The following CFG handles mathematical expressions:

\[ G_4 = (V, \Sigma, R, \langle expr \rangle) \]

- \( V = \{\langle expr \rangle, \langle term \rangle, \langle factor \rangle\} \)
- \( \Sigma = \{a, +, \times, (, )\} \)
- \( R: \)
  
  \[
  \begin{align*}
  \langle expr \rangle & \rightarrow \langle expr \rangle + \langle term \rangle \mid \langle term \rangle \\
  \langle term \rangle & \rightarrow \langle term \rangle \times \langle factor \rangle \mid \langle factor \rangle \\
  \langle factor \rangle & \rightarrow (\langle expr \rangle) \mid a
  \end{align*}
  \]

Fig 2.5: check \( a + a \times a \)
Example 2.4 II

$$\begin{align*}
\text{check } (a + a) \times a
\end{align*}$$
Example 2.4 III

\[(E \rightarrow T) \times (a \rightarrow T \rightarrow F) \rightarrow (E \rightarrow F \rightarrow E) \times (a \rightarrow F \rightarrow (a + a)).\]
Design Grammars I

\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}

\begin{align*}
S_1 \rightarrow & \ 0S_11 \mid \epsilon \\
S_2 \rightarrow & \ 1S_20 \mid \epsilon \\
S \rightarrow & \ S_1 \mid S_2
\end{align*}

If CFG regular \(\Rightarrow\) by DFA

\begin{align*}
R_i \rightarrow & \ aR_j \text{ if } \delta(q_i, a) = q_j \\
R_i \rightarrow & \ \epsilon \text{ if } q_i \in F
\end{align*}

We see the main difference between CFG and DFA
Design Grammars II

CFG: a rule can be like

\[ R_i \rightarrow aR_j b \]

DFA: a rule can only be

\[ R_i \rightarrow aR_j, \]

where we treat each \( R_i \) as a state and let

\[ \delta(R_i, a) = R_j \]
Ambiguity I

- The same string but obtained in different ways
- For the example of mathematical expressions discussed earlier, what if we consider the following rules?

\[
\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \\
\langle \text{expr} \rangle \times \langle \text{expr} \rangle \mid (\langle \text{expr} \rangle) \mid a
\]

We see the following ways to parse

\[a + a \times a\]
This CFG does not give the precedence relation. We want that $a \times a$ is done first.
By the more complicated CFG earlier, the parsing is unambiguous.

Question: how to formally define the ambiguity?

We need to define “leftmost derivation” first.
Leftmost derivation

Even for an unambiguous CFG we may have the same parse tree, but different derivations. For the CFG discussed earlier we can do

\[
\langle \text{expr} \rangle \Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \\
\Rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle \times \langle \text{factor} \rangle
\]

where the second part is expanded. On the other hand, we can handle the first one first.

\[
\langle \text{expr} \rangle \Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \\
\Rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle
\]
This is not considered ambiguous

Definition of leftmost derivation: at every step the leftmost remaining variable is the one replaced.
Formal definition of ambiguity I

- $w$ ambiguous if there exist two leftmost derivations
- Some context-free languages can be generated by ambiguous & unambiguous grammars
- We say a CFG is inherently ambiguous if it only has ambiguous grammars

See prob 2.29 in the textbook. Details not given here.