CSCI 3155, Spring 2018 Lab2 WriteUp
Adam Ten Hoeve (01/28) adam.tenhoeve@colorado.edu

---

2 **Grammars: Synthetic Examples.**

(a) Describe the language defined by the following grammars.

$$S ::= ABA$$
$$A ::= a|aA$$
$$B ::= \epsilon|bBc|BB$$

Any string in the language begins with the S non-terminal character, which can only become ABA. A can only become a*n, for some number n, the any string will begin and end with some number of a's. B only has one terminal state, ϵ, so if that is not selected then it could make equal amounts of b's and c's. Some example stings:

`aa` `aabca` `aabcbcaa` `abbccbcaa`

(b) Consider the following grammar:

$$S ::= AaBb$$
$$A ::= Ab|b$$
$$B ::= aB|a$$

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

Strings 1 and 4 can be generated from this grammar, strings 2 and 3 can not.
String 1: $S- > AaBb- > (b)a(a)b- > baab$
String 4: $S- > AaBb- > (Ab)a(a)b- > (b)baab- > bbaab$

(c) Consider the following grammer:

$$S ::= aScB|A|b$$
$$A ::= cA|c$$
$$B ::= d|A$$

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

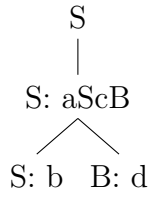
2 acccbd

3 acccbcc

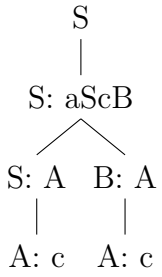
4 acd

5 accc

Strings 1 and 5 can be generated from the given grammar, strings 2, 3, and 4 can not be.
Parse Tree for String 1:



Resulting in string *abcd*.
Parse tree for string 5:



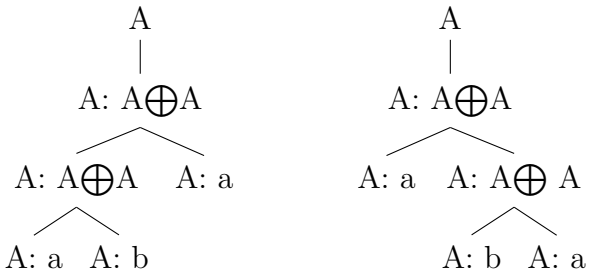
Resulting in string *accc*.

(d) Consider the following grammar:

$$A ::= a|b|A \bigoplus A$$

Show that this grammar is ambiguous.

A grammar is ambiguous if there are multiple parse trees for the same string. Take the string $a{+}b{+}a$. This string can be formed multiple ways meaning the grammar is ambiguous.



Both of these trees result in the string a⊕b⊕a, so the grammar is ambiguous.

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

$$\text{A} \downarrow \text{n}$$

for the judgment form that should mean A has a total n a symbols where n is the meta-variable 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).

Axioms:

$$\frac{}{a \downarrow 1} \qquad \frac{}{b \downarrow 0}$$

These axioms are for the terminal cases, meaning that an a string has a correlating $\downarrow n = 1$, which is from our rule set, and that b has a correlating $\downarrow n = 0$. From here we can make an inference rule for the non-terminal case such as:

$$\frac{A_1 \downarrow n \quad A_2 \downarrow m}{A_1 \bigoplus A_2 \downarrow n + m}$$

From this inference rule and our axioms, we have defined the judgment for and can find the $\downarrow n$ of any string from our grammar.

3 (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.

$$\text{e ::= operand} \mid \text{e operator operand}$$

$$\text{e ::= operand esuffix}$$
$$\text{esuffix ::= operator operand esuffix} \mid \epsilon$$

i. Intuitively describe the expressions generated from the two grammars.

The first grammar with either be an operand or a string of operators and operands with the tree expanding to the left. The very leftmost and rightmost values will always be operands, while flipping between operator and operand for each entry.
The second grammar will also either be a single operand or a string of operators and operands. This parse tree for this grammar will extend to the right as the non-terminal value is on the right side. Again, the leftmost and rightmost values will always be operand while flipping between operator and operand for each entry.

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

These two grammars will produce the same string even though the parse trees will look different. Both grammars have the base case of a single operand even though one is terminal and the other is not. Then at each iteration of the grammar, we would be adding a single *operator* and *operand* strings, but to opposite sides in opposite order. Because both of these are flipped, the final expressions are again always the same.

(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.

In Node.js, we calculated the following.

$$
\begin{aligned}
&> 3 - 2 << 1 \\
&2 \\
&> 3 << 2 - 1 \\
&6 \\
&> (3 - 2) << 1 \\
&2 \\
&> 3 - (2 << 1) \\
&-1
\end{aligned}
$$

From these we can see that 3 - 2 << 1 and (3 - 2) << 1 returned the same values, whereas the other combinations did not. This means that '-' has greater precedence than '<<' because the parenthesis did not affect the

outcome.

(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:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, E, -, .\}$$

Your grammar should be completely defined (i.e., it should not count on a non-terminal that it does not itself define).

$S ::= NegNum.Dec \mid NegNum.DecENegNum$ ('.' is decimal point and 'E' is the exponent)
$Neg ::= - \mid \epsilon$ (ϵ is empty character)
$Num ::= D \mid NumZ$
$Dec ::= Z \mid ZDec$
$D ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
$Z ::= 0 \mid D$