CSCI 3155: Lab 2 (due February 9)

Ian Smith & Vy Le

1. **Grammars: Synthetic Examples**

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

$$
\begin{aligned}
S &::= A\ B\ A \\
A &::= \texttt{a} \mid \texttt{a}\ A \\
B &::= \epsilon \mid \texttt{b}\ B\ \texttt{c} \mid B\ B
\end{aligned}
$$

   The sentences in this language begin with one or more a's, followed by 0 or more b's and c's, where the number of b's equals the number of c's and, further, this substring of b's and c's acts like a string of balanced parentheses with each "opening" b having a paired "closing" c to somewhere to the right of it in the string.

   (b) Consider the following grammar:

$$
\begin{aligned}
S &::= A\ \texttt{a}\ B\ \texttt{b} \\
A &::= A\ \texttt{b} \mid \texttt{b} \\
B &::= \texttt{a}\ B \mid \texttt{a}
\end{aligned}
$$

   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.

   i. `baab` - is in this language

$$
\begin{aligned}
S &\Rightarrow A\ \texttt{a}\ B\ \texttt{b} \\
&\Rightarrow \texttt{b}\ \texttt{a}\ B\ \texttt{b} \\
&\Rightarrow \texttt{b}\ \texttt{a}\ \texttt{a}\ \texttt{b}
\end{aligned}
$$

   ii. `bbbab` - **not** in this language because B will produce at least one `a` and as there is already a terminal `a` in first step derivation of start symbol S, there must be at least 2 `a`'s in the sentences described by this grammar.

   iii. `bbaaaaa` - **not** in this language because all sentences in this language must end in a `b` since there is a terminal `b` in any derivation from start symbol S.

iv. `bbaab` - is in this language

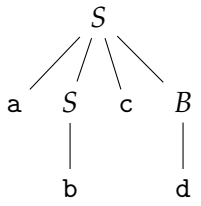
$$\begin{aligned} S &\Rightarrow A\,\mathtt{a}\,B\,\mathtt{b} \\ &\Rightarrow A\,\mathtt{b}\,\mathtt{a}\,B\,\mathtt{b} \\ &\Rightarrow \mathtt{b}\,\mathtt{b}\,\mathtt{a}\,B\,\mathtt{b} \\ &\Rightarrow \mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\mathtt{a}\,\mathtt{b} \end{aligned}$$

(c) Consider the following grammar:

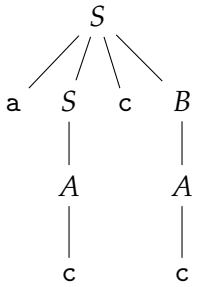
$$\begin{aligned} S &::= A\,S\,\mathtt{c}\,B \mid A \mid \mathtt{b} \\ A &::= \mathtt{c}\,A \mid \mathtt{c} \\ B &::= \mathtt{d} \mid A \end{aligned}$$

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` - is in the language



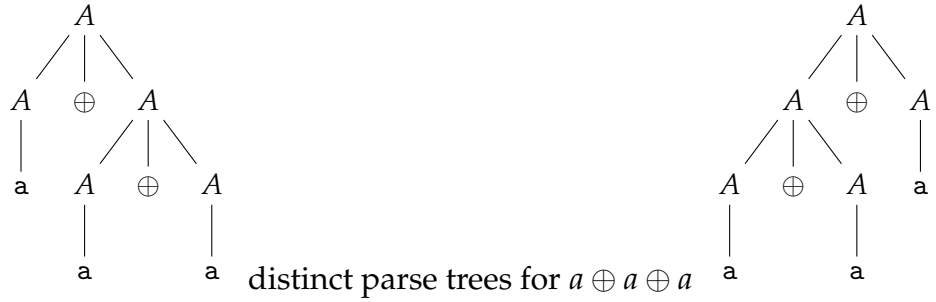
2. `acccbd` - **not** in the language because S may produce $ASc B$, A or `b`. A can only produce one or more `c`'s. `b` is terminal and the B derived from S can only generate a substring of one or more `c`'s or a single `d` neither of which will give an end of `bd`.
3. `acccbd` - **not** in the language because there is no way to produce an internal `b` without producing another `a` and there is only one `a` in `acccbd`.
4. `acd` - **not** in the language because there is no production in which S can go to an empty string, which would be necesarry to produce just the 3 terminal symbols.
5. `accc` - is in the language



(d) Consider the following grammar:

$$A ::= \texttt{a} \mid \texttt{b} \mid A \oplus A$$

Show that this grammar is ambiguous.
I show that this grammar is ambiguous by giving two distinct parse trees for the sentence, a⊕a⊕a:



distinct parse trees for $a \oplus a \oplus a$

(e) Let us write

$$A \Downarrow n$$

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).
We begin with two base case judgments that follow directly from our definition, these are axioms:

$$\frac{}{\texttt{a} \Downarrow 1}\,(1) \quad \text{and} \quad \frac{}{\texttt{b} \Downarrow 0}\,(2)$$

From these two rules, we can construct our third inference rule, for the third production of the non-terminal A.
Given a sentence of the form $x \oplus y$, where x and y are productions of A, and each x and y have n and m a symbols, respectively, we can write our last rule:

$$\frac{x \Downarrow n \quad y \Downarrow m}{x \oplus y \Downarrow n + m}\,(3)$$

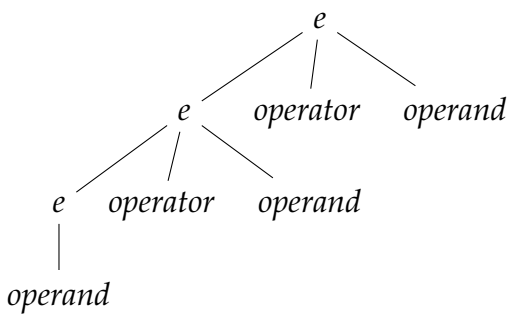
2. **Grammars: Understanding a Language**

(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.

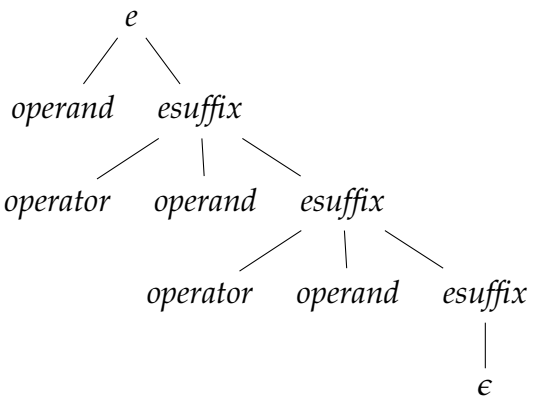
$$e ::= \textit{operand} \mid e\ \textit{operator operand}$$

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

For this first grammar, the smallest expression that can be generated is a single "*operand*". This first grammar can also generate expressions that begin with an *operand* and are followed by "*operator operand*" for any number of repetitions. In this first grammar, expressions grow leftward because the recursive non-terminal symbol *e* is to the left of operator operand and recursive productions are put to the left of "*operator operand*".
The second grammar, also has a smallest possible generated expression of a single ''*operand*". It can generate expressions of the form "*operator operand operator operand ... operand*" (just like the first grammar). However, the expressions in this grammar grow rightward as the recursive symbol *esuffix* is to the right of "*operator operand*" in the non-terminal production of the recursive definition of *esuffix*.
These grammars can generate the same expressions but with different parse trees. The first grammar is left associative and the second grammar is right associative. This is exemplified by the following two parse trees for an example expression in both grammars:



First Grammar, left associative



Second Grammar, right associative

(b) Write a Scala expression to determine if '-' has higher precedence than '<<' or vice versa.
I use the expression `4−2 << 4−3` to determine which has higher precedence.
Here are the possible parse trees:

`((4 - 2)<<(4 - 3))` $\rightarrow^*$ 4 (for $-$ has higher precedence)

`((4 -(2<<4))-3)` $\rightarrow^*$ -31 (for$<<$ higher precedence, $-$ left associative)

`(4 - ((2<<4)-3))` $\rightarrow^*$ -25 (for $<<$ higher precedence, $-$ right associative)

`(((4 - 2)<<4)-3)` $\rightarrow^*$ 29 (for equal precedence, left associative)

`(4 - (2<<(4-3)))` $\rightarrow^*$ 0 (for equal precedence, right associative)

When put in the Scala interpretor, we get the output value of 4. Thus, the first parse tree must be the correct one and $-$ **has higher precedence than** $<<$**.**

(c) Give a BNF grammar for floating point numbers.
The following grammar defines floating point numbers according to these rules.

$S ::= \Sigma\,\delta\,.\,\Omega \mid 0\,.\,Z$

$\Omega ::= \Delta\,E\,\Sigma\,\delta \mid \Delta$ (1 + digits and exponent | 1+ digits)

$\Sigma ::= n \mid -n$ (+|− a non-zero digit)

$\Delta ::= n\,\delta \mid Z\,\delta$ (1+ digits)

$\delta ::= n\,\delta \mid Z\,\delta \mid \epsilon$ (0+ digits)

$n ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$Z ::= 0 \mid 0\,Z$ (1+ zero digits)

Numbers **in** this grammar: 3.0 , 1.0E4 , 0.0 , 0.00 , 123.0 , 123.123E-34
Numbers **not in** this grammar: 0. , 1. , 3.E3 , E3 , 3.0E4.5