Cut Down the Trees, and Save the Environment

Ian C. Stirk

I'm afraid the explanation of my cryptic title will have to wait till the end of a rather lengthy

Introduction

In early work on Generalized Phrase Structure Grammar (see, for example, Gazdar, 1981 and 1982), it was regularly pointed out that context sensitive rules could be used in a grammar without giving it more than context free powers, provided only that the rules of the grammar were treated as node admissibility conditions. Oddly enough, neither then nor since, as far as I know, has this useful possibility actually been made use of in any variety of Phrase Structure Grammar. It is not mentioned at all, for instance, in Gazdar, Klein, Pullum and Sag (1985).

That fact about node admissibility conditions was proved by Joshi and Levy in their (1977). I presented a different and simpler proof in my (1987). That proof depended on a result of Salomaa (1973, p68ff), to the effect that any context free language can be represented as a homomorphic image of the intersection of a finite state language and a Dyck language.

It is useful to sketch that result again here. If we use the familiar old CF language a^nb^n as an example, and generate it with the following grammar:

(1) $X_0 \rightarrow X_1X_2$

(2) $X_2 \rightarrow X_0X_3$

(3) $X_0 \rightarrow X_1X_3$

(4) $X_1 \rightarrow a$

(5) $X_3 \rightarrow b$

then derivations like the following emerge:

X_0

X_1X_2

$X_1X_0X_3$

$X_1X_1X_3X_3$

$X_1X_1X_3X_3$

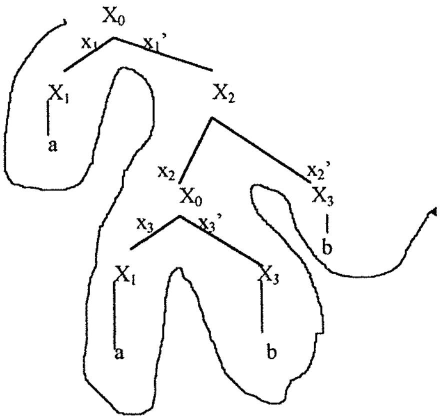
$aX_1X_3X_3$

aaX_3X_3

$aabX_3$

aabb

The tree structure associated with this derivation, together with a few bits and pieces, would be:



The bits and pieces include a labelling for some of the branches in the tree. There are three rules in the grammar which rewrite a symbol as a pair of symbols, meaning that there will be three kinds of paired branches, each kind receiving a labelling of the form x_i, x_i'. Now if we travel round the tree in the manner shown by the wiggly arrowed line, we pass the various labels and terminal symbols in this order:

x_1 a x_1' x_2 x_3 a x_3' b x_2' b

The insight into the result is that the paired labels behave like a proper bracketing of left hand and right hand brackets. After passing x_2 and x_3 we cannot reach x_2' before x_3'. The brackets must be closed in the proper order.

It will be handy later to have a brief way of referring to *left hand brackets* and *right hand brackets*. The physicist P.A.M. Dirac used the terms "bras" and "kets" for similar brevity in his notation for quantum mechanics. I think I shall try "bracs" and "kets" and avoid any ambiguity.

Now Dyck grammars generate sets of brackets only, so in what follows I generalise them a little to what shall be called bracketing grammars. A suitable *bracketing grammar* to generate the result of the tree tour above, together with an infinity of other sentences, would be:

(1) $Y \rightarrow YY$
(2) $Y \rightarrow x_1 Y x_1'$
(3) $Y \rightarrow x_2 Y x_2'$
(4) $Y \rightarrow x_3 Y x_3'$
(5) $Y \rightarrow a$
(6) $Y \rightarrow b$

It is easy to see that this grammar will generate correct bracketings with the three sorts of brackets, together with a random sprinkling of a's and b's, by virtue of rules 1, 5 and 6. Bracketing grammars are only allowed to have one non-terminal symbol, and rules of only three kinds, exemplified above by 1, 2 and 5.

The following FS grammar will also generate the tour, among any number of other things:

(1) $Z_0 \rightarrow x_1 Z_1$

(2) $Z_2 \rightarrow x_2 Z_0$

(3) $Z_0 \rightarrow x_3 Z_1$

(4) $Z_1 \rightarrow a x_1' Z_2$

(5) $Z_1 \rightarrow a x_3' Z_3$

(6) $Z_3 \rightarrow b x_2' Z_3$

(7) $Z_3 \rightarrow b$

The FS rules are clearly related to those of the original CF grammar. It can be seen how the FS rules attach the terminal symbols a and b to the correct kets. Thus the intersection of the bracketing and the FS languages consists of sentences of the CF grammar, together with unwanted bracs and kets. "Homomorphic image" is just a fancy way of referring to the deletion of the unwanted items, leaving a CF sentence.

My version of the context sensitive node admissibility condition proof involved adding extra pairs of symbols to the bracketing language, which were exempt from the bracketing condition, and could occur more freely. These take care of the context sensitivity, and provided the souped up bracketing grammar remains context free, only context free languages can be generated.

In effect, the bracketing grammar with its additions provides two or more systems of brackets which are oblivious to each other. It seemed interesting to probe this possibility further. Suppose, instead of adding extra rules to one bracketing grammar, we just have several simple bracketing grammars. What kind of languages would appear in the intersection between their languages and a finite state one?

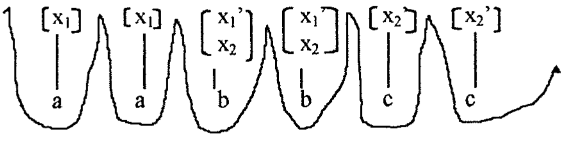
More than just context free ones is the immediate answer. For there is no difficulty in generating the classic context sensitive language $a^n b^n c^n$. The following two bracketing languages, D_1 and D_2, together with the finite state grammar G will do the trick:

(D_1)	(D_2)
$Y_1 \to Y_1 Y_1$	$Y_2 \to Y_2 Y_2$
$Y_1 \to x_1 Y_1 x_1'$	$Y_2 \to x_2 Y_2 x_2'$
$Y_1 \to a$	$Y_2 \to a$
$Y_1 \to b$	$Y_2 \to b$
$Y_1 \to c$	$Y_2 \to c$
$Y_1 \to x_2$	$Y_2 \to x_1$
$Y_1 \to x_2'$	$Y_2 \to x_1'$

(G)

$Z_0 \to x_1 a Z_0$; $Z_0 \to x_1' x_2 b Z_1$, $Z_1 \to x_2' c Z_1$, $Z_1 \to x_2' c$

Notice how D_1 generates equal numbers of x_1 and x_1', but everything else at random, while D_2 does a similar job for x_2 and x_2'. Meanwhile G links every x_1 with an a, and every x_2' with a c, while each b is preceded by both an x_1' and an x_2. This forces the sentences in the intersection to have equal numbers of each, with a's, b's and c's in the correct order. A typical intersection sentence could be represented as follows:



This is read in the order indicated by the arrowed wiggly line. The bracs and kets are segregated in boxes, and are the items eliminated by the homomorphism to leave the final result aabbcc.

Escape from the confines of context freeness might seem to doom languages generated in this way from having any interest to students of human languages. On the other hand, it can be proved that the set of these languages cannot include all context sensitive ones, and is unlikely to include any languages that are not context sensitive. The necessary proofs and speculations are somewhat technical, however, so I have relegated them to an appendix to this paper. Even more significantly, this way of generating languages can accommodate the troublesome examples of non-context-freeness that have occasionally been observed in human tongues.

As an example, I shall use the subset of (possibly) English sentences brought to light by Higginbotham (1984). We begin with the sentence.

The mermaid combed her hair

Adding a subordinate clause to this could produce

The mermaid such that the flying fish jumped from her to him combed her hair

The point about a "such that" relative clause is that it needs to contain a retained pronoun "her" referring to "the mermaid". The sentence is equivalent in meaning to the marginally less clumsy

> The mermaid from whom the flying fish jumped to him combed her hair

There is no need to stop at one relative clause:

> The mermaid such that the mermaid such that the flying fish jumped from her to him swam from her to him combed her hair

and so on. The crucial point in these examples is that each "such that" clause must contain a referring "her" within it. Thus the following sentence is also grammatical, if no less unintelligible:

> The mermaid [such that the mermaid [such that the flying fish jumped from her to her] swam from him to him] combed her hair

Here the two occurrences of "her" appear in the inmost clause. The next sentence is ungrammatical:

> The mermaid [such that the mermaid [such that the flying fish jumped from him to him] swam from her to her] combed her hair

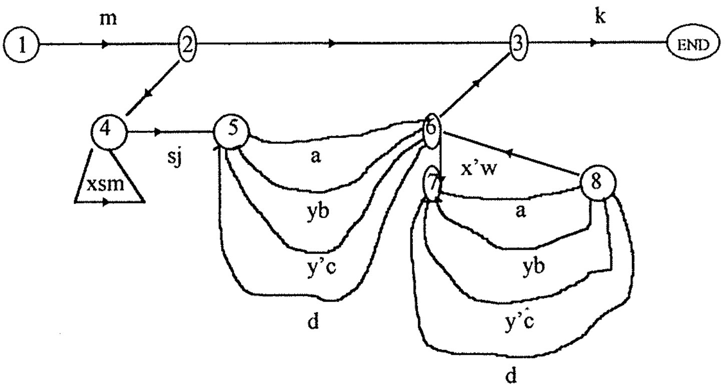
This is because the inmost clause does not contain a "her", even though there are two in the sentence overall. Some reflection shows that, in a grammatical sentence, as you go from left to right *the number of him's you have traversed must never exceed the number of her's you have traversed by more than one.* It is this property that makes the set of sentences context sensitive.

The next task is to generate the set as the intersection of bracketing languages and a finite state language. To save space a few abbreviations will be useful:

m	the mermaid	a	from her to him
k	combed her hair	b	from her to her
s	such that	c	from him to him
j	the flying fish jumped	d	from him to her
w	swam		

The last grammatical example above would abbreviate to msmsjbwck, and the last ungrammatical one to msmsjcwbk. In the abbreviated sentences, the grammaticality condition boils down to saying that as you go from left to right, the number of c's passed must never exceed the number of b's passed.

It is easiest to appreciate the FS grammar needed from its state diagram:



There are two sets of brackets, xx' and yy', which are to be independent. Thus there will be two bracketing grammars, namely:

(D_1) (D_2)

$Y_1 \rightarrow Y_1Y_1$	$Y_2 \rightarrow Y_2Y_2$
$Y_1 \rightarrow xY_1x'$	$Y_2 \rightarrow yY_2y'$
$Y_1 \rightarrow y$	$Y_2 \rightarrow x$
$Y_1 \rightarrow y'$	$Y_2 \rightarrow x'$
$Y_1 \rightarrow a$	$Y_2 \rightarrow a$
etc	etc

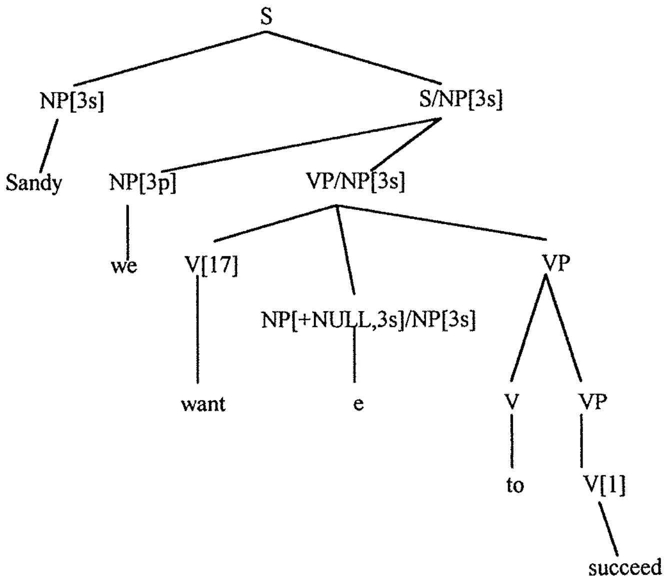
The etceteras obviously cover rules for generating all other terminal symbols at random. Strolling through the FS grammar, or rather its state diagram, we see that from the initial state 1 we must go straight to 2, outputting m, after which we have a choice of going straight to 3 and ending with a k. Otherwise we can go from 2 to 4, and thence loop with an output of xsm, or go straight to 5 emitting sj. From 5 there are four possible routes to 6, with outputs of a, yb, y'c and d respectively. After 6, there are paths to 3, emitting nothing, or through a loop back to 6, on which x'w and one of a, yb, y'c or d must be produced.

Now clearly D_1 will ensure the xx' pairs, meaning that the beginning of each clause has precisely one conclusion. As for y, it will only appear when the abbreviation for two "her"s is chosen, thus preempting a possible later "her". There cannot be too many "her"s, though, because y' must come up later together with the abbreviation of "from him to him". The resulting set of sentences, then, is the one required.

So these languages, the homomorphic images of the intersections of a finite number of bracketing languages and one finite state language, may well have a significance in the analysis of human languages. They do not have the possible weakness of the context free, nor the unwanted power surge of the totally context sensitive. In fact they deserve some special name, so, for reasons given in the next section, I shall call them treeless languages.

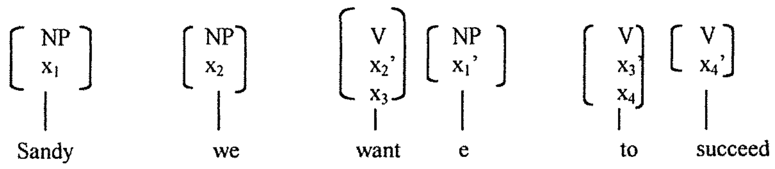
Down with the Trees!

To provide an example almost at random, I reproduce a tree diagram from Gazdar, Klein, Pullum and Sag (1985, p145):



Here we can see something of all the paraphernalia of phrase structure grammars. The vast categories at each node (of course grossly abbreviated), the head feature conditions for passing feature values down the trees, the interconnecting slash features to relate "Sandy" to the rest of the monster, and so on. And all this passing down and passing up of feature values is, it seems to me, due only to not taking advantage of the possibility of using context sensitivity without being carried out to sea by it.

Suppose the trees were just cut away, leaving just a string of categories and the words that go with them, as I intended to show by the diagram on page 4. We might get this:



Bracs and kets can occur in categories like features, linking them directly. The x_1,x_1' bracket links "Sandy" with the later hole, x_2,x_2' links the subject "we" with the verb "want" in its correct form, x_3,x_3' links the verb "want" to an infinitive, x_4,x_4' links "to" with a verb "succeed" in base form, and so on. Why do we need vast trees to perform these simple correlations?

What is more, at least one problem associated with trees can be dispensed with, namely the problem of excessive structure that can go with rules like NP$\rightarrow$ NP and NP. Gazdar, Klein, Pullum and Sag, like others, need "coordination schemes" (1985, p248) to evade this problem.

Of course I am neglecting here the semantic component, the translation of a sentence into intensional logic, at least in earlier phrase structure grammars. There is no room for details here, but intensional logic is after all a language, and could also be treated as a treeless one, so that a sentence in ordinary language would be correlated with another one in intensional logic.

This might not work for the Barwise and Perry (1983) kind of semantics which is used, for instance, in "Head-Driven Phrase Structure Grammar" (Pollard and Sag, 1994), which may not be expressible in sentences. On the other hand, we may note the devastating criticism of situational semantics to be found in Cresswell (1985).

Conclusion

A somewhat lame conclusion to come to, perhaps, but actual natural language examples of treeless languages will have to await a further paper. Meanwhile, I think I have laid most of the technical foundations for a formulation of grammars which

(1) is slightly more powerful than CF grammars, as seems to be required,
(2) simplifies the nagging problem of coordination, and
(3) simplifies the whole system of categories and their interrelations.

Further, I believe there is a simple reason why the structures of human languages can so easily be expressed in treeless form, but its revelation must await further work.

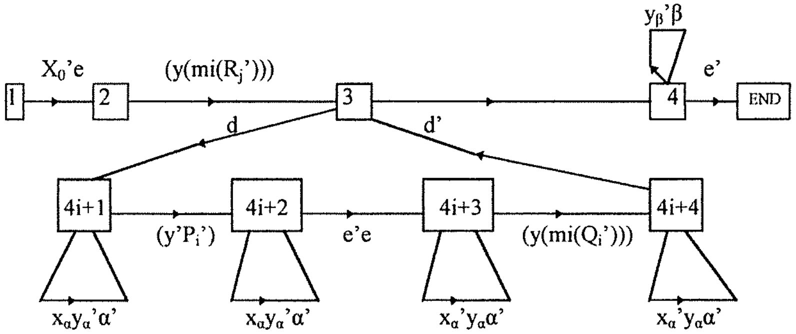
Technical Appendix

Firstly I prove that these intersections of bracketing languages and finite state languages are dangerously close to being able to generate the full monty of unrestricted rewriting (UR) languages, that is, virtually anything rule governed. At least, with a possibly essential addition to the power of the bracketing grammars, and a generously purgative homomorphism, that terror may be unleashed. For consider any arbitrary UR grammar.

Suppose it has a non-terminal vocabulary V_N, a terminal vocabulary V_T, and that V stands for $V_N \cup V_T$. We construct a disjoint vocabulary V' so that any member α of V has a counterpart α' in V'. The symbols α and α' will be used to represent individual members of V and V' in what follows, while, if s is some string over V, s' represents the string over V' in which each symbol of s has been replaced by its counterpart. Thus if the ith rule in the UR grammar is $P_i \rightarrow Q_i$, then $P_i' \rightarrow Q_i'$ is the same rule using V' instead of V. The symbol R_j will be used to represent the right hand side of the jth rule of the UR grammar which has X_0, the initial symbol, as its left hand side. There must of course be at least one such rule in the grammar. Finally β will represent any member of V_T.

As well as V', we shall need two pairs of brackets, x_α, x_α' and y_α, y_α' for each member α of V. If s' is a string over V', then (ys') is the string in which each symbol of s' is preceded by the corresponding brac. Thus if s' is a'b'c', (ys') will be y_aa'y_bb'y_cc'. Similarly for (y's'). Two other pairs of brackets will also be used: d, d' and e, e'. Finally, mi(s) is to stand for the *mirror image* of a string s. If s is abc, then mi(s) will be cba.

It is easiest to appreciate the structure of the FS language L by looking at a simplified state diagram of its grammar:



In moving from the initial state 1 to state 2, only X_0'e can be output, meaning that every sentence of the FS language must begin with X_0'. From 2 to 3, however, there are a number of possible paths, which here have been compressed into one for simplicity. There will in fact be a fresh path for each rule of the UR grammar which rewrites X_0 as something. This is represented by the subscript j. From state 3, we can go directly to 4, outputting nothing, or follow the loop through four other states before returning to 3. Or rather, one of the loops, for there will be a different one for each of the UR rules from $P_1 \rightarrow Q_1$, the first one, to the last. When i is 1, we pass through states 5, 6, 7 and 8, when i is 2, through 9, 10, 11, 12 and so on.

In the general case, after outputting d and reaching state (4i+1), we can either go on to (4i+2) and produce the left hand side of a UR rule in vocabulary V' and accompanied by y-kets, or we can take any number of loops first. Once more, there is a simplification here: there is a separate loop for each $\alpha' \varepsilon$ V'. Each pass through a loop will introduce a member of V' preceded by its x-brac and y-ket. From (4i+2) we can make any number of similar loops before outputting e'e and going on to (4i+3). A somewhat different kind of looping is available from (4i+3): this time each member of V' is preceded by an x-ket and a y-brac. Going from (4i+3) to (4i+4) produces the mirror image of the right hand side of the UR rule, written in V' and accompanied by y-bracs. From (4i+4) there is the possibility of more looping before producing a d' and returning to 3.

State 4 can be reached from state 3, and from it we can either go directly to the final state, outputting an e', or make some loops first. There is one of these loops for each member β of V_T, preceded by its y-ket.

We are not interested in the entire FS language, but only in those sentences which can also be generated by the two *almost* bracketing languages, D_1 and D_2. D_1 has the initial symbol Y_{10} and these rules:

(1) $Y_{10} \rightarrow eY_1e'$

(2) $Y_{10} \rightarrow Y_{10}Y_{10}$

(3) $Y_1 \rightarrow y_\alpha Y_1 y_\alpha'$

(4) $Y_1 \rightarrow Y_1Y_1$

(5) $Y_1 \rightarrow \alpha'$

(6) $Y_1 \rightarrow \beta$

(7) $Y_1 \rightarrow x_\alpha$

(8) $Y_1 \rightarrow x_\alpha'$

(9) $Y_1 \rightarrow d$

(10) $Y_1 \rightarrow d'$

All of these rules, except for 1, 2, 4, 9 and 10 are in fact schemata: there is a rule of forms 3, 5, 7 and 8 for each $\alpha \in V$, and one of form 6 for each $\beta \in V_T$.

Clearly rules (schemata) 5 to 8 enable the grammar to produce random strings of symbols from V' and V_T, together with x-bracs and x-kets. 3, however, ensures that y-bracs and y-kets are always paired. D_2, on the other hand, with initial symbol Y_{20}, pairs x-bracs and x-kets while scrambling everything else:

(1) $Y_{20} \rightarrow dY_2d'$
(2) $Y_{20} \rightarrow Y_{20}Y_{20}$
(3) $Y_2 \rightarrow x_\alpha Y_2 x_\alpha$
(4) $Y_2 \rightarrow Y_2Y_2$
(5) $Y_2 \rightarrow \alpha'$
(6) $Y_2 \rightarrow \beta$
(7) $Y_2 \rightarrow y_\alpha'$
(8) $Y_2 \rightarrow y_\alpha'$
(9) $Y_2 \rightarrow e$
(10) $Y_2 \rightarrow e'$

D_1 and D_2 each have two non-terminal symbols, instead of the one of a true bracketing grammar. Notice that the effect of the extra rules is to constrain the bracketings to a series of independent blocks, bounded by occurrences of e and e' and d and d' respectively.

Let us have a look at what kinds of sentences can be generated by all three of these grammars. Every one begins with X_0', for that is prescribed in L and allowed by D_1 and D_2. After that we get a string of symbols preceded by e, which, apart from the fact that it is written in reverse and is scattered with y-bracs, could be the second line of a derivation using the UR grammar. The developing sentence must also be generable by D_1, so the presence of the y-bracs means that any following y-kets must be paired, in reverse order. The y-kets can come, whichever path is followed from state 3: either on the way to state 4i+3, or by looping from state 4. Thus the next stretch of symbols will have to be y' Rj followed by e', apart from the possible addition of some x-bracs. In other words, we basically have the second line of a derivation of the UR grammar, this time in its usual order. Moreover, if we have now reached state 4i+3, this second line will contain some P_i', the left hand side of some UR rule which could lead to a third line of some successful derivation.

D_2 will force any x-bracs in this second line to be matched by some x-kets, coming along in reverse order. This means that any loops that we make from 4i+3 and 4i+4 will have to generate paired x-kets. So by the time we output d' and return to 3, we shall have a reversed third line of a derivation, which differs from the second line only in that Q_i' (backwards) has replaced P_i'. Taking a further loop from 3 will continue the process: first a line identical to the third except that it is in the correct order and contains x-bracs will be formed, leading on to a fourth line and so on.

An end comes when the path to 4 is selected. Now a series of terminal symbols together with y-kets is generated, to match a previous line with the y-bracs and some Q_i' in it. This previous line can only contain members of V_T', apart from bracket symbols, because of the match. This is only possible if it represents the last line of a *complete* derivation of the UR grammar. The last series of terminal symbols must be a correct sentence of the UR language, in that case.

The reason for including the d,d' and e,e' brackets can now be appreciated. Consider e,e'. If any y_α,y_α' pair is represented by (,), and e,e' itself by {,}, and other symbols are ignored for the sake of simplicity, then two "cycles" of the FS grammar, each going from some e to some e', could be pictured like this:

{ (((( )))) } { (((((( )))))) }

The structure of the grammar D_1 ensures that each e,e' pair contains a complete set of y_α,y_α' pairs, while the FS grammar prevents any patterns like the following:

$$\{ (( ) ((\;\;))) \} \{ ((((((\;\;)))))) \}$$

All bracs have to come before all kets inside each e,e' block. If it were not for the e,e' system, though, then nothing could prevent the following sort of pattern over the two "cycles":

$$(_1 (_2 ((\;\;)) \quad ((((((\;\;)))))))_2)_1$$

Here the first two bracs are not paired with their kets until the end of the second "cycle". In terms of the UR derivation, this would mean the possibility of successive lines being different in ways not allowed by the rewriting rules. Similar considerations apply to x_α,x_α' pairs.

This shows that even a seemingly trivial increase in the powers of bracketing languages will take us straight into the realm of generative omnipotence. But it does not show that UR languages cannot be generated as homomorphisms of the intersections of a FS language and a finite number of bracketing languages. Perhaps I just was not clever enough to find the right way to do it.

Fortunately it is quite easy to find a context sensitive language which is not treeless, and so eliminate the possibility that they can generate absolutely any language. The CS language is generated by a very simple grammar:

(1) $X_0 \rightarrow X_1c$
(2) $X_1 \rightarrow X_1X_2$
(3) $X_1 \rightarrow bX_3$
(4) $X_3X_2 \rightarrow X_2X_3X_3$

(5) $X_3c \rightarrow cc$

(6) $bX_2 \rightarrow bb$

The first two rules enable the generation of strings consisting of an initial X_1, any number (say n) of X_2's, and a final c. Using rule 3 halts the production of X_2's, and places an initial b. Rule 4 is the one that makes this grammar interesting. Every time an X_3 jumps over an X_2, it doubles, so that eventually a large number of X_3's emerge to the right of the X_2's. With n X_2's, 2^n X_3's will be produced in this way. Rules 5 and 6 change all the X_3's into c's and all the X_2's into b's.

Sentences of the language thus consist of a string of (n+1) b's, where $n > 0$, followed by a string of (2^n+1) c's.

Suppose first that this language is treeless. Thus there is some FS grammar, and some finite number of bracketing grammars, and some homomorphism which between them will generate its sentences. Consider first the finite state component. The FS grammar must allow recursion, or otherwise only a finite number of sentences could be generated by it. Suppose there is a sentence with a chunk of length t which could be multiplied by recursion. The sentence with the chunk doubled to 2t, but otherwise the same, would also be grammatical in the FS language, but if the doubling added only bracs or only kets belonging to one or more bracketings, then it would be ungrammatical in at least one of the bracketing languages. But there must be some other chunks, of lengths s_1, s_2, ..., s_p, say, which could be multiplied recursively, and so ensure that with, say, k_1, k_2, ..., k_p repetitions respectively, together with m repetitions of the original chunk, we could produce something acceptable by all the grammars. At least, unless this is true for at least some chunks in some sentences, there can only be a finite number of grammatical ones altogether.

An actual example will make this clear. Consider this one bracketing grammar:

(D) $Y \rightarrow YY$, $Y \rightarrow xYx'$, $Y \rightarrow a$, $Y \rightarrow b$

and this finite state one:

(G) $Z_0 \rightarrow xxxxxaZ_0$, $Z_0 \rightarrow xxxxxaZ_1$, $Z_1 \rightarrow x'x'x'bZ_1$, $Z_1 \rightarrow x'x'x'b$

G can provide recursive chunks xxxxxa and x'x'x'b, but the bracketing grammar will only accept sentences with equal numbers of bracs and kets. The briefest way to ensure this is to have three lots of xxxxxa and five lots of x'x'x'b, to give 15 bracs and 15 kets, so the shortest sentence of the treeless language will be aaabbbbb. Now we know that three repetitions and five repetitions will give something grammatical, we can be sure that 6 and 10, 9 and 15 and so on will work as well, giving sentences of twice the length of the shortest sentence, three times the length, and so on.

The same must be true in the general case we were considering before. If there is a grammatical sentence with parts of length k_1s_1, k_2s_2, ... , k_ps_p and mt, there will also be one with parts of length $2k_1s_1$, $2k_2s_2$, ... , $2k_ps_p$ and 2mt, and so on. Of course the homomorphism will reduce the lengths by removing bracs and kets, but in general we can argue that, for some values of K and L, then, in any particular treeless language, there is a grammatical sentence of length K+L, and also ones of lengths K+2L, K+3L, etc.

We are assuming that the CS language whose sentences consist of (n+1) b's followed by (2^n+1) c's is treeless, so for some values of K, L and n we must have

$K+L = 2^n + n + 2$ and also

$K+2L = 2^{n+x} + n + x + 2$, since this is also of grammatical length.

A bit of arithmetic shows that $L= 2^{n+x} - 2^n + x$, and

$K= 2^{n+1} - 2^{n+x} + n - x + 2$. This means that K would be impossibly negative if x is greater than one, so K+2L is the length of the next longest sentence, $2^{n+1} + n + 3$.

A sentence of length K+3L, or $3.2^n + n + 4$, if we work it out, must also be grammatical. But according to the CS grammar, the length of the next longest after $2^{n+1} + n + 3$ would be $2^{n+2} + n + 4$, or $4.2n + n + 4$. The one of length K+3L is too short, and so cannot be grammatical.

The contradiction shows that here we have a CS language which is not treeless, as required.

This result is not enough to make treeless languages a proper subset of CS ones, however. There are sets of languages which do not include some CS ones, but which do include non-recursive ones, for instance, the set of *local filtering languages* discussed by Peters and Ritchie (1973).

Fortunately it is not difficult to show that treeless languages must be recursive. A language is recursive if and only if both L, the set of sentences of the language, and -L, the set of strings over the same vocabulary which are not in L, are both recursively enumerable. Proofs of this can be found in Salomaa (1973) and Stirk (1988).

We check if it is possible to enumerate both these sets in the case of a treeless language. First we enumerate all possible strings over the vocabulary of the treeless language (by following alphabetical order and increasing length). These strings are checked in turn to see if they can be generated by the FS grammar concerned.

This process should be finite, since clearly a FS grammar can only generate a finite number of strings up to a certain length, but here we have the complication of bracs and kets, made invisible in the treeless language by the action of the homomorphism. It has already been stipulated that only brackets should be deleted by the homomorphism, but bracketing languages may still generate any number of brackets to go with any terminal symbol, for instance, xax', xxax'x' and so on. We can halt this proliferation by requiring that each rule of the FS grammar that generates bracs or kets must also generate at least one terminal symbol. In that way, each terminal symbol will only appear associated with a finite number of bracs and kets. This proviso has been tacitly observed above.

Now we can use it to show that each possible terminal string needs only a finite check to see whether or not the FS grammar can generate it, for there can only be a finite number of strings in the FS language which contain all and only the terminal symbols in correct order plus a certain finite number of bracs and kets. These can each be checked in a finite way to see whether or not they can be generated by all the finite number of bracketing languages. If one of them can be so generated, the string of terminal symbols can be added to the growing list L, if not, to the growing list -L. This is enumerating both L and -L, so the treeless language must be recursive.

We still cannot say that treeless languages are all context sensitive, however, because unfortunately there are some recursive languages that are not CS, as is shown in Salomaa (1973) and Stirk (1988). It seems unlikely that any of these exotic languages could be treeless, but I have not so far devised any formal proof of this.

Bibliography

M. J. Cresswell (1985) "Structured Meanings" (MIT Press)

G. Gazdar (1981) "Unbounded Dependencies and Coordinate Structure" Linguistic Inquiry, 12

G. Gazdar (1982) "Phrase Structure Grammar" in Jacobson and Pullum, eds

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

J. Higginbotham (1984) "English is not a Context-Free Language" (Linguistic Inquiry,
15, 225-234)

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

P. Jacobson and G. Pullum (eds) (1982) "The Nature of Syntactic Representation" (Reidel)

A. K. Joshi and L. S. Levy (1977) "Constraints on Structural Descriptions: Local Transformations" (Siam Journal of Computing, 6, 272-284)

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

C. Pollard and I. Sag (1994) "Head-Driven Phrase Structure Grammar" (U. of ChicagoPress)

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

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)