**LAB2**

**Mingxuan Zhang**

**Henok Hailemariam**

**Team: BTS**

**2:**

**(a) Describe the language defined by the following grammar:**

**S ::= A B A**

**A** **::= a | a A**

**B** **::= ε | b B c | B B**

S, A, B are Non-Terminals, and a, b, c, εare Terminals.

A can be {a, aa, aaa, aaaa, ……..}

B can be { ε, bc, bcbc, bcbcbc, ……}

S is made by ABA

**(b) Consider the following grammar:**

**S ::= A a B b**

**A ::= A b | b**

**B ::= a B | a**

**Which of the following sentences are in the language generated by this grammar? For**

**the sentences that are described by this grammar, demonstrate that they are by giving**

**derivations.**

**1. baab**

**2. bbbab**

**3. bbaaaaa**

**4. bbaab**

1 is described by this grammar because:

baab => S => A a B b => b a B b (A = B ) => b a a b (B = a) .

4 is described by this grammar because:

bbaab => S =>A a B b =>A b a B b (A=Ab) => b b a B b (A=b)=> b b a a b (B=a).

**(c) Consider the following grammar:**

**S ::= a S c B | A | b**

**A ::= c A | c**

**B ::= d | A**

**Which of the following sentences are in the language generated by this grammar? For**

**the sentences that are described by this grammar, demonstrate that they are by giving**

**parse trees.**

**1. abcd**

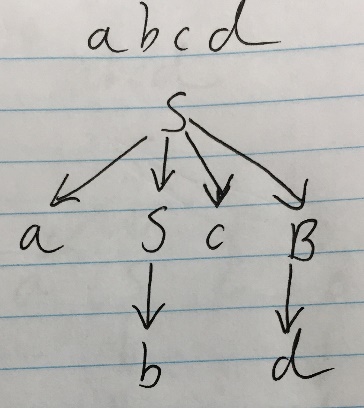
**2. acccbd**

**3. acccbcc**

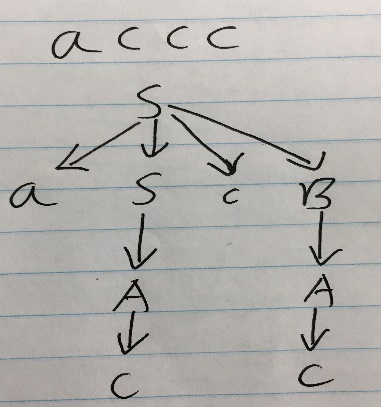
**4. acd**

**5. accc**

1. abcd is described by this grammar



5. accc is described by this grammar



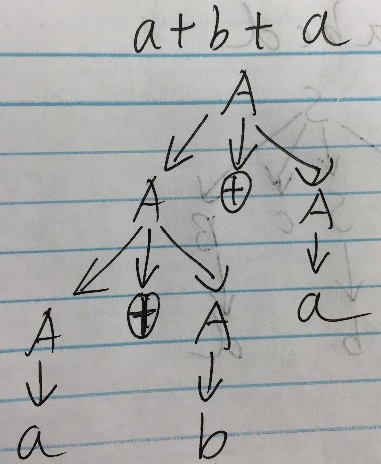
**(d) Consider the following grammar:**

**A ::= a | b | A⊕A**

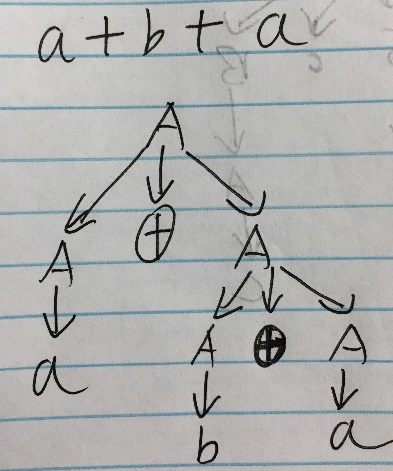
**Show that this grammar is ambiguous.**

It is ambiguous, because, when we have string like this : a + b + a , we can have two parse trees.

First parse tree:



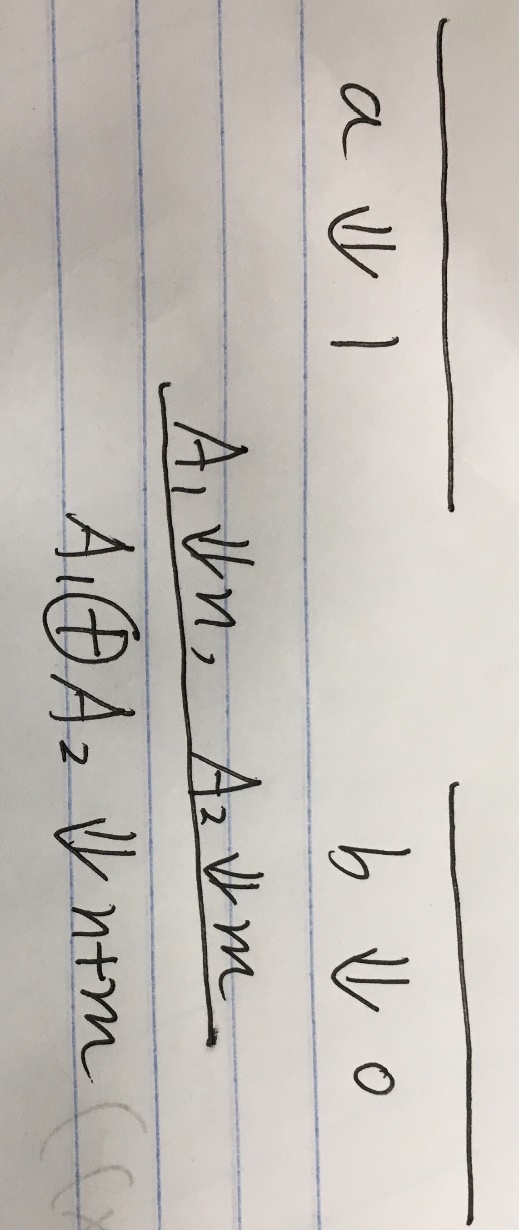
Second parse tree;



**(e)** **Let us ascribe a semantics to the syntactic objects A specified in the above grammar from part d. In particular, let us write**

**A ⇓ n**

**for the judgment form that should mean A has a total n a symbols where n is the metavariable for natural numbers. Define this judgment form via a set of inference rules. You may rely upon arithmetic operators over natural numbers. Hint: There should be one inference rule for each production of the non-terminal A (called a syntax-directed judgment form).**

****

**3. Grammars: Understanding a Language.**

**(a) Consider the following two grammars for expressions e. In both grammars, operator**

**and operand are the same; you do not need to know their productions for this question.**

**e ::= operand | e operator operand**

**e ::= operand esuffix**

**esuffix ::= operator operand esuffix | ε**

**i. Intuitively describe the expressions generated by the two grammars.**

**ii. Do these grammars generate the same or different expressions? Explain.**

(i) :

The first grammar is left associative, it can generate long string by adding operator operand to the left recursively.

The second grammar is right associative, it can generate long string by adding operator operand to the right recursively.

(ii):

Yes, these two grammars will generate same expressions.

Because:

The type of expression that the grammar generates is:

1. operand
2. operand operator operand operator operand………………

Well the type of expression that the grammar generates is also:

(1) operand

(2) operand operator operand operator operand………………

**(b) Write a Scala expression to determine if ‘−’ has higher**

**precedence than ‘<<’ or vice versa. Make sure that you are**

**checking for precedence in your expression and not for left or**

**right associativity. Use parentheses to indicate the possible**

**abstract syntax trees, and then show the evaluation of the**

**possible expressions. Finally, explain how you arrived at the**

**relative precedence of ‘−’ and ‘<<’ based on the output that**

**you saw in the Scala interpreter.**

val test = 4 << 1 – 1

val r1 = (4 << 1) – 1

val r2 = 4 << (1 - 1)

if (test == r1)

println (“<< has higher precedence”)

if (test == r2)

println(“ - has higher precedence”)

**(c) Give a BNF grammar for floating point numbers that are made**

**up of a fraction (e.g., 5.6 or 3.123 or -2.5) followed by an optional**

**exponent (e.g., E10 or E-10). The exponent, if it exists, is the letter ‘E’**

**followed by an integer. For example, the following are floating point**

**numbers: 3.5E3, 3.123E30, -2.5E2, -2.5E-2, and 3.5. The following are**

**not examples of floating point numbers: 3.E3, E3, and 3.0E4.5. More**

**precisely, our floating point numbers must have a decimal point, do**

**not have leading zeros, can have any number of trailing zeros, non**

**zero exponents (if it exists), must have non-zero fraction to have an**

**exponent, and cannot have a ‘-’ in front of a zero number. The**

**exponent cannot have leading zeros. For this exercise, let us assume**

**that the tokens are characters in the following alphabet**

**Σ: Σ def = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,E, -,.}**

**Your grammar should be completely defined (i.e., it should not count**

**on a non-terminal that it does not itself define).**

S := -A|A

A := B.C|B.CEB|B.CE-B

B := DC|D

C := Z|CZ

D := 1|2|3|4|5|6|7|8|9