. Consider now any sentences generated by the transformational grammar, and their final phrase markers, represented as labelled bracketings. Their outermost brackets will of course be $[_s$ and $]_s$. Substitute them for the various occurrences of S' in the above schema. The result is a labelled bracketing of the kind to which the final cycle of a transformational derivation is applied. Suppose that one of the sentences we have just substituted for S' has length y, and that it is longer than any of the other substitutions, if there are any, and longer also than any string of terminals dominated by any non-terminal symbol in the whole labelled bracketing. There must be a minimum length x for y which makes this possible. How much longer than y could the sentence resulting after the final cycle be? Transformations in the cycle could cause lengthening by making multiple copies of substrings, of which the longest is y terminals long. However, since there are only a finite number of transformations in the cycle, of which each could make only a finite number of copies of anything, the result could not be longer than a certain multiple, call it M, of y. This establishes the result.

Peters and Ritchie (1973) realised that the same theorem 1 would also apply to languages generated by grammars with a certain restricted kind of filtering which they called "local filtering". A good example of this may be seen in the way relative clauses are handled in certain transformational grammars for English. A phrase like "the man

who came to dinner" might have the following schematic underlying structure:

$$[_{NP}[_{NP}\text{ the man }]_{NP}\ [_{S}[_{NP}\text{ the man }]_{NP}\text{ came to dinner }]_{S}\ ]_{NP}$$

The relative clause transformation would replace the second occurrence of "the man" by "who". A condition of identity is involved. If the NP in the embedded sentence is not identical to the NP on the left, the transformation will not apply, and boundary symbols in the sentence of which the entire NP is part will be left in place. Clearly nothing further could happen to rectify the situation: if the relative clause transformation does not apply to such a structure, then the result must be ungrammatical, and the boundary symbols cannot be removed.

This is the key to the concept of local filtering. All boundary symbols in any subsentence must be removed by the end of the transformational cycle applying to that subsentence. If any boundary symbols remain, there is no point in continuing the transformational derivation, for they will still remain at the end, and the resultant string is bound to be ungrammatical. Ungrammaticality can be detected more quickly with a grammar that employs only local filtering. Grammars of the kind presented in the last section are obviously not local filtering: the whole point of the last rules in the cycle is to remove boundary symbols left over from previous cycles.

Consider again the above proof concerning non-filtering grammars. How does it change if the grammar in question is a local filtering

one? All the subsentences substituted for the various instances of S^1 must of course be grammatical, and contain no boundary symbols. On the other hand, there might be boundary symbols left over at the end of the last cycle. Suppose in the worst case that none of the finite number of phrase markers whose longest subsentence has lengthy results in a sentence. Now it must be the case that for some longest subsentence of length z, less than y, at least one sentence can be derived of length greater than y. If this were not so, there would be no grammatical strings longer than y, which is impossible if the language is infinite. Summing up, in the local filtering case also, there must be another grammatical string whose length is at most My.

Theorem 1 can be strengthened to Theorem 2, remembering that non-filtering grammars are a subset of local filtering ones:

Theorem 2 For any infinite local filtering language L, there are constants M and x such that for any sentence of length $y \geqq x$, there is another sentence of length not greater than My.

Relations of length between grammatical strings have some connections with hierarchies of grammars. For instance, consider the language generated by this context-sensitive grammar:

$$X_0 \rightarrow X_1 b$$
$$X_1 \rightarrow X_1 X_2$$
$$X_1 \rightarrow bX_3$$
$$X_3X_2 \rightarrow X_2 X_3 X_3$$

X_3 b $\rightarrow$ b b

b X_2 $\rightarrow$ b b

It is easy to check that this grammar generates a language whose sentences consist of strings of 2^m+m+2 b's, for $m \geqq 0$. Each sentence is a little more than twice as long as the next shortest. The language is not context-free. For consider a context-free derivation of some sentence for some value of m. The penultimate line of that derivation must consist of some number $2^m+m+2-q$ of b's, together with some non-terminal symbol of the CF grammar, X_i, say. To obtain the last line, X_i is rewritten as a string of q b's.

It is well known that CF languages can be generated by grammars all of whose non-terminal symbols are recursive (see, for instance, Stirk, 1987, p104). It must be possible to continue from that penultimate line by rewriting X_i once recursively, and then terminating it by rewriting it as the string of q b's. The recursive rule used in rewriting X_i may bring in further non-terminal symbols, but there must be some minimum number p, say, of b's which these other symbols can dominate. Altogether this means that the grammar will generate a string of $2^m+m+2+p$ b's. Adding an extra p b's in this way cannot, except for a particular value of m, produce a grammatical string. Yet since there are only a finite number of non-terminal symbols in the CF grammar, X_i must be present in the penultimate line of an infinite number of derivations, producing an infinite number of ungrammatical strings. Thus there cannot be a context-free grammar for the language.

There is, of course, a local filtering grammar for that language.

In fact, a non-filtering grammar can easily be constructed. The phrase structure component consists of the rules

$$S \rightarrow bS,\quad S \rightarrow bN,\quad N \rightarrow b$$

The transformational cycle contains the following three rules, applied in this order:

	$X_1 - [_N X_2 - b\ ]_N$			$- X_3$
SD :	1	2	3	4
SC :	1	2	3 + 3	4

	b	$- [_S X_1$	$- N$	$- X_2\ ]_S$
SD :	1	2	3	4
SC :	1 + 3	2	ϕ	4

	b	$-$ N	$-$ S	
SD :	1	2	3	
SC :	3	2	3	

The context-sensitive grammar given above achieved its multiplication because of the rule $X_3\ X_2 \rightarrow X_2\ X_3\ X_3$. That process can be repeated, as in the following grammar:

$$X_0 \rightarrow X_1\ X_4\ b$$
$$X_1 \rightarrow X_1\ X_2$$
$$X_1 \rightarrow b\ X_3$$
$$X_3\ X_2 \rightarrow X_2\ X_3\ X_3$$

$X_3\ X_4 \rightarrow X_4\ X_4\ X_3$

$b\ X_2 \rightarrow b\ b$

$b\ X_4 \rightarrow b\ b$

$X_3\ b \rightarrow b\ b$

Using the notation "a exp(b)" for "a^b", this grammar produces strings of b's of length 2exp $(2^m)+2^m+m+2$. The first term, 2exp (2^m), is the square of 2exp (2^{m-1}), meaning that there can be no constant multiple relating the lengths of sentences. The following theorem follows from this and Theorem 2:

Theorem 3 There are context-sensitive languages which cannot be generated by local filtering grammars.

This might raise one's hopes about the possible linguistic significance of local filtering grammars, until one observes that

Theorem 4 Any UR language can be expressed as the intersection of a finite state language and a local filtering language.

In fact, the local filtering language here can be a non-filtering one. For consider some filtering grammar G for any UR language. If we replace every occurrence of the boundary symbol in the rules of G by a new terminal symbol, c, say, we obtain a non-filtering grammar for a language whose sentences consist of those of the UR language together with others containing c. Now consider a finite state language whose sentences consist of random strings over the vocabulary of the UR language but not c. The intersection of this and the non-filtering

language will clearly exclude just those including c: the intersection will be the UR language itself.

It is also easy to prove this:

Theorem 5 The intersection of two recursive languages is itself recursive.

It is well known that a language L is recursive if and only if both L and $-L$ are recursively enumerable, where $-L$ is the set of possible strings over the terminal vocabulary of L which are not in L. (See, for instance, Stirk, 1988, p203.) Suppose that L_1 and L_2 are recursive languages, and consider $L_3 = L_1 \cap L_2$. We can enumerate L_3 by enumerating L_1 and L_2 and picking out the sentences common to both lists. We can enumerate $-L_3$ just by combining the lists of $-L_1$ and $-L_2$. Thus L_3 is recursive.

We can now go on to prove

Theorem 6 Local filtering grammars can generate non-recursive languages.

For according to Theorem 4, any UR language, in particular some non-recursive one, can be expressed as the intersection of a FS language, which is certainly recursive, and a non-filtering one. So, from Theorem 5, the non-filtering language cannot be recursive.

Thus local filtering grammars generate a class of languages which

cuts across the Chomsky hierarchy: some CS languages are excluded, yet some non-recursive languages are included. There is no clear linguistic significance.

The Survivor Property

An examination of the filtering grammars presented in the first section of this paper shows that most of the subsentences in any transformational derivation disappear utterly by the time we reach the final phrase marker. Peters (1973) investigates how grammars might be restricted if such total obliteration of subsentences is restricted. The most obvious restriction would be to require that every subsentence must have at least one survivor: that is, it must contain at least one terminal symbol which is still present in the final string. In that case, a sentence x terminals long could only have been generated by a derivation involving at most x subsentences. There could only be a finite number of such derivations, meaning that it can be decided whether any string of terminals is generated by a particular grammar or not. The grammars could only generate recursive languages, in other words.

Peters also considers a less restrictive, more subtle survivor property. According to this, and here I quote Peters (1973): "..if Φ is the input domain of any cycle ... and Ψ is the output from that cycle, then Ψ contains more terminal nodes than any subpart of Φ on which the transformational cycle operated earlier in the derivation."

An example will make this more clear. Consider the following

labelled bracketing, where each subsentence is numbered:

$$[_{S1}\ .\ .\ [_{S2}\ .\ .\ ]_{S2}\ .\ .\ [_{S3}\ .\ .\ [_{S4}\ .\ .\ ]_{S4}\ ]_{S3}\ ]_{S1}$$

The lowest sentence is S4, and the transformational cycle may delete it completely, as it contains no sentential subpart. S3, however, must leave at least one survivor - one more than its subpart S4's zero survivors. S2 may leave no survivors, but S1 must leave at least 2: one more than its subpart S3.

With this kind of survivor property, the number of terminal symbols in a string determines not the total number of subsentences in a possible derivation, but rather the derivation's S-depth. On tracing branches from the root S in a tree to terminals, the S-depth would be the maximum number of S nodes passed through. The simple example above has an S-depth of 2.

So a string of x terminals must be transformationally derivable from a structure with S-depth x. To make sure that the language is recursive, however, we must fix a finite upper bound to the total number of subsentences in that structure. Peters estimates this generously.

Firstly we estimate just how much a transformational cycle can shorten a string of terminals. Suppose that n is the maximum number of terms mentioned in any structural description of the transformational cycle, while c is the length of the longest terminal string mentioned. If the output of a very destructive transformation is z

terminals long, the transformation might have operated on a string of n terms each z+c terminals long. It proceeded to delete n−1 of these under a condition of identity, and disposed of a further c terminals as members of a specified set. The input string might thus have been n(c+z) terminals long. The shortening factor of such a transformation is n(c+z)/z, or rather less than n(c+1). If there are k transformations in the cycle, each of them might effect the same shortening, providing a generous estimate of the shortening factor of one cycle of $(n(c+1))^k$.

Now two or more cycles occurring at the same S-depth cannot multiply their shortening effects, because each one operates only on part of a string at that depth. The combined shortening effect could not be more than that of one cycle operating on the entire string at that depth. So, if the S-depth is x, the overall shortening effect would be at most $(n(c+1))^{kx}$, and the maximum length of the string generated by the phrase structure component would be at most $x(n(c+1))^{kx}$.

We now estimate the maximum number of S nodes possible in a phrase marker for a string of z terminals, independently of a particular set of PS rules. It is a problem in recreational arithmetic to show that this number of nodes is $2z-1$. If we call the expression $(n(c+1))^k$, which is a constant for a particular grammar, K, then the maximum number of subsentences underlying a derivation of a string x terminals long amounts to $2xK^x-1$, which is certainly finite.

This proves that grammars with the survivor property can only

generate recursive languages. This does not seem to be of great linguistic significance, especially in light of the possibility that human languages are context-free. I believe it is still an open question whether grammars with the survivor property can generate languages that are not context-sensitive.

The very generous estimate of the maximum number of subsentences in a derivation is an exponential function of the sentence length. In spite of the crudeness of the estimate, Peters provides some evidence that such exponential functions do occur in the structures of natural languages.

Consider Peters' example:

Their sitting down promises to steady the canoe.

Here both the subject and object of "promises" are arguably derived from separate subsentences. Altogether there are 2+1=3 subsentences.

Now consider

Their sitting down's promising to steady the canoe threatens to spoil the joke.

Here again the subject and object of "threatens" are (arguably) subsentences, while the subject itself contains two. Now we have 2. 2+1=5 subsentences.

The method of constructing these sentences can be continued:

Their sitting down's promising to steady the canoe's threatening to spoil the joke purports to follow the script.

The number of subsentences rises exponentially! There is certainly food for thought in this rather neglected article of Peters.

Bibliography

M. Gross, M. Halle and M-P. Schutzenberger (eds) "The Formal Analysis of Natural Languages" (Mouton, 1973)

K. J. J. Hintikka, J. M. E. Moravcsik and P. Suppes (eds) "Approaches to Natural Language" (Reidel, 1973)

P. S. Peters (1973) "On Restricting Deletion Transformations" (in Gross, Halle and Schutzenberger, eds)

P. S. Peters and R. W. Ritchie (1971) "On Restricting the Base Component of Transformational Grammars" (Information and Control 18, pp 483-501)

P. S. Peters and R. W. Ritchie (1973) "Nonfiltering and Local-Filtering Transformational Grammars" (in Hintikka, Moravcsik and Suppes, eds)

Ian C. Stirk (1987) "Context-Free Languages Revisited Yet Again" (大阪外大英米研究 15, pp 103-132)

Ian C. Stirk (1988) "Counting Languages" (大阪外大英米研究 16, pp 191-209)

Ian C. Stirk (1990) "The Continuing Importance of Peters and

Ritchie" （大阪外大英米研究　17, pp 41-59）