COMP 330 Autumn 2021 McGill University

Assignment 4 Solutions and Grading Guide

Remarks for the graders appear in sans serif font.

Question 1[25 points] A sequence of parentheses is a sequence of (and) symbols or the empty sequence. Such a sequence is said to be *balanced* if it has an equal number of (and) symbols **and** in *every* prefix of the sequence the number of left parentheses is greater than or equal to the number of right parentheses. Thus (()()) is balanced but, for example, ())(is not balanced even though there are an equal number of left and right parentheses. The empty sequence is considered to be balanced.

Consider the grammar

$$S \to (S)|SS|\varepsilon.$$

This grammar is *claimed to* generate balanced parentheses.

1. Prove that any string generated by this grammar is a properly balanced sequence of parentheses. [10]
2. Prove that every sequence of balanced parentheses can be generated by this grammar. [15]

Solution

1. We can prove this by induction on the length of the derivation. The base case is a derivation that has a single step. There is only one derivation like that and it can only produce the empty string which is balanced. For the inductive case we proceed as follows. We assume that for all derivations of length up to n the claim that we are trying to prove is correct. Now consider a derivation of length $n+1$. There are two cases: (a) the first rule used is $S \to (S)$ and (b) the first rule used is $S \to SS$. In case (a) we get a string of the form (w) where w is generated from S by a derivation of length n. So w is a balanced string and hence clearly (w) is a balanced string. In case (b) we have a string of the form w_1w_2 where each of w_1 and w_2 are generated by

S by derivations that are of length n or less. Thus each of w_1 and w_2 is a balanced string. Clearly every prefix of w_1 has either (i) more left parentheses than right parentheses or (ii) equal number of left and right parentheses. If we consider a prefix of $w_1 w_2$ that includes all of w_1 and part of w_2; we see that the portion coming from w_1 has an equal number of left and right parentheses while the portion coming from w_2 has at least as many left parentheses as right parentheses. Taken together, such a prefix has at least as many left parentheses as right parentheses. Finally, the entire string $w_1 w_2$ has equal number of left and right parentheses. Thus the string $w_1 w_2$ is balanced. I am happy if they say that every time a right parenthesis is introduced a left is also introduced and the left parenthesis is always to the left of the right parenthesis. Furthermore, every time a right parenthesis is produced a left parenthesis is also produced and the left parenthesis is to the left of the right parenthesis. This means that the final string must have equal numbers of left and right parentheses and the prefix of the string always has more or equal left parentheses as right parentheses. The careful induction I have done is more for practice with induction as the verbal argument I have just given is perfectly rigourous.

2. We prove this by induction on the length of the string of parentheses. The empty string is balanced and can be generated by the grammar: the base case is done. For the induction case we assume that any string of balanced parentheses of length up to and including n can be generated by the grammar. Now we consider a string w of balanced parentheses of length $n+1$. Since it is balanced the first symbol must be a left parenthesis. Now define a function $b(p)$ which is a function of the position in the string. This function is defined to be the number of left parentheses up to position p *minus* the number of right parentheses up to position p. In a balanced string $b(1) = 1$ and for every p, $b(p) \geq 0$. At the end of the string $b(p)$ must be zero. We know that as we move right, the value of $b(p)$ increases or decreases by exactly 1 at each step. Let p_0 be the position where the function attains zero for the first time. This may happen only at the end of the string, for example in (((()))) or it may happen before the end as in ((()))(()). If p_0 is at the end of the word then we know that our string w has the form (w_1) where w_1 is a balanced string. Since w_1 has length $n-1$ by the induction hypothesis it can be generated by the grammar and thus w can be generated from the grammar by starting with the rule $S \rightarrow (S)$ and then using the S produced on the right

hand side of this derivation to generate w_1. If p_0 occurs inside the string w then we break the string into two pieces $w = w_1w_2$ where p_0 is at the end of w_1. Now, clearly, w_2 and w_1 both have to be balanced strings and they are both of length n or less so they can be generated by the grammar. Thus w can be generated by starting with the rule $S \to SS$ and then using the two S's generated to produce w_1 and w_2. This proof requires some argument to justify why one can always break a balanced string into balanced substrings as I have described above. They should lose 7 points if they do not justify this step.

Question 2[15 points] Consider the following context-free grammar

$$S \longrightarrow aS \mid aSbS \mid \varepsilon$$

This grammar generates all strings where *in every prefix* the number of a's is greater than or equal to the number of b's. Show that this grammar is ambiguous by giving an example of a string and showing that it has two different derivations.

Solution Consider the string aab. This can be generated by

$$S \to aS \to aaSb \to aab$$

but also by

$$S \to aSbS \to aaSbS \to aabS \to aab.$$

They must give two distinct trees. Two different derivation sequences is not enough as unambiguous grammars can give different derivations just by altering the order in which non-terminals are expanded. If they give two linearized derivation that correspond to different trees give them full marks but if they give two different derivation sequences that correspond to the same tree give them 5.

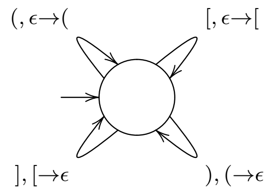
Question 3[15 points] We define the language $PAREN_2$ inductively as follows:

1. $\varepsilon \in PAREN_2$,
2. if $x \in PAREN_2$ then so are (x) and $[x]$,
3. if x and y are both in $PAREN_2$ then so is xy.

Describe a PDA for this language which accepts by empty stack. Give all the transitions.

Solution Please note that when you are recognizing by empty stack you do not need accept states. Note also that you must be at the end of the string in order to accept; this is true whether by empty stack or by accept state.

Here is a simple DPDA to recognise the language $PAREN_2$.



It pushes open parentheses and brackets onto the stack and pops them off only if the matching type is seen. The machine will jam if an unexpected symbol is seen. Since it accepts by empty stack, a word will be accepted only if every open parenthesis or bracket was closed.

Note that it rejects strings like `[(])`.

I did not spend a lot of time in class explaining the difference between empty stack acceptance and acceptance by state. Give them full marks for a correct machine regardless of how they did it.

Question 4[20 points] Consider the language $\{a^n b^m c^p \mid n \leq p$ **or** $m \leq p\}$. Show that this is context free by giving a grammar. You need not give a *formal* proof that your grammar is correct but you must explain, at least briefly, why it works.

Solution
The following grammar G generates the given language, which we will call L.

$$\begin{array}{rcl} S & \to & N \mid AM \mid SC \\ N & \to & aNc \mid B \\ M & \to & bMc \mid \epsilon \\ A & \to & aA|\epsilon \\ B & \to & bB|\epsilon \\ C & \to & cC|\epsilon \end{array}$$

To see that any string generated by G is in L, we analyse the productions. The start symbol S gives the choice between having no fewer cs than as

($n \leq p$ in the question), and no fewer cs than bs ($m \leq p$). The symbol N (respectively M) generates words with as many cs as there are as (bs), while allowing any number of bs (as). The production $S \rightarrow SC$ allows for more cs to be added.

Now we need to show that any word in L can be generated by G. Given a word $w = a^n b^m c^p$ in L, we can generate it with the above grammar as follows. Let $q = \min(n, m)$ and split it $w = w_1 \cdot w_2 = a^n b^m c^q \cdot c^{p-q}$. Then w_1 is in L, and c^{p-q} is generated by C in G, so that if w_1 is generated by G, w will be generated by the rule $S \rightarrow SC$.

Suppose $n \leq m$, then $w_1 = a^n b^m c^n$. The symbol B generates b^m, and by using the production $N \rightarrow aNc$ n times, N can generate w_1. Using the rule $S \rightarrow N$, this gives that w_1 is generated by G as required. If instead $m \leq n$, then $w_1 = a^n b^m c^m$. In this case, the symbol A generates a^n, while the symbol M generates $b^m c^m$. Using the rule $S \rightarrow AM$, this too gives that G generates w_1 and so G generates w.

This question is tricky so mark it generously. Give them 5 marks for the explanation; i.e. if this is completely missing take off 5.

Question 5 [25 points] A *linear* grammar is one in which every rule has exactly one terminal and one non-terminal on the right hand side or just a single terminal on the right hand side or the empty string. Here is an example

$$S \rightarrow aS|a|bB; \quad B \rightarrow bB|b.$$

1. Show that any regular language can be generated by a linear grammar. I will be happy if you show me how to construct a grammar from a DFA; if your construction is clear enough you can skip the straightforward proof that the language generated by the grammar and the language recognized by the DFA are the same.

2. Is every language generated by a linear grammar regular? If your answer is "yes" you must give a proof. If your answer is "no" you must give an example.

Solution

1. Suppose we have a regular language L, there has to be a DFA to recognize it. We fix the alphabet to be Σ. Let the DFA be

$$M = (S, s_0, \delta, F)$$

with the usual meanings of these symbols. We define a linear grammar as follows:
the nonterminals N are identified with the states S of the DFA. Thus for any state $p \in S$ we introduce a nonterminal P in our grammar. The start state of the DFA is identified with the start symbol S of the grammar. Now for every transition $\delta(p, a) = q$ we have a rule written as $P \to aQ$. For every accept state $q \in F$ of the DFA we have a rule $Q \to \varepsilon$ in the grammar. It is easy to see that every run through the automaton exactly produces the sequence of rules needed to generate the string. Yes, I do know that most of you Googled it.

2. The answer is “no”. If we defined a *left*-linear grammar in which we insisted that the terminal symbol appears to the left of the nonterminal and then asked if left-linear grammars always produce regular languages then the answer would be “yes”. The same could be said of right-linear gramamrs. But in the definition above I have allowed you to mix left and right rules. Consider the grammar below

$$S \to aA \mid \varepsilon \qquad A \to Sb.$$

This generates our old friend $\{a^n b^n | n \geq 0\}$.

I think everyone will get the first part easily. I didin’t ask for a proof so if the construction is clear give them full marks. For part 2, people may have looked up random definitions on the web. Give them an IMMEDIATE ZERO if they did not use my definition. They need to learn that a definition I give is the one they have to use. If they come to complain send them to me; don’t waste time arguing with them about it. However, if they use a definition they found elsewhere and proved that it is equivalent to my definition then, of course, they should get full marks.