COMP 330 Assignment 4

Name: Yuhao Wu
ID Number: 260711365

2018-10-25

Question 1:

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

$$S \to (S) \mid SS \mid \varepsilon$$

This grammar is claimed to generate balanced parentheses.

- Prove that any string generated by this grammar is a properly balanced sequence of parentheses.
- Prove that every sequence of balanced parentheses can be generated by this grammar.

(a):

We will show "any string generated by this grammar is a properly balanced sequence of parentheses" by induction on the number of derivation steps.

- **Base Case:**
  If the derivation has only one step then the derivation must be $S \to \varepsilon$, obviously, we have ε is a properly balanced sequence of parentheses.

- **Inductive Step:**
  We assume for w, after n $(n \leq k)$ derivations, we will get a properly balanced sequence of parentheses.

  So, we now consider an $(n+1)$ step derivation of w, it has two forms:

  1 $S \to (S) \overset{*}{\to} (x) = w$ in this case, $S \overset{*}{\to} x$ is less than or equal to k derivations. So, we have x as a balanced sequence of parentheses.

  Obviously, (x) has an equal number of right parenthesis and left parenthesis.
  Since the left parenthesis is added in the front of x and for x and every prefix of S, for every prefix of (x), the number of left parenthesis is also greater than or equal to that of right parenthesis.
  $\Big($the equal happens when the prefix is the whole word $(x)\Big)$

  2 $S \to SS \overset{*}{\to} x_1x_2 = w$ in this case, $S \overset{*}{\to} x_1$ is less than or equal to k derivations. So is $S \overset{*}{\to} x_2$. we have x_1, x_2 as a balanced sequence of parentheses.

  Since x_1, x_2 has an equal number of right and left parenthesis, $w = x_1x_2$ also has an equal number of right and left parenthesis.

  $w = x_1x_2$, if we look from left to right until x_1, according x_1 is properly balance string, we know for this part, every prefix the number of left parenthesis is greater than or equal to that of right parenthesis.
  Now we consider the case prefix of w is of the form x_1x_2' where x_2' is the prefix of x_2. For every prefix of x_2 we have $\#(\geq \#)$, and x_1 has equal number of left parenthesis and right parenthesis. So, for every prefix of $w = x_1x_2$, we still have $\#(\geq \#)$

  So, w is a properly balanced sequence of parentheses.

Now, we can say any string any string generated by this grammar has [1] equal number of left parenthesis and right parenthesis and [2] in every prefix of the sequence the number of left parentheses is greater than or equal to the number of right parentheses, which is a properly balanced sequence of parentheses

(b):

We now need to prove that every sequence of balanced parentheses can be generated by this grammar.

Let w be the string of balanced parentheses. We can use induction on the length of w to show w can be generated by the given grammar.

Base Case: Suppose that $w = \varepsilon$. Since the grammar has rule $B \to \varepsilon$, we can see it generates w.

Induction Step:

Induction Hypothesis: Suppose that every sequence of balanced parentheses whose length is less than n can be generated by this grammar.

Now we assume $|w| = n$, and consider the least $k > 1$ such that the prefix of w of length k has the same number of (as).

- If $k = n$, then we write w as $w = (w_1)$, now we need to show w_1 is a string of balanced parentheses.
  - As there are same number (as) in $w = (w_1)$, so, for w_1, there are same number of (and) as well.
  - Suppose that some prefix of w_1 has more) than (, then there exists some w_1' has exactly one more) than (, so k is not the least, as the prefix (w_1' has the same number of ('s and)'s and has length less than k. So, for every prefix of w_1, $\#(\geq \#)$

  So, we can now say w_1 is a string of balanced parentheses.
  According to our hypothesis, we can say w_1 can be generated by CFG.
  Using the rule $S \to (S)$, $w = (w_1)$ can also be generated by CTG

- If $k < n$, then we write w as $w = (w_1)w_2$, where $|(w_1)| = k$ now we need to show w_1, w_2 is a string of balanced parentheses.
  - according to previous case, we have w_1 has the same number of (and), and for every prefix of w_1, $\#(\geq \#)$, which means w_1 is a string of balanced parentheses.
  - As there are as manny ('s and)'s in both w and (w_1), it is the same in w_2
  - For every prefix w_2' of w_2, $(w_1)w_2'$ is also a prefix of w, as we know for every prefix of w, $\#(\geq \#)$ and for (w_1), the number of (is the same as), so for every prefix w_2' of w_2, we still have $\#(\geq \#)$

  So, we can conclude that w_1, w_2 is a string of balanced parentheses. According to our hypothesis, we can say w_1, w_2 can be generated by CFG.
  Using the rule $S \to (S)$, (w_1) can also be generated by CTG
  Using the rule $S \to SS$, $w = (w_1)w_2$ can also be generated by CTG

Thus, we can conclude that every sequence of balanced parentheses can be generated by this grammar.

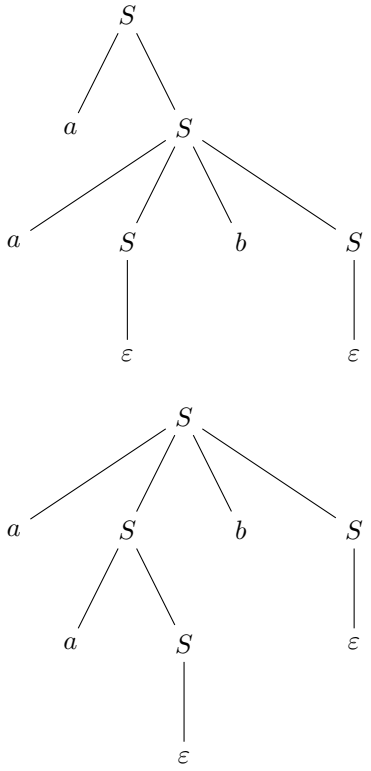
Question 2:

Consider the following context-free grammar:

$$S \to aS \mid aSbS \mid \varepsilon$$

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.

I choose string aab



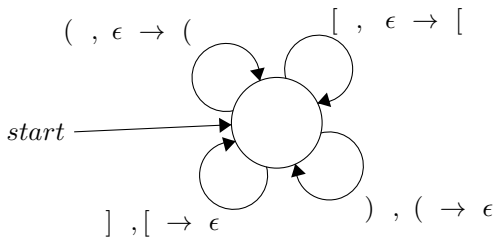
Question 3:

We define the language $PAREN_2$ inductively as follows:

- $\varepsilon \in PAREN_2$
- if $x \in PAREN_2$ then so are (x) and $[x]$
- 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:



As this question asks for a PDA for this language which accepts by empty stack, we do not need accept states any more. It will accept the word if there is no input anymore and the stack is empty.

It pushes open parentheses and brackets into the stack and pops them off if it sees the matched closed parentheses and brackets.

All the other transition which is not on the graph will lead the PDA to jam

Question 4:

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:
To begin with, I will give the grammar first.

$$S \to M|AN|SC$$

$M \to aMc|B$ M builds equal numbers of a's and c's

$N \to bNc|\varepsilon$ N builds equal numbers of b's and c's

$A \to aA|\varepsilon$ this builds a list of a's

$B \to bB|\varepsilon$ this builds a list of b's

$C \to cC|\varepsilon$ this builds a list of c's

Then, I will show every world in the language $L = \{a^n\ b^m\ c^p \mid n \leq p\ or\ m \leq p\}$ can be generated by the grammar.
As we can see, if the world $w \in L$, the number of c must be greater than the number of a or the number of c, which means $p > min(n, m)$

- Suppose that $n \leq m$, then we can write $w\ as\ w = a^n\ b^m\ c^n\ c^{p-n}$, we write $w = w_1 \cdot w_2 = (a^n\ b^m\ c^n) \cdot c^{p-n}$
  Obviously, we can generate $w_1 = a^n\ b^m\ c^n$ using rule $M \to aMc|B$ combined with rule $B \to bB|\varepsilon$.
  Then, by rule $S \to M$, we have w_1 can be generated by the grammar.
  We can generate $w_2 = c^{p-n}$ using rule $C \to cC|\varepsilon$.
  Finally, use the rule $S \to SC$, we can get $w = w_1 \cdot w_2$ can be generated by the grammar.

- Suppose that $n > m$, then we can write $w\ as\ w = a^n\ b^m\ c^m\ c^{p-m}$, we write $w = w_1 \cdot w_2 = (a^n\ b^m\ c^m) \cdot c^{p-m}$
  Obviously, we can generate $b^m\ c^m$ using rule $N \to bNc|\varepsilon$,
  we can generate a^n using rule $A \to aA|\varepsilon$.
  Then, by rule $S \to AN$, we have $w_1 = (a^n\ b^m\ c^m)$ can be generated by the grammar.
  We can generate $w_2 = c^{p-n}$ using rule $C \to cC|\varepsilon$.
  Finally, use the rule $S \to SC$, we can get $w = w_1 \cdot w_2$ can be generated by the grammar.

Then, I will show that, every string generated by the grammar is in $L = \{a^n\ b^m\ c^p \mid n \leq p\ or\ m \leq p\}$

As we can see, M generates the same number of a's and c's, so, the rule $S \to M$ always gives $n = p$, if it is combined with rule $S \to SC$, we can have $n \leq P$
And, N generates the same number of b's and c's, so, the rule $S \to AN$ always gives $m = p$, if it combines with the rule $S \to SC$, we can have $m \leq P$
Thus, we can say that every string generated by the grammar is in $L = \{a^n\ b^m\ c^p \mid n \leq p\ or\ m \leq p\}$

Question 5:

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

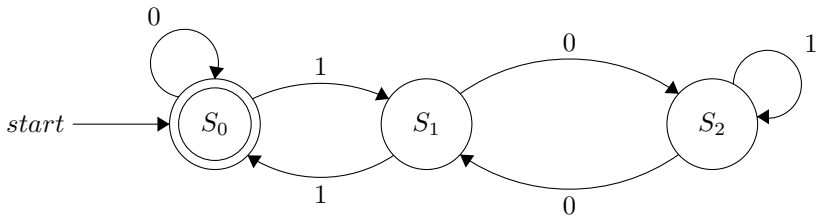
$$S \to aS \mid a \mid bB; \qquad B \to bB \mid b.$$

- Prove 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.
- 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.

(a):

- To begin with, we use all the **alphabet of the DFA** as the **Terminal of our CFG**:
- Then, we use all the **states of the DFA** as the **non-Terminal of CFG**:
- Then we need to choose a start symbol $S \in V$, the **start symbol** is the same as the **start state in DFA**:
- Now, for each non-Terminal (each states of the DFA), we have a transition function in this form $\delta(S_i, a) = S_j$, then we can convert the transition function into CFG rules $S_i \to aS_j$:
- Now, we need to end our grammar. For those $\delta(S_i, a) = S_j$,
  if S_j is the accept state, we need to add one more rule $S_j \to \varepsilon$
  if the start state S_0 is also accept state, we need to add $S_0 \to \varepsilon$

I used a concrete DFA to make my way to construct the grammar more clear.



- To begin with, we use all the **alphabet of the DFA** as the **Terminal of our CFG**:
  so, the terminal is $T = \{0, 1\}$
- Then, we use all the **states of the DFA** as the **non-Terminal of CFG**:
  $V = \{S_0, S_1, S_2\}$
- Then we need to choose a start symbol $S \in V$, the **start symbol** is the same as the **start state in DFA**:
  $S = \{S_0\}$
- Now, for each non-Terminal (each states of the DFA), we have a transition function in this form $\delta(S_i, a) = S_j$, then we can convert the transition function into CFG rules $S_i \to aS_j$: In our case, we have:

$$S_0 \to 0S_0 \mid 1S_1$$
$$S_1 \to 0S_2 \mid 1S_0$$
$$S_2 \to 1S_2 \mid 0S_1$$

- Now, we need to end our grammar. For those $\delta(S_i, a) = S_j$,
  if S_j is the accept state, we need to add one more rule $S_j \to \varepsilon$
  if the start state S_0 is also accept state, we need to add $S_0 \to \varepsilon$

$$S_0 \to 0S_0 \mid 1S_1 \mid \varepsilon$$
$$S_1 \to 0S_2 \mid 1S_0$$
$$S_2 \to 1S_2 \mid 0S_1$$

(b):

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

As we can see, this grammar can generate string in the form $a^n\ b^n$, which we know is non-regular.