Context-Free Grammars: Ambiguity, Associativity and Precedence

- **Context-Free, or Type 2, Grammars, are used to formally define the syntax of programming languages. They are also known as BNF, or Backus-Naur Form.**
- **Two grammars are *equivalent* if they generate the same language.**

- A derivation in a context-free grammar can be represented by a *parse tree*.
  - the root is the start symbol
  - the interior nodes are non-terminals
  - the leaves in left-to-right order form a sentential form
  - X1, X2, ..., Xm are the children of A only if A -> X1X2...Xm is a production

- **A grammar is *ambiguous* if it generates a string with two distinct parse trees.**

  Example:

$$\begin{array}{rcl} < expr > & ::= & x|y|z|(< expr >) \\ & | & < expr > + < expr > \\ & | & < expr > * < expr > \end{array}$$

  Ambiguous: $x + y + z$

  Rule: A grammar is ambiguous if it is both left and right recursive.

  Fix: remove right recursion

$$\begin{array}{rcl} < expr > & ::= & < element > \\ & | & < expr > + < element > \\ & | & < expr > * < element > \\ < element > & ::= & x|y|z|(< expr >) \end{array}$$

Example: "Dangling Else" Problem

$$
\begin{array}{lll}
\textbf{S} & ::= & \textbf{if } E \textbf{ then } S \\
& | & \textbf{if } E \textbf{ then } S \textbf{ else } S \\
& | & \textbf{other}
\end{array}
$$

Ambiguous: if E then if E then S else S

Fix:

$$
\begin{array}{lcl}
S & \rightarrow & S1 \mid S2 \\
S1 & \rightarrow & \textbf{if } E \textbf{ then } S1 \textbf{ else } S1 \\
S1 & \rightarrow & \textbf{other} \\
S2 & \rightarrow & \textbf{if } E \textbf{ then } S \\
S2 & \rightarrow & \textbf{if } E \textbf{ then } S1 \textbf{ else } S2
\end{array}
$$

Associativity

- **Productions that are left-recursive force evaluation in left-to-right order (left associativity).**
- **Productions that are right-recursive force evaluation in right-to-left order (right associativity).**
- **Example: This left-recursive grammar enforces left-associativity:**

$$
\begin{array}{lcl}
< expr > & ::= & < element > \\
 & | & < expr > + < element > \\
 & | & < expr > * < element > \\
< element > & ::= & x|y|z|(< expr >)
\end{array}
$$

  - **Parse $x + y + z$ again.**
  - **Parse $x{+}y{*}z$ and $x{*}y{+}z$; note that left-associativity is maintained, but + and * have the same precedence.**

Precedence

- **Precedence is introduced by adding new non-terminal symbols.**
- **Cascade symbols with lowest precedence closest to the start symbol.**
- **Example**

$$
\begin{array}{lcl}
< expr > & ::= & < term > \\
 & | & < expr > + < term > \\
< term > & ::= & < element > \\
 & | & < term > * < element > \\
< element > & ::= & x|y|z|(< expr >)
\end{array}
$$

Now Parse $x + y * z$ and $x * y + z$; note that left-associativity and the correct precedence are enforced.

- **Example**
- **Precedence: (), - (unary), ↑, */, +-**
- **Associativity:**
  left: */+-
  right: ↑

$$
\begin{array}{lcl}
< expr > & ::= & < term > \\
 & | & < expr > + < term > \\
 & | & < expr > - < term > \\
< term > & ::= & < factor > \\
 & | & < term > * < factor > \\
 & | & < term > / < factor > \\
< factor > & ::= & < primary > \uparrow < factor > \\
 & | & < primary > \\
< primary > & ::= & - < primary > \\
 & | & < element > \\
< element > & ::= & x|y|z|(< expr >)
\end{array}
$$