COMP 330 Assignment 4 - 03/10/21

Denis [redacted]

Q1a)

Consider the following grammar: S--> (S) | SS | ε

We will analyse the productions to prove this grammar produces only balanced parenthesis. To achieve this 2 properties must be met:

a)An equal number of opening and closing parentheses

b)In every prefix there must be more or equal number of opening parentheses than closing parentheses

Clearly property 1) is maintained at all times. The first substitution is the only one that adds parentheses to the string and it does so b adding an opening and closing parenthesis surrounding a non-terminal symbol. This addition also maintains property 2) as the number of opening parentheses is always equal to the number of right parentheses. Thus the first rule manitains balance.

The second rule subsititutes in 2 non-terminal symbols without introducing any parentheses. Each of these can only introduce parentheses by means of rule 1) which we proved to comply with our requirements.

The last rule is epsilon which is by definition balanced.

Therefore since all rules of this grammar comply with the 2 requirements for balanced parentheses we can say that any string produced by this grammar has balanced parentheses.

Q1b)

Proof by induction on the set of parentheses pairs.

Base case: n=0 -> ε which is balanced

Inductive assumption: assume the grammar produces all possible combinations of n pairs of balanced parentheses strings.

We need to prove it can generate n+1 pairs of balanced parentheses strings.

Let w be a string with n+1 pairs of balanced parenthesis. We will prove that this is achieved from a compliant string with n pairs of parentheses. 2 possible scenarios arise:

a)Adding a nested pair of parentheses inside a group of parenthesis in a string of n-pairs by using the "(S)" rule on a non-terminal symbol in a string of n pairs instead of ε (i.e. this means we add a nested pair of parentheses thus increasing the pair count by 1).

Let w = ((...ε...)) be a string with n pairs of balanced parentheses. Let w' = ((...(S)...)) where we have taken w and replace an inner ε symbol rule replacement by rule 1) instead. Clearly w' has n+1 balanced pairs of parentheses.

b)Using rule 2) "SS" and then rule 1) "(S)" on one or both non-terminal symbols to append a pair of parentheses symbols.

Let w = SS = w'w'' s.t. w' = (...) is a string with n pairs of balanced parentheses and w'' = S = ε. Thus clearly w is covered by our inductive assumption. Now instead of having w'' = ε replace it with rule 1) instead. We have then w'' = (S) = (ε) = (). Now we have the following string:

w = SS = w'w'' where w' has n pairs of balanced parentheses and w'' has 1 pair of balanced parentheses. Clearly w has n+1 balanced parentheses now.

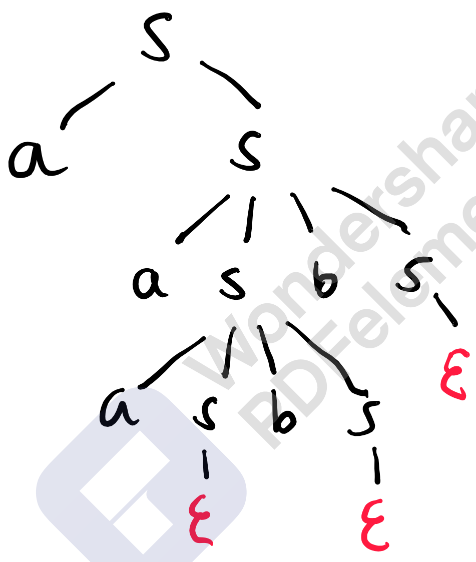
By induction we have proved we can derive strings with n+1 pairs of balanced parentheses from strings with n pairs.

Q2)

$aS \mid aSbS \mid \varepsilon$

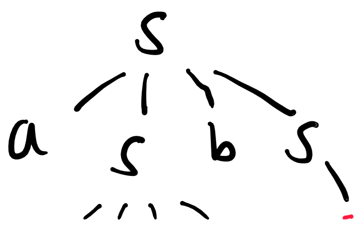
aaabb

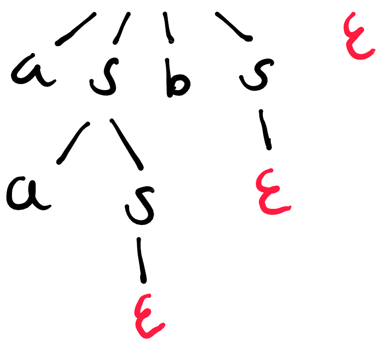
Version 1



$\Rightarrow$ aaabb

Version 2



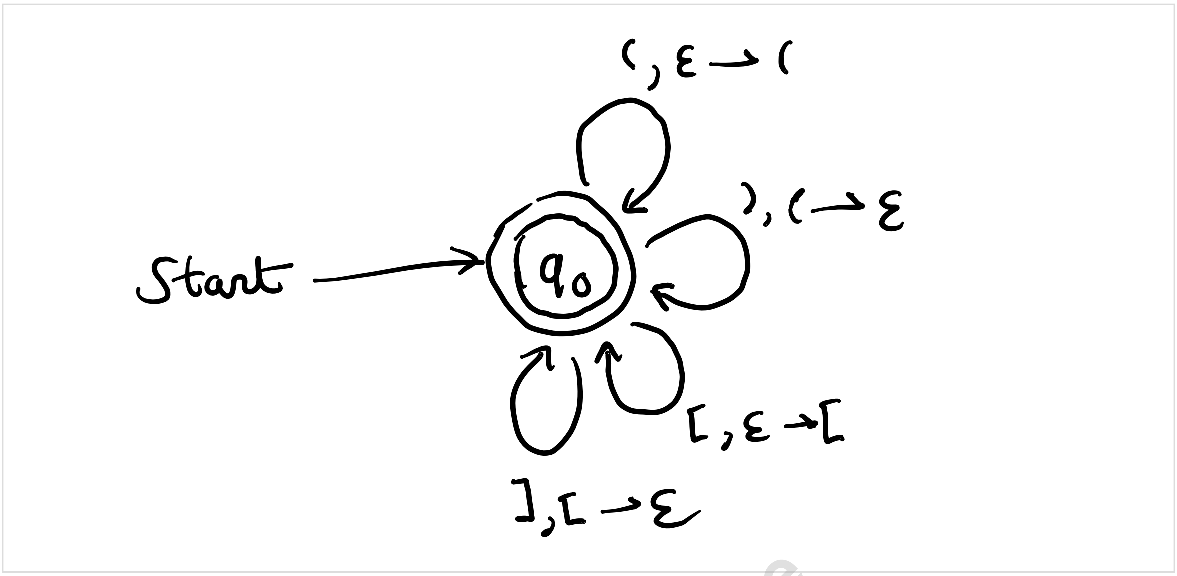


$\Rightarrow aaabb$

We have 2 parsing that are different for the same string.

Thus, the grammar is ambiguous.

Q3)



The language accepts strings with proper parenthesis structure
It also accepts an empty string.

Opening parenthesis and brackets are pushed. Closing symbols must be matched with an opening symbol of the same type.

If we try to close with a now. match then the machine jams and rejects.

If we stop reading input and we haven't reached the bottom of the stack we jam and reject

Q4)

$$L = \{a^n b^m c^p \mid n \leq p \text{ or } m \leq p\}$$

$$\Sigma = \{a, b, c\}$$

$$V = \{S, A, A_2, B, B_2, C\}$$

$$S \to aAc \mid B_2 b Bc \mid C \mid \varepsilon$$

$$A \to aAc \mid Ac \mid A_2 b \mid \varepsilon$$

$$A_2 \to A_2 b \mid \varepsilon$$

$$B \to bBc \mid Bc \mid \varepsilon$$

$$B_2 \to aB_2 \mid \varepsilon$$

$$C \to cC \mid \varepsilon$$

S is the starting symbol and allows for 4 replacements, 2 of them deciding which of a and b c will be the upper bound of.

1)"a A c" Means c is a's upper bound. The non-terminal symbol A allows for 4 substitutions:

1.1)"a A c" is a recursive substitution to keep #a's = #c's

1.2)"A c" increases the c's without increasing the a's

1.3)"A2" b once we are done adjusting the a's and c's we can move to adding b's. We use a new non-terminal symbol called A2. A2 only has 2 substitutions: its recursive reference and epsilon. This allows us to add as many b's as we wish.

1.4)"c C" allows for a string of only arbitrarily long chain of c's

1.5)Epsilon or end of subsitution chain

2)"B2 b B c" Means we chose to make c be the upper bound of the b's. a's can grow arbitrarily large. The non-terminal symbols are B2 and B:

2.1)"B2" which has 2 substitutions: its recursive reference to add as many a's as we wish and epsilon.

2.2)"B" which has 3 substitutions:

2.2.1)"b B c" is a recursive substitution that increases b's and c's by 1

2.2.2)"B c" increases the c's without increasing the b's

2.2.3)Epsilon or end of substitution chain

Note that in any way we always enforce the a's, the b's or both to be bounded by the number of c's. Also note that with the placement of non-terminal symbols we never have out of order letters.

Q5a) We will create a DFA to showcase that any regular language can be generated from a right-linear grammar.

Let L define a right linear grammar. L = (V, T, S, P)

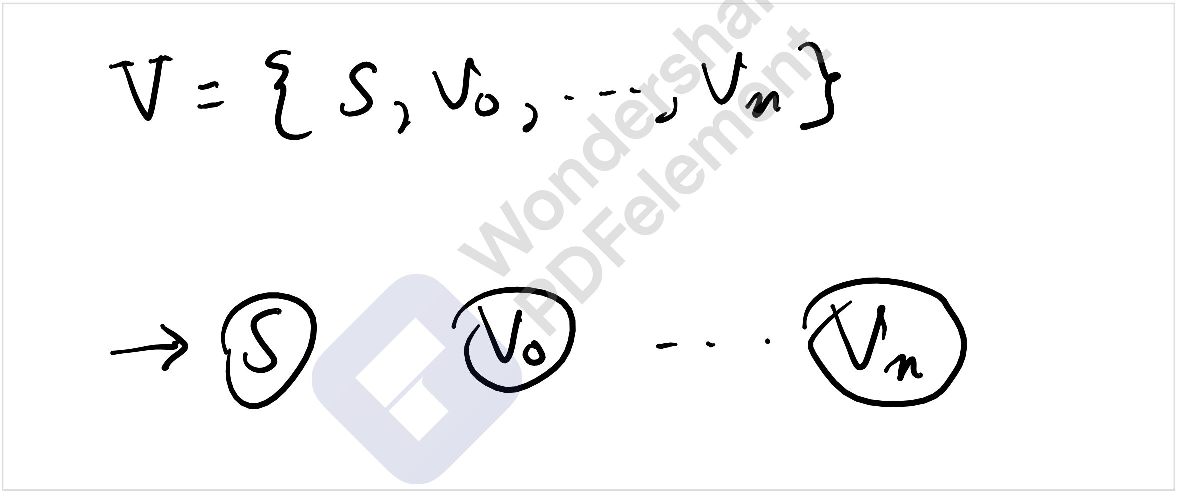
Let M be a DFA. M=(Q, Σ, δ, q0, f)

Clearly Σ=T.

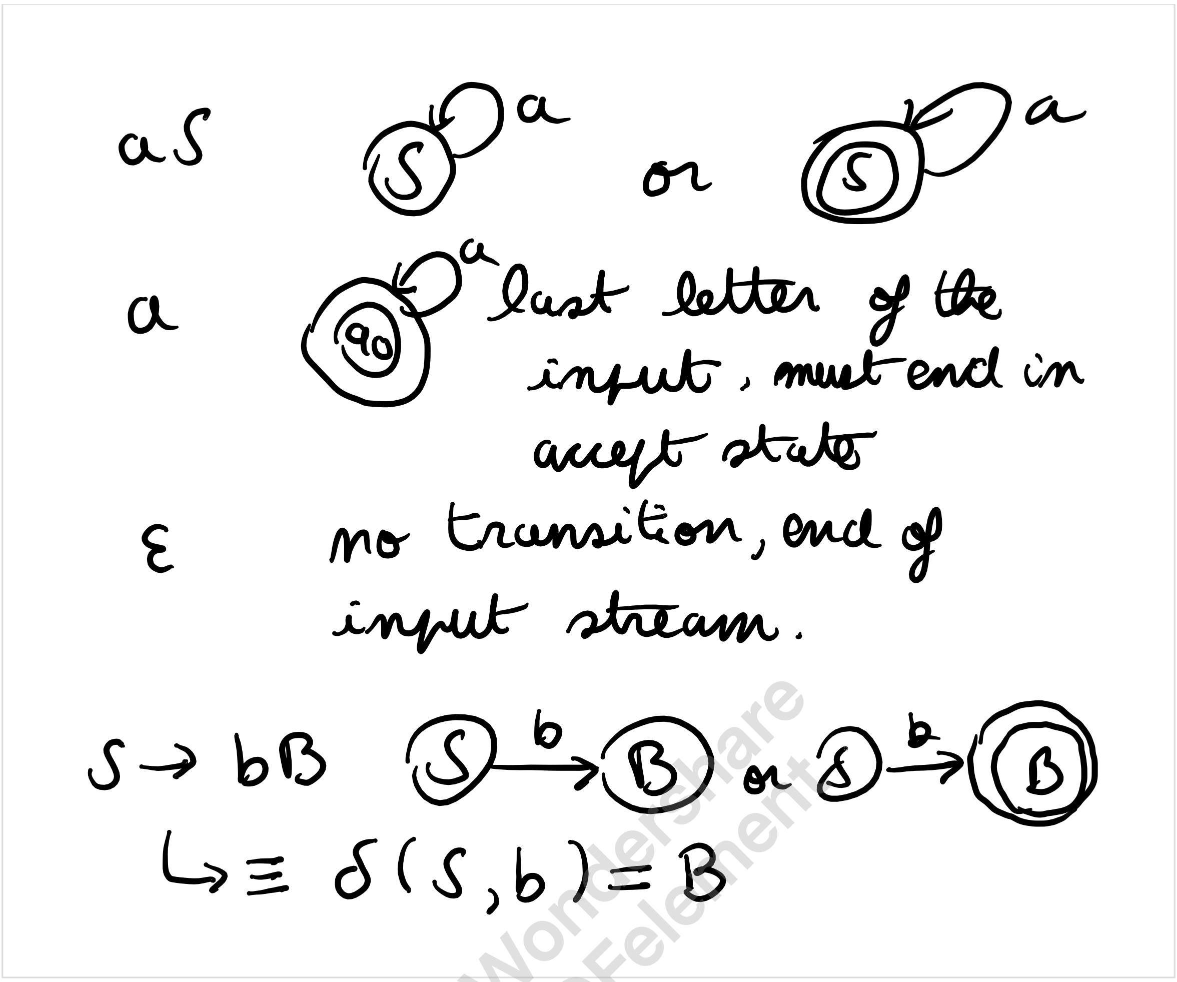
A right linear grammar has 2 types of rules:

1) "a S"
2) "a" or "ε"

Create one state per non-terminal symbol in V. Therefore Q = V, and S is in q0.



Each rule per non-terminal symbol is a transition. In case of terminal only symbol rules it must lead to an accept state. Thus f is a subset of V and isn't empty.



aS or

a last letter of the input, must end in accept state

ε no transition, end of input stream.

$S \to bB$ or

$\hookrightarrow \equiv \delta(S, b) = B$

Therefore we have proved we can wirte a DFA for every right-linear grammar.

Q5b) False, consider the following grammar:

S--> aB | ε

B--> Sb

Clearly it generates the language a^n b^n which is known to be non-regular.