After this tutorial you should be able to:

1. Convert an English or mathematical description of a context-free language into a CFG, and argue why your grammar is correct.
2. Describe in English or mathematics the language of a CFG, and argue why your description is correct.
3. Argue if a CFG is ambiguous or not.

Problem 1. Let $\Sigma = \{0, 1\}$.

1. Describe in one sentence the language generated by the following grammar:

$$\begin{aligned} S &\to X1X \\ X &\to 0X \mid \epsilon \end{aligned}$$

2. Describe in one sentence the language generated by the following grammar:

$$S \to 0S0 \mid 1$$

Solution 1.

1. The set of binary strings of the form 0^n10^m for $n, m \geq 0$ (i.e., the language is $L(0^*10^*)$).
2. The set of binary strings of the form 0^n10^n for $n \geq 0$.

Note that although X stores the information "any number of zero", the two occurrence of X in $S \to X1X$ are independent of each other.

Note that 0100 is derivable by the first grammar, but not by the second grammar.

Problem 2. Consider the following grammar:

$$\begin{aligned} E &\to E + T \mid E - T \mid T \\ T &\to F \times T \mid T/T \mid F \\ F &\to (E) \mid V \mid C \\ V &\to a \mid b \mid c \\ C &\to 1 \mid 2 \mid 3 \end{aligned}$$

1. Indicate the set of variables, terminals, production rules, and the start variable.
2. Give a left-most derivation of the string $a + b \times c$
3. Give a right-most derivation of the string $a + b \times c$
4. Give a parse tree for $a \times b - 2 \times c$

5. Give a parse tree for $a \times (b - 2 \times c)$

Solution 2.

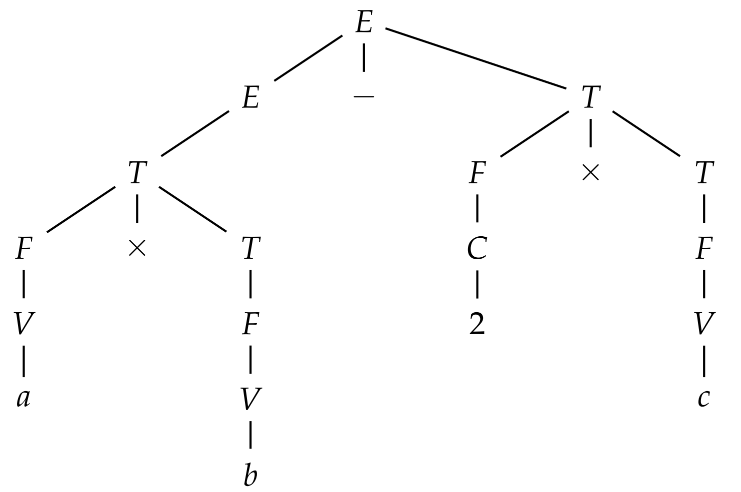
1. $V = \{E, T, F, V, C\}$
   $T = \{a, b, c, 1, 2, 3, (,), \times, /, +, -\}$
   Production rules as above (note there are 15 rules!)
   Start variable as E (if it's not stated we assume it's the first one, or S)

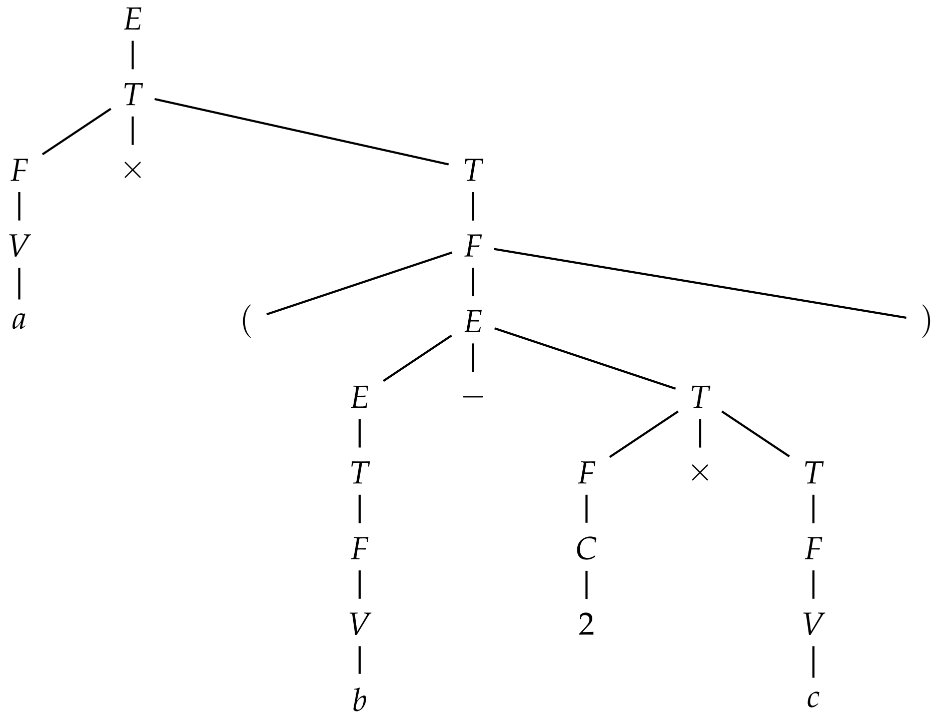
2.

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow V + T \\ &\Rightarrow a + T \\ &\Rightarrow a + F \times T \\ &\Rightarrow a + V \times T \\ &\Rightarrow a + b \times T \\ &\Rightarrow a + b \times F \\ &\Rightarrow a + b \times V \\ &\Rightarrow a + b \times c \end{aligned}$$

3.

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + F \times T \\ &\Rightarrow E + F \times F \\ &\Rightarrow E + F \times V \\ &\Rightarrow E + F \times c \\ &\Rightarrow E + V \times c \\ &\Rightarrow E + b \times c \\ &\Rightarrow T + b \times c \\ &\Rightarrow F + b \times c \\ &\Rightarrow V + b \times c \\ &\Rightarrow a + b \times c \end{aligned}$$

4. 

5. 

Problem 3.

1. Write a recursive definition of the set of strings of balanced parentheses. So, e.g., the following strings are in the language:

   - ()(())
   - ()
   - ϵ

   and the following are not:

   -)()
   - (()))

2. Write a CFG for the language of balanced parentheses.
3. If you haven't already, write an unambiguous CFG for the language of balanced parentheses. Can you explain why your grammar is unambiguous?

Solution 3. Here is a recursive definition:

1. The empty string is balanced.
2. If w is balanced then so is (w).
3. If v, w are balanced then so is vw.

Here is the corresponding CFG: $S \to SS \mid (S) \mid \varepsilon$.

This grammar is ambiguous since every leftmost derivation can be prefixed by $S \Rightarrow SS \to S$ (using the $S \to \varepsilon$ rule).

An unambiguous grammar that generates the same language is $S \to \epsilon | (S)S$.

Problem 4.

1. Show that every regular language is context-free. You may want to give a recursive translation of regular expressions to CFGs.
2. Show that there is some context-free language that is not regular.

Solution 4. We give a recursive transformation of a RE R into a CFG. Let's call this RETOCFG:

1. If $R = \emptyset$ then return the grammar $S \to S$.
2. If $R = \epsilon$ then return the grammar $S \to \epsilon$.
3. If $R = a$ for $a \in \Sigma$ then return the grammar $S \to a$.
4. If $R = (R_1 R_2)$ then let $G_1 = \text{RETOCFG}(R_1)$ and $G_2 = \text{RETOCFG}(R_2)$. Suppose that the start state of G_i is S_i. Return the grammar consisting of the rules of G_1, the rules of G_2, and the additional rule $S \to S_1 S_2$, where S is a new state.
5. If $R = (R_1 \cup R_2)$ then let $G_1 = \text{RETOCFG}(R_1)$ and $G_2 = \text{RETOCFG}(R_2)$. Suppose that the start state of G_i is S_i. Return the grammar consisting of the rules of G_1, the rules of G_2, and the additional rule $S \to S_1 | S_2$ where S is a new state.
6. If $R = (R_1)^*$ then let $G_1 = \text{RETOCFG}(R_1)$. Suppose the start state of G_1 is S_1. Return the grammar consisting of the rules of G_1 and the new rule $S \to S_1 S \mid \epsilon$.

For the second part, note that $S \to 0S1|\epsilon$ generates the non-regular language $\{0^n1^n : n \geq 0\}$.

Problem 5. Some programming languages define the `if` statement in similar ways to the following grammar:

$$\begin{aligned} \textit{Conditional} &\to \text{ if } \textit{Condition} \text{ then } \textit{Statement} \\ \textit{Conditional} &\to \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ else } \textit{Statement} \\ \textit{Statement} &\to \textit{Conditional} \mid S_1 \mid S_2 \\ \textit{Condition} &\to C_1 \mid C_2 \end{aligned}$$

1. Show that this grammar is ambiguous.
2. Show that the string you provided can be interpreted in two different ways, i.e., resulting in programs with different behaviours.
3. Write a CFG that captures `if` statements but is not ambiguous.

Solution 5.

1. The string "if C1 then if C2 then S1 else S2" has two leftmost derivations.
   (a) Here is one derivation (not drawing $\Rightarrow$):
      i. *Conditional*
      ii. if *Condition* then *Statement*
      iii. if C1 then *Statement*
      iv. if C1 then *Conditional*
      v. if C1 then if *Condition* then *Statement* else *Statement*
      vi. if C1 then if C2 then *Statement* else *Statement*
      vii. if C1 then if C2 then S1 else *Statement*
      viii. if C1 then if C2 then S1 else S2
   (b) Here is another:
      i. *Conditional*
      ii. if *Condition* then *Statement* else *Statement*
      iii. if C1 then *Statement* else *Statement*
      iv. if C1 then *Conditional* else *Statement*
      v. if C1 then if *Condition* then *Statement* else *Statement*
      vi. if C1 then if C2 then *Statement* else *Statement*
      vii. if C1 then if C2 then S1 else *Statement*
      viii. if C1 then if C2 then S1 else S2
2. If $C1$ is true and C_2 is false, then one of the programs does S_2, and the other doesn't do S_2.

3.

$$\begin{aligned}
\textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ endif} \\
\textit{Conditional} &\rightarrow \text{ if } \textit{Condition} \text{ then } \textit{Statement} \text{ else } \textit{Statement} \text{ endif}
\end{aligned}$$

Problem 6. Describe the language generated by the following context-free grammar:

$$\begin{aligned}
S &\rightarrow X1Y \\
X &\rightarrow \epsilon \mid X0 \\
Y &\rightarrow \epsilon \mid 1Y \mid Y0
\end{aligned}$$

Briefly explain why your answer is correct.

Is the grammar ambiguous?

Solution 6. Language of the regexp $0^*11^*0^*$.

Informally, the reason is that S generates $X1Y$; X generates 0^*; Y generates 1^*0^*.

More precisely, we will show two things: that every string matching the regexp $0^*11^*0^*$. can be generated by this grammar; and that every leftmost derivation generates a string matching this grammar.

Every string of the form $0^n11^m0^l$ can be generated as follows: $S \Rightarrow X1Y \Rightarrow^n 0^n1Y \Rightarrow^m 0^n11^mY \Rightarrow^l 0^n11^m0^l$. On the other hand, a leftmost derivation must look as follows $S \Rightarrow X1Y \Rightarrow^n 0^n1Y$ for some $n \geq 0$, and $Y \Rightarrow^* 1^m0^l$ for some $m, l \geq 0$.

It is ambiguous because, e.g., $S \Rightarrow X1Y \Rightarrow 1Y \Rightarrow 11Y \Rightarrow 11Y \Rightarrow 11Y0 \Rightarrow 110$ and $S \Rightarrow X1Y \Rightarrow 1Y \Rightarrow 1Y0 \Rightarrow 11Y0 \Rightarrow 110$ are two leftmost derivations of the string 110.