Context-Free Grammar Exercises
CS 364 — Spring 2022

1 Definitions and Background

1. Define the following terms and give examples where appropriate.
   (a) Context-Free Grammar:

   (b) Non-terminal:

   (c) Terminal:

   (d) Language of a Context-Free Grammar:

   (e) Derivation:

   (f) Parse Tree:

   (g) Ambiguous Grammar:

   (h) Left-Recursive Grammar:

2 Context-Free Grammars

1. Let L_1 be the language consisting of all non-empty palindromes over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. That is, L_1 consists of all sequences of a's and b's that read the same forward or backward. For example, `aba` $\in L_1$ and and `bb` $\in L_1$ and `aabbbaa` $\in L_1$, but `abb` $\notin L_1$.

   Let L_2 be the language over $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ denoted by the regular expression `a(a|b)*`.

   (a) Write a context-free grammar for L_1.

   (b) Draw a parse tree for the string `ababa` using your grammar from the previous part.

   (c) The language $L_3 = L_1 \cap L_2$ is context-free. A string s is in L_3 if $s \in L_1$ and $s \in L_2$. Write a context-free grammar for the language L_3.

2. Consider the following grammar:

$$\begin{aligned} S &\rightarrow aSb \\ S &\rightarrow Sb \\ S &\rightarrow \varepsilon \end{aligned}$$

   (a) Give a one-sentence description of the language generated by this grammar.

   (b) Show that this grammar is ambiguous by giving a single string that can be parsed in two different ways. Draw both parse trees.

   (c) Give an unambiguous grammar that accepts the same language as the grammar above.

3. Using the context-free grammar for snail given in the snail specification, draw a parse tree for the following expression.

```
while ( !(x = z = 0) ) {
    y = z + 2 + x * 1;
}
```

Note that the context-free grammar by itself is ambiguous, so you will need to refer to the precedence and associativity rules to get the correct tree.

4. Consider the following grammar:

$$\begin{array}{lcl} E & \rightarrow & E \texttt{ + } T \\ E & \rightarrow & T \\ T & \rightarrow & T \texttt{ * } F \\ T & \rightarrow & F \\ F & \rightarrow & \texttt{(} \ E \ \texttt{)} \\ F & \rightarrow & \texttt{x} \\ F & \rightarrow & \texttt{y} \end{array}$$

(a) Why couldn't a recursive descent parser use this grammar to recognize token sequences?

(b) Rewrite the grammar in such a way that a recursive descent parser could successfully use the resulting grammar.