CSCI 3155 LAB 2 WRITEUP

Jason Lubrano & Lucas Sward

February 11, 2018

1 Feedback

2 Grammars: Synthetic Examples

Problem a). Describe the language defined by the following grammar:

$$\begin{aligned} S &::= ABA \\ A &::= a|aA \\ B &::= \varepsilon|bBc|BB \end{aligned}$$

Solution:
The language defined by this grammar contains sentences that begin and end with any number of a's. There can be sentences that contain no b's or c's because the epsilon terminates a B but there has to be at least two a's in a sentence. When there exist b's and c's in the sentence, there must be an equal number of b's to c's.

Problem b). Consider the grammar:

$$\begin{aligned} S &::= AaBb \\ A &::= Ab|b \\ B &::= aB|a \end{aligned}$$

Which of the following sentes are in the language generated by this grammar? For the sentences that are, demonstrate by **derivations**

Problem 1 baab. baab is in the language generated by the grammar

$$\textbf{baab} : S \rightarrow AaBb \rightarrow baBb \rightarrow baab$$

Problem 2 bbbab. bbbab is not in the language generated by the grammar. Due to the language, we would need a second b after the last a.

Problem 3 bbaaaaa. bbaaaaa is not in the language described by the grammar. Due to the language, we would have to finish the sentence with a b.

Problem 4 bbaab.

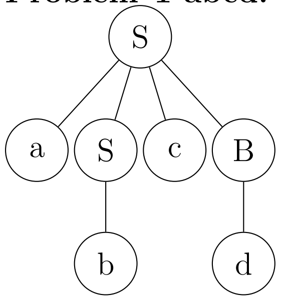
$$\textbf{bbaab} : S \rightarrow AaBb \rightarrow AbaBb \rightarrow bbaBb \rightarrow bbaab$$

Problem c). Consider the following grammar:

$$\begin{aligned} S &::= aScB|A|b \\ A &::= cA|c \\ B &::= d|A \end{aligned}$$

Which of the following sentences are in the language generated by this grammar? For the sentences that are desribed by this grammar, demonstrate that they are by giving **parse trees**

Problem 1 abcd. abcd

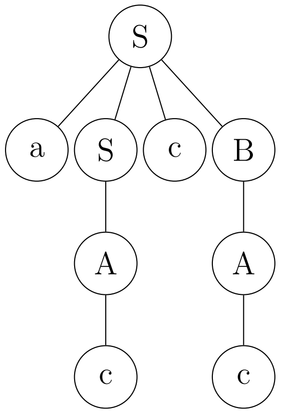


Problem 2 acccbd. acccbd is not in the language described by this grammar. Due to the grammar, for a sentence to begin in a, we can end in d, so our sentence works for the first two letters; however, we are unable to input a b between the c and d.

Problem 3 acccbcc. acccbcc is not in the language described by this grammar. Since our sentence begins with an a, our sentence must end with either a c or a d, so this sentence fits so far, but the problem arises when we have *ccc*. We are either able to input a single b or *ccc* but not both.

Problem 4 acd. acd is not in the language described by this grammar. Since our sentence begins with a, it must have at least 4 letters in the sentence. Since this sentence only has 3 letters, it is impossible to include it in our language.

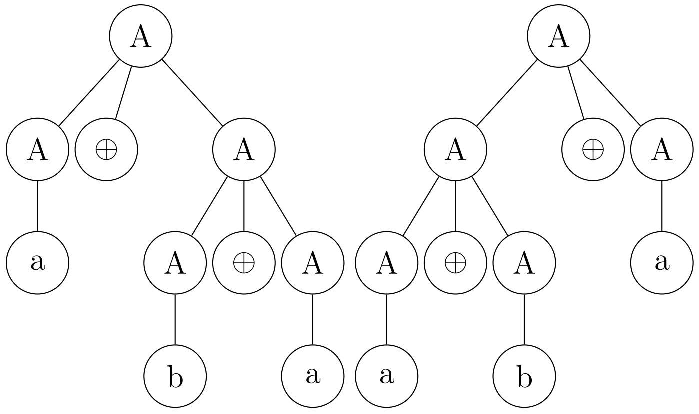
Problem 5 accc. accc is in the language:



Problem d). Consider the grammar:

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

Show it is ambiguous: This is ambiguous because for some sentences there exist multiple parse trees. Ex: $a \oplus b \oplus a$



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

$$A \Downarrow n$$

for the judgment form that should mean A has a total n symbols where n is the meta-variable for natural numbers. Define this judgment form via a set of inference rules.
Solution:
Two Base Cases:

$$\frac{}{a \Downarrow 1}$$

$$\frac{}{b \Downarrow 0}$$

For any sentence, let x be the string left of the ⊕ and right of it be y. Then:

$$\frac{x \Downarrow n, y \Downarrow m}{x \oplus y \Downarrow n + m}$$

3 Grammars: Understanding a Language

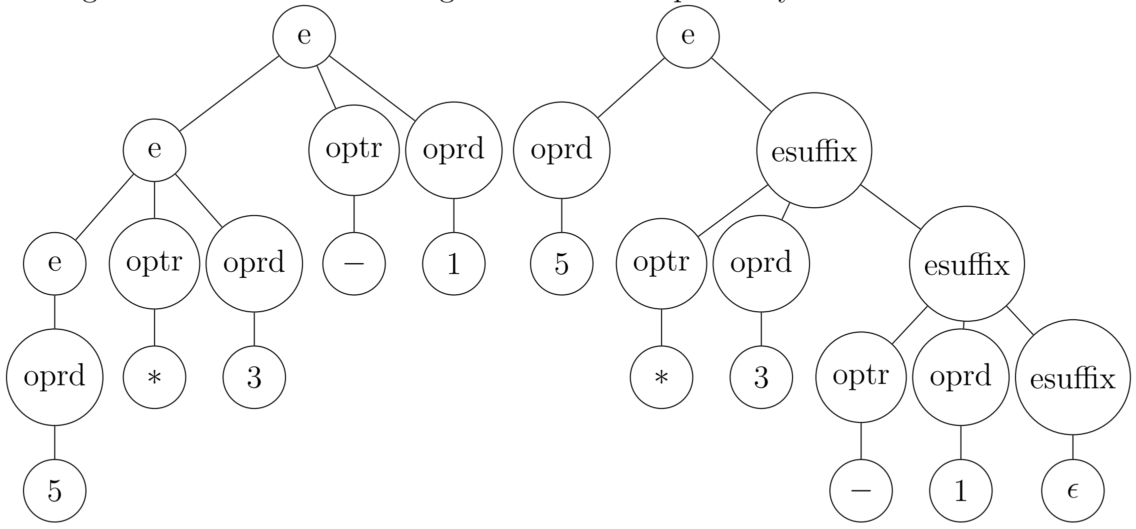
Problem a). Consider the following two grammars for expressions e. In both grammars, *operator* and *operand* are the same; you do not need to know their products for this question.

$$e ::= \textit{operand} | e \ \textit{operator operand}$$

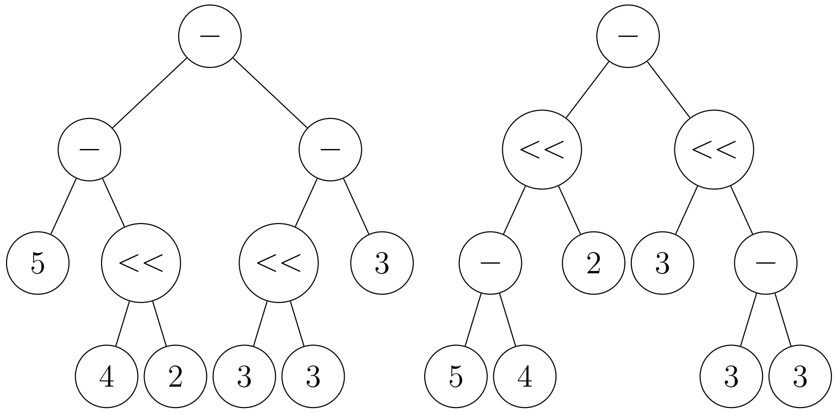
$$e ::= \textit{operand esuffix}$$
$$\textit{esuffix} ::= \textit{operator operand esuffix} | \varepsilon$$

I. *Intuitively describe the expressions generated by the two grammars.*
Solution:
Each grammar generates expressions that, at the minimum, contain just one operand. The two grammars allow for chains of operands and operators where each operator must have an operand on either side of it, every operand must be followed by an empty string or

another operator. There cannot be two operands next to each other, and there cannot be two operators next to each other. Every expression must begin and end with an operand, and there will always be one less operator than operands.
II. *Do these grammars generate the same or different expressions? Explain.*
Solution:
Although these grammars have the same rules for what expressions can be created, they do not generate the same expressions. The first grammar generates expressions that give precedence to operators from left to right while the second grammar generates expressions that give precedence to operators from right to left. This would mean that each grammar would assign a different precedence to the same operators in identical expressions.
Ex. $5 * 3 - 1$: 5, 3, and 1 are operands and $*$ and $-$ are operators:
First grammar tree and second grammar tree respectively:



Problem 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.
Solution:
Expression: $(5 - 4 << 2) - (3 << 3 - 3)$
Possible precedence: $(5 - (4 << 2)) - ((3 << 3) - 3)$ or $((5 - 4) << 2) - (3 << (3 - 3))$:
Possible trees:



From the scala output, it is clear that '$-$' has precedence over '<<'.
$(5-4 << 2)-(3 << 3-3)$ evaluates to 1 which means scala did $((5-4) << 2)-(3 << (3-3))$ which is $4-3$ which is 1. If << had precedence over $-$ then the evaluation of $(5-4 << 2)-(3 << 3-3)$ would be -32. The precedence of the $-$ operator did not depend on left or right associativity because the two numbers being subtracted changed left and right associativity in the two parts of the expression.

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

$$\begin{aligned}
S &::= GDP \\
P &::= \epsilon | EL | EGL \\
G &::= \epsilon | - \\
D &::= LN.N | L.N \\
N &::= 0 | L | NN \\
L &::= 1|2|3|4|5|6|7|8|9|LL
\end{aligned}$$