COMP2212 Programming language concepts

Lecture 2 - Syntax, BNF, Parse trees
Syntax vs semantics

- **syntax**: the rules for writing programs
  - not following the rules results in compile-time errors (eg. syntax error)
- **semantics**: the meaning of programs, i.e. what happens when we execute
BNF - A metasyntax for syntax

- A convenient syntax for CFGs
  - ie a **metalanguage**
- non-terminals
- terminals (sometimes called tokens, lexemes)
  - reserved words, variable names, literals
BNF Example - simple assignment language

<program> ::= \texttt{begin} <stmt_list> \texttt{end}

<stmt_list> ::= <stmt> | <stmt> ; <stmt_list>

<stmt> ::= \texttt{skip} | <assgn>

<assgn> ::= <var> = <expr>

<var> ::= X | Y | Z

<expr> ::= <var> + <var> | <var> - <var> | <var>
Parse trees and ambiguity

- The legal sentences of a grammar are those strings for which there is a derivation in the BNF.
- The derivations can be transformed into **Parse Trees** and are often used by the compiler or interpreter to assign semantics to phrases.
- Some sentences may have **more than one** derivation.
- In that case we say that the **grammar is ambiguous**.
Example ambiguous parse trees

\(<expr> ::= <expr> + <expr> | <expr> * <expr> | <lit> \)
\(<lit> ::= 1 | 2 | 3 | 4 \)

Q. Why is this potentially problematic?
Solving ambiguity

- parentheses everywhere?
  - maybe if you are a Lisp programmer…

- operator precedence: typically * binds tighter than +, or is sad to have higher precedence
  - 2*3+5 is understood as (2*3)+5
Example

2+3*4 now has a unique parse tree
Associativity

- +,- and *,/ usually have the same precedence
- in that case we need to specify how to “bracket”

- note that - and / are not associative as mathematical operations: eg. (1-1)-1 is not equal to 1-(1-1)
Exercise

- Write two grammars for +, - expressions:
  - one left-associative
  - one right-associative
if-then-else ambiguity

\[
\texttt{<if stmt>} ::= \texttt{if} \texttt{<expr>} \texttt{then} \texttt{<stmt>} \texttt{else} \texttt{<stmt>}
\]

\[
\mid \texttt{if} \texttt{<expr>} \texttt{then} \texttt{<stmt>}
\]

Does the following program loop or terminate?

```plaintext
if true then
  if false
    then skip
    then skip
  else loop_forever();
skip;
```
Example - Java if

IfThenStatement:
   if ( Expression ) Statement

IfThenElseStatement:
   if ( Expression ) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:
   if ( Expression ) StatementNoShortIf else StatementNoShortIf

http://docs.oracle.com/javase/specs/
Syntactic sugar(s) for BNF

- do not increase expressivity
- although standard exists, many people roll their own

eg

\[
\begin{align*}
\langle \text{if_stmt} \rangle & ::= \textbf{if} \ \langle \text{expr} \rangle \ \textbf{then} \ \langle \text{stmt} \rangle \ [ \textbf{else} \ \langle \text{stmt} \rangle ] \\
\langle \text{if_stmt} \rangle & ::= \textbf{switch} \ (\langle \text{var} \rangle )\{ \ \{ \text{case} \langle \text{lit} \rangle : \langle \text{stmt} \rangle ; \} \} \\
\langle \text{expr} \rangle & ::= \langle \text{var} \rangle (+ \ | \ - \ | \ * \ | \ / ) \langle \text{var} \rangle
\end{align*}
\]
Example – Java for

BasicForStatement:
  for ( ForInit opt ; Expression opt ; ForUpdate opt ) Statement

ForStatementNoShortIf:
  for ( ForInit opt ; Expression opt ; ForUpdate opt ) StatementNoShortIf

ForInit:
  StatementExpressionList
  LocalVariableDeclaration

ForUpdate:
  StatementExpressionList

StatementExpressionList:
  StatementExpression
  StatementExpressionList , StatementExpression

http://docs.oracle.com/javase/specs/
Processing syntax

- We have seen how to use grammars to go from derivations to strings.
- Lexing and parsing (next week) goes the other way: Given a string, does there exist a derivation?
  - program string
  - identify tokens
    - lexer, using regular expressions
  - produce abstract syntax
    - parser, producing a parse tree