COMP 598

Talise Wang 260829722

May 25, 2020

Question 1

1.1

We will prove this by induction.
So base case:ϵ (for empty strings) It is obviously balanced.
Induction step: Assume it is well balanced up to k applications of rules. Prove it is true for $k+1$ applications.
Suppose there's a string produced by up to k operations of the rules. Then for the $k+1$ oprations,we have the following cases.
Case 1:For S in that string. $S \to SS$
Since S can only be SS, ϵ and (S), S is always well-balanced since we always add pairs of parentheses with each left parenthese always shows before its matching right parenthese. So obviously, it's well-balanced. So by what we proved, both S on the right side are well-balanced. Concatinate 2 well-balanced strings together is still well-balanced string.
Case 2:$S \to (S)$
Since the original string is well-balanced and S itself is balanced. Then if we add a pair of parentheses to it. Wherever the S is, the whole string is still balanced. So proved!
Case 3: $S \to \epsilon$
Since we proved above, S is always balanced. So if we change it into ϵ. It will remove the matched pairs of parentheses. This means the left parenthese with its matched right parenthese will disappear together. So obviously, the remaining string is still balanced since we didn't change the remaining parentheses' place order and matching.
So proved!

Combine above together, we proved!

1.2

We will prove this by counting the nuumber of pair of parentheses using the induction:
Base case: $n = 0$ number of pair is zero, which means empty string. It can clearly be generated.
Assume this grammar can be generated all balanced strings with up to k pairs, prove it is also holds for $k + 1$.
Suppose a_k is a string with k pairs (it can be any form as long as it has k pairs), Then for a_{k+1} which has $k + 1$ pairs, there are following cases.
Case 1:a_{k+1}=(a_k)
Since a_k can be generated, Clearly, a_{k+1} can be generated with $S \to (S)$ operation. Which means take this operation at the begining then just do the same applications that made the a_k.
Case 2: It is not closed by a pair of parentheses. Then we can define a_{k+1} like $a_{k+1} = a_{\leq k}a'_{\leq k}$. Since both strings has $\leq k$ pars of parentheses, then both of them can be generated by this grammar. Then this means a_{k+1} can be generated by concatinating the applications for $a_{\leq k}$ and $a'_{\leq k}$.
Combine above together. So proved!

Question 2

Example: consider the string aab, we will generate it by two different ways
1. $S \to aSbS \to aSb \to aaSb \to aab$
2. $S \to aS \to aaSbS \to aabS \to aab$
So proved! It's ambiguous.

Question 3

3.1

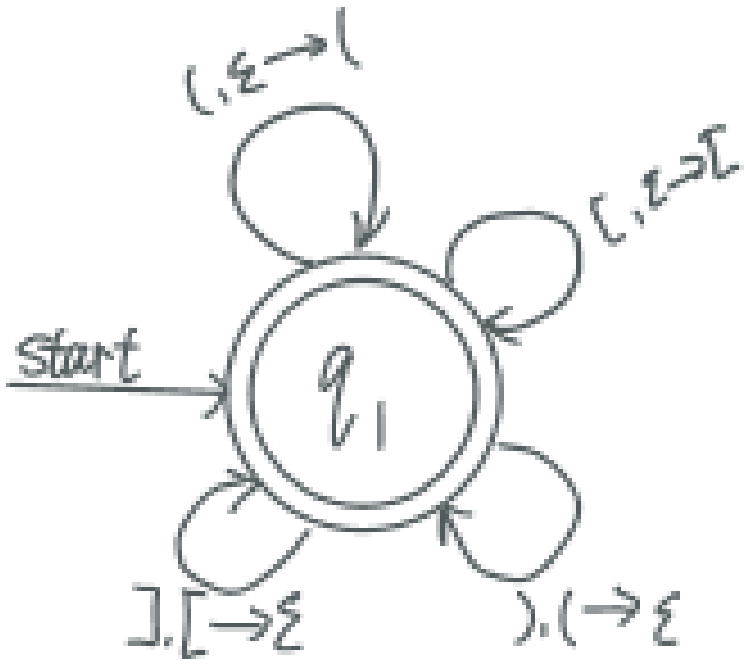
We will prove it by induction.
Base case: n=0, $S \to \epsilon$, empty string, the property holds. Assume this property holds for up to k oprations.

Pick arbitrary string with k operations, then for any S in that string. we will have these following cases:
Case 1: $S \to aS$, then for the prefix which not contain this S, we didn't change anything so it holds the property. For the prefix which contain this S, it will contain aS or just a. In both cases, we just add one more a, So clearly, it holds.
Case 2: $S \to aSbS$, then for the prefix which not contain this S, we didn't change anything so it holds the property. For the prefix which contain this S, there are two cases:
Containing a and aS is the first case, which we just add one more a, So clearly, it holds.
Containg aSb and $aSbS$ is the second case, which we add one in both a and b, since the prefix holds the property before the S changes and add one to both a and b didn't change the ratio between a and b. So it still holds.
Case 3: $S \to \epsilon$, then we didn't change the ratio between a and b. So it still holds. So clearly, it holds the property.
Combine together, we proved!

3.2

We will prove this by counting the length of the strings using the induction.
Base case:length n=0 then the string x can be generated by this grammar by using $S \to \epsilon$
Assume all string up to length k can be generated by this grammar. Then for the string with length $k+1$, we have the following cases:
Case 1: x defined as a string with length $k+1$ and x can be written as $x = ay$ with y as a string with length k and y holds this property. Since y can be generated by this grammar and we can add a in front of y by using $S \to aS \to a\epsilon \to a$ operation, then we can say that x can be generated by this grammar.
Case 2: x can be written as $x = ay$ but y doesn't hold the property. Then for some prefixes in y,number of $a's$ is less than number of $b's$. such that $y = s_l b s_r$ until s_l and s_r both satisfy the property so that $y = a s_l b s_r$. Since the length of both substrings is $\leq k$, we can get s by appying $S \to aSbS$ first, then the productions to generate s_l for the first S, the the productions to generate s_r for the second S.
Combine together, so we proved!

Question 4



Question 5

$S \to X \mid A$
we use X for the case where $n \leq p$ and A for $m \leq p$
$X \to aXc \mid Xc \mid B$
the case for $n \leq p$, by using X, we can generate the a and c first and can make sure that $n \leq p$. then finally generate b
$B \to Bb \mid \epsilon$
B basically generates b^* for X cases
$A \to aA \mid Y$

this generates all the as for A cases
$Y \to bYc \mid Yc \mid \epsilon$
the case for $m \leq p$, by using X, we can generate the b and c first and can make sure that $m \leq p$.
So clearly, this language generates $\{a^n b^m c^p | n \leq p \text{ or } m \leq p\}$

Question 6

6.1

Define $(S, \Sigma, s_0, F, \delta)$ as a DFA that recognizes L. We will construct a left-linear CFG to make sure that $L(G) = L$.
Define the CFG $G = (V, \Sigma', P, S)$
Clearly $\Sigma' = \Sigma$.
To get V. For every state in the DFA, we will create a non-terminal symbol for it such that $V = S$.
For every transition $\delta(s, a) = s'$, we construct a rule such that $s \to as'$
For $q \in F$, create a rule such that $q \to \epsilon$
So each jump is an application of the rule.
Clearly $S = s_0$ for the start symbol. This is clearly left-linear.

6.2

The answer is No. Here comes an example:
Consider the following linear CFG:$S \to aSb \mid \epsilon$ It generates $\{a^n b^n \mid n \geq 0\}$ but clearly, it's not regular(proved in previou assignment & lecture notes).

Question 7

Give an outline of a PDA to recognize it:
And there will be two epsilon jumps, one for $i \neq j$, one for $j \neq k$:
First,We push as to the stack until we reach the first b and pop a a for a b. If the stack is empty before finishing all bs or after we reach the first c, move to an accept state and read b^*c^* or c^* respectively. Then, read a^* until we reach the first b and push all bs until we reach a c. If the stack is empty before or after finishing all cs, move to an accept state.

But it's not deterministic beacuse the complement of a deterministic CFL is still a deterministic CFL. And we will show that the complement of it is not deterministic.
$overline L = \{a^i b^i c^k \mid i = j = k\}$
And clearly, $\overline{L}$ is not even a CFL.

Question 8

Prove that a CFL over one-letter alphabet is regular: $L = \{a^n | n$ with some restrictions$\}$, since we know this language is a CFL, we need to prove it is regular.
Since it's CFL, so it satisfies the pumping lemma for CFL. so there exist $p > 0$,pick arbitrary $s \in L$ with $|s| \geq p$, There exist $u, v, w, x, y \in \Sigma^*$ with $s = uvwxy$, $|vx| > 0$,$|vwx| \leq p$ hold. And for $\forall i \geq 0, uv^i wx^i y \in L$.
So let's define $y' = vx = a^e$ for $e \leq p$. since $|vwx| \leq p$, then define $w = a^r$ with $e + r \leq p$. define $u = a^t$ and $y = a^m$, then since the language is over one-letter alphabet, so change the u, v, w, x, y's place didn't change the result. so we rewrite $uvwxy$ as $s = wvxuy$, all the properties hold the same.
Define $x' = w$, then we get $|y'| > 0$ and $|x'y'| = |wvx| = |vwx| \leq p$ hold. And let $z' = uy$. Then we will get $s = x'y'z'$ with $|y'| > 0$ and $|x'y'| \leq p$. Since $\forall i \geq 0, uv^i wx^i y = wv^i x^i uy = x'y'^i z' \in L$. so by pumping lemma for regular language and we pick s arbitrary, we showed this is regular.
So combine together, we proved!

Question 9

Let $L = \{a^n b^m c^m dde^{3n} \mid m, n > 0\}$. It is a CFL since it can be genarated by the following grammar:
$S \to aSeee \mid aTddeee$
$T \to bTc \mid bc$
lefthalf(L) is not context free since its intersection with a regular language is not context free:
$$\text{lefthalf}(L) \cap \underbrace{\{a^* b^* c^* d\}}_{\text{regular language}} = \{a^n b^n c^n d \mid n > 0\}$$
And $\{a^n b^n c^n d \mid n > 0\}$ is clearly not context free.

Question 10

10.1

False
Since the only way to check if an element is in the union is to check if it's in any of the sets individually, but there is countably infinite sets so you won't halt if you're looking for an element that's not in the set.

10.2

True
call the algorithm described above ALG, so ALG = running each M_n for $n = 0$ to infinite on input x.
We can use dovetailing to run the ALG for all elements in Σ^*, and suspend after a few steps for each x. Every x will be tested eventually. so it will output all elements of union of C_n (output order doesn't matter). So the set of union of C_n is computably enumerable.
So we proved!