CS 381 Homework 10 solutions

November 6, 2000

Problem 24.1 Consider the set $a^*b^*c^* - \{a^nb^nc^n | n \geq 0\}$. Give a CFG for the set and prove your grammar is correct.

This may not be the shortest solution, but it was the first I came up with. What the grammar does is ensure that either $\#b \neq \#c$ (through S_1) or $\#a \neq \#b$ (through S_3).

$$
\begin{aligned}
S &\rightarrow S_1 | S_3 \\
S_1 &\rightarrow aS_1 | S_2 \\
S_2 &\rightarrow bS_2c | B | C \\
S_3 &\rightarrow S_3c | S_4 \\
S_4 &\rightarrow aS_4b | A | B \\
A &\rightarrow aA | a \\
B &\rightarrow bB | b \\
C &\rightarrow cC | c
\end{aligned}
$$

Since a formal proof would take too much space and time, and we're all sick of them after the previous homeworks, I'll be brief:

To show: S generates strings in $a^*b^*c^* - \{a^nb^nc^n | n \geq 0\}$. We'll examine S_1; the case for S_3 is similar. S_1 lets us generate as many a's as we wish (as a prefix). Then we must eventually go to S_2. S_2 ensures the number of b's and c's are not the same, by generating an equal number of b's and c's, then forcing a choice of at least one more b (through the B production) or at least one more c (through the C production). It is clear that the string is of the form $a^ib^jc^k$, and that $j \neq k$.

To show: a string of the form $a^ib^jc^k$ with at least one of ($i \neq j$, $i \neq k$, or $j \neq k$) true can be generated by S. There's really only 2 cases here: $i \neq j$ or $j \neq k$. Suppose the string is in the language by virtue of $i \neq k$. Well then either ($j = k$) or ($j = i$) or ($j \neq i$ and $j \neq k$). The third case is done. If $j = k$ then $i \neq j$. If $j = i$ then $j \neq k$. So now we must merely look at whatever string x is desired and see whether $i \neq j$ or $j \neq k$. The cases are similar; I will examine the $i \neq j$ case. We start with $S \rightarrow S_3$, and after generating k c's we then generate matching a and b until we've generated the smaller of i or j. Then we finish with as many more a's or b's as are needed.

Problem 24.2 f)

$$
\begin{aligned}
S &\rightarrow AC \\
A &\rightarrow aAb | ab \\
C &\rightarrow cCd | cd
\end{aligned}
$$

g) Not a CFL – consider $z = a^mb^{m+1}c^md^{m+1}$. Since $|vwx| \leq m$, either one or both of v and x are all a's, all b's, all c's, all d's, or one is of the form a^ib^j, b^ic^j, or c^id^j. In any of these cases, it is clear that uv^2wx^2y is not an element of the language, as it is either not of the form $a^ib^jc^kd^j$, or we no longer have one of $i = k$ or $j = l$.

h)

$$S \to aSd|aBd$$
$$B \to bBc|bc$$

Problem 24.3 For each part, I'll use L to be the language discussed in that part.

a) Not CFL – consider $a^*b^*c^* \cap L_a = \{a^n b^n c^n | n \geq 0\}$.

b) Not a CFL – consider a^{2^k}. $uv^2wx^2y = a^{2^k+j}$, where $j < k$, which is clearly not a power of 2.

f) Not regular – $a^*b^* \cap \sim L_f = a^n b^n$. $S \to aSb|A|B$, $A \to aA|a$, $B \to bB|b$ generates L_f.

m) Not regular – $a^*b^* \cap L_m = \{a^n b^m | n > m\}$ which we've seen to not be regular. Consider an NPDA M which pushes an arbitrary (non-zero) number of B's onto the stack, then everytime it reads an a it pops a B if possible otherwise pushes an A, for each b it reads pops an A if possible otherwise pushes a B, and accepts by empty-stack. $L(M) = L_m$.

Problem 25.1

0			
A	1		
$\emptyset$	A	2	
S, B	S, B	B	3

Problem 26.1

a) The idea is keep track of the number of distinct parse trees a non-terminal A can generate when we add A to V_{ij}. For this we need a three dimensional array C_{ij}^A which gives the number of parse trees generated by A for the string of length j beginning at position i

The following changes are to be made to the algorithm given in HO 25.

1. $C_{ij}^A = 1$. (just before line 2. this should be a part of the for-loop beginning at line 1.)
2. $C_{ij}^A = C_{ij}^A + C_{ik}^B * C_{i+k,j-k}^C$. (after line 7 , before *end.*)

Finally, the number of distinct parse trees is given by C_{1n}^S.

Construction of a Parse tree if it exists

Number the productions in grammar. If A is added to V_{ij} because of a B in V_{ik} and C in $V_{i+k,j-k}$, then record $\{(i,k),(i+k,j-k),l\}$ (where $A->BC$ is the l^{th} production) with the A in V_{ij}. If there are more than one way of getting A in V_{ij}, just record one such way. At the end if V_{1n} has S then, using the indices & the production numbers stored, in linear time, the parse tree can be built.

b)

1						
0	1					
1	1	1				
0	0	0	1			
2	1	1	1	1		
0	0	0	0	0	1	
5	2	2	1	1	1	1