COMP 141: Ambiguity, EBNFs and Parsers
Instructions: In this exercise, we are going to review EBNFs and parsers.

1 Ambiguous Grammars

Consider the following grammar with terminals: `number`, $+$, $*$, (, and).

$$\begin{aligned}
&\mathit{expr} ::= \mathit{expr} + \mathit{expr} \mid \mathit{expr} * \mathit{expr} \mid (\mathit{expr}) \mid \texttt{number} \\
&\texttt{number} = \texttt{[0-9]+}
\end{aligned}$$

1. As you have seen in the slides, we can disambiguate the grammar by revising it as follows

$$\begin{aligned}
&\mathit{expr} ::= \mathit{expr} + \mathit{term} \mid \mathit{term} \\
&\mathit{term} ::= \mathit{term} * \mathit{factor} \mid \mathit{factor} \\
&\mathit{factor} ::= (\mathit{expr}) \mid \texttt{number} \\
&\texttt{number} = \texttt{[0-9]+}
\end{aligned}$$

   (a) Since the grammar is disambiguated, there is only one parse tree now for each expression that can be derived from the grammar. What is the unique parse tree for deriving $78 * 20 + 5 * 39$. Let's call this parse tree $\mathbb{P}$.
   (b) Give the corresponding AST for $\mathbb{P}$. Let's call it $\mathbb{A}$.
   (c) What would be the final value if you pass $\mathbb{A}$ to an evaluator (in an interpreter)?

2 EBNFs

1. As discussed in the class, given the BNF grammar

$$\begin{aligned}
&\mathit{expr} ::= \mathit{expr} + \mathit{term} \mid \mathit{term} \\
&\mathit{term} ::= \mathit{term} * \mathit{factor} \mid \mathit{factor} \\
&\mathit{factor} ::= (\mathit{expr}) \mid \textbf{\textit{number}} \\
&\mathit{number} ::= \texttt{NUMBER} \\
&\texttt{NUMBER} = [0-9]+
\end{aligned}$$

   we can rewrite it in EBNF as follows.

$$\begin{aligned}
&\mathit{expr} ::= \mathit{term}\ \{+\ \mathit{term}\} \\
&\mathit{term} ::= \mathit{factor}\ \{*\ \mathit{factor}\} \\
&\mathit{factor} ::= (\mathit{expr}) \mid \textbf{\textit{number}} \\
&\mathit{number} ::= \texttt{NUMBER} \\
&\texttt{NUMBER} = [0-9]+
\end{aligned}$$

   Let's call this definition of grammar g_1.

   (a) What is the derivation for $78 * 20 + 5 * 39$ using the rules in g_1?
   (b) What is the corresponding parse tree?
   (c) What is the corresponding AST?

2. Moreover, we can rewrite the BNF grammar

$$\begin{aligned}
&\mathit{expr} ::= \mathit{term} + \mathit{expr} \mid \mathit{term} \\
&\mathit{term} ::= \mathit{factor} * \mathit{term} \mid \mathit{factor} \\
&\mathit{factor} ::= (\mathit{expr}) \mid \textbf{\textit{number}}
\end{aligned}$$

as the EBNF grammar:

$$\begin{aligned} expr &::= term\ [+\ expr] \\ term &::= factor\ [*\ term] \\ factor &::= (expr) \mid \textit{number} \end{aligned}$$

Let's call this definition of grammar g_2.

(a) What is the derivation for $78 * 20 + 5 * 39$ using the rules in g_2?
(b) What is the corresponding parse tree?
(c) What is the corresponding AST?

3 Recursive-descent parser

1. Give the pseudo-code for recursive-descent parser that implements g_1.
2. Give the pseudo-code for recursive-descent parser that implements g_2.

4 Boolean expressions

Consider the following CFG with the eight terminals: `true`, `false`, $\wedge$, $\vee$, `!`, `==`, `(`, and `)`.

$$expr ::= \texttt{true} \mid \texttt{false} \mid expr \wedge expr \mid expr \vee expr \mid \texttt{!}\, expr \mid expr == expr \mid \texttt{(}expr\texttt{)}$$

Indeed, the starting symbol is $expr$. Let's call this grammar G. This grammar is ambiguous, i.e., there exist at least two parse trees for some expression. For example, in a previous lab you were able to give two different syntax trees for the following expression:

$$\texttt{!true} \wedge \texttt{false} \vee \texttt{true == true}$$

1. Let's disambiguate G. We want to impose the following precedence cascade among operators:
   - The highest precedence is for parentheses,
   - the second highest precedence is for `!`,
   - the third highest precedence is for $\wedge$,
   - the fourth highest precedence is for $\vee$, and finally
   - the least precedence is for ==.

   In addition, all binary operators are left-recursive. Define the disambiguated version of the grammar in BNF. Let's call your disambiguated grammar G'.
2. In your defined G', is operator $\wedge$ left-associative or right-associative? What about operator $\vee$?
3. Using G', give the derivation for the expression below:

   $$\texttt{!true} \wedge \texttt{false} \vee \texttt{true == true}$$

   Note that since G' is disambiguated, there must be a unique parse tree for this expression.
4. Give the corresponding parse tree for the derivation in the previous question.
5. Give the corresponding AST for the parse tree in the previous question.
6. If you pass this AST to an evaluator, what would be the final result?
7. Redefine G' using EBNF. Let's call this version of grammar G''.
8. Give the pseudo-code for the recurisve-descent parser that implements G''. The parser needs to generate the AST (so it is not a recognizer!).