Review

- **Syntax** (form): expressed by *grammar*
- CFG/BNF: (1) terminal (2) nonterminal (3) production
- Left-recursive vs. right-recursive production
- Derivation & parse tree
  - Left-most vs. right-most derivation
- Sentential form
Resolving ambiguity: C++

selection-statement:
  if ( condition ) statement
  if ( condition ) statement else statement

“The else ambiguity is resolved by connecting an else with the last encountered else-less if.” [Stroustrup, 1991]

Resolving ambiguity: Java


- Statement:
  - IfThenStatement
  - IfThenElseStatement
  - ...
- IfThenStatement:
  - if ( Expression ) Statement
- IfThenElseStatement:
  - if ( Expression ) StatementNoShortIf else Statement

Anything other than simple if-then (known as short if)
Resolving ambiguity: Algol 68

- By syntax of if-fi
- Example:

```
if (x > 0)
    if (y > 0)
        y := -5;
    else
        y := 5;
    fi
fi
```

What about Python?

New terms

- Sentence
- Language
- Ambiguous grammar: 2 parse trees for the same sentence
Extended BNF (EBNF)
ITERATIONS, OPTIONS, AND CHOICES
BNF IS AS POWERFUL AS EBNF (SEE CLASS NOTE)

Grammar for CLite
LEXICAL SYNTAX (LOW LEVEL: TYPED THINGS)
CONCRETE SYNTAX (HIGHER LEVELS OF ABSTRACTION)
Lexical Syntax:
Lexemes → Tokens

\[
\begin{align*}
\text{Identifier} & \rightarrow \text{Letter} \{ \text{Letter} \mid \text{Digit} \} \\
\text{Letter} & \rightarrow a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z \\
\text{Digit} & \rightarrow 0 \mid 1 \mid \ldots \mid 9 \\
\text{Literal} & \rightarrow \text{Integer} \mid \text{Boolean} \mid \text{Float} \mid \text{Char} \\
\text{Integer} & \rightarrow \text{Digit} \{ \text{Digit} \} \\
\text{Boolean} & \rightarrow \text{True} \mid \text{False} \\
\text{Float} & \rightarrow \text{Integer} \cdot \text{Integer} \\
\text{Char} & \rightarrow \text{‘ASCIIChar’} \\
\end{align*}
\]

ASCIIChar is an ASCII character

Concrete Syntax (EBNF):
Hierarchy of categories/abstractions

\[
\begin{align*}
\text{Program} & \rightarrow \text{int} \ \text{main} \ ( \ ) \{ \ \text{Declarations} \ \text{Statements} \ \} \\
\text{Declarations} & \rightarrow \{ \ \text{Declaration} \ \} \\
\text{Declaration} & \rightarrow \text{Type} \ \text{Identifier} \{ \text{[ Integer ]} \} \{ , \ \text{Identifier} \{ \text{[ Integer ]} \} \} \ ; \\
\text{Type} & \rightarrow \text{int} \mid \text{bool} \mid \text{float} \mid \text{char} \\
\text{Statements} & \rightarrow \{ \ \text{Statement} \ \} \\
\text{Statement} & \rightarrow \ ; \mid \text{Block} \mid \text{Assignment} \mid \text{IfStatement} \mid \text{WhileStatement} \\
\text{Block} & \rightarrow \{ \ \text{Statements} \ \} \\
\text{Assignment} & \rightarrow \text{Identifier} \{ \text{[ Expression ]} \} = \text{Expression} ; \\
\text{IfStatement} & \rightarrow \text{if} \ ( \ \text{Expression} \ ) \ \text{Statement} \ [ \text{else} \ \text{Statement} \ ] \\
\text{WhileStatement} & \rightarrow \text{while} \ ( \ \text{Expression} \ ) \ \text{Statement}
\end{align*}
\]
Precedence of operators in C

1. ( ), [ ], Type ( )
   Parentheses, array subscript, type casting

2. +, −
   Unary + or −

3. * / %
   Multiplication, division, and remainder

4. + -
   Addition and subtraction

5. << >>
   Bitwise left shift and right shift
   We ignore in CLite

6. < <=
   For relational operators < and ≤ respectively

7. > >=
   For relational operators > and ≥ respectively

8. &
   Bitwise AND

9. ^
   Bitwise XOR (exclusive or)
   We ignore in CLite

10. |
    Bitwise OR (inclusive or)

11. &&
    Logical AND

12. ||
    Logical OR


Concrete Syntax (cont...)

Expression → Conjunction { || | | Conjunction }
Conjunction → Equality { && | Equality }
   Equality → Relation [ EquOp Relation ]
      EquOp → == | !=
   Relation → Addition [ RelOp Addition ]
      RelOp → < | <= | > | >=
   Addition → Term { AddOp Term }
      AddOp → + | −
   Term → Factor { MulOp Factor }
      MulOp → * | / | %
   Factor → [ UnaryOp ] Primary
      UnaryOp → − | !
   Primary → Identifier [ [ Expression ] ] | Literal | ( Expression ) | Type ( Expression )

Note: Operators and punctuation symbols are part of "lexical syntax" (next)
Abstract Syntax Tree

Parse Tree for $z = x + 2*y$; Fig. 2.9
Abstract Syntax Tree for 
\[ z = x + 2 \times y; \]
Fig. 2.10

Optional Reading

Pages 49 -- 54
Vocabulary

- **Syntax** (form): expressed by grammar
- CFG/BNF: (1) terminal (2) nonterminal (3) production
- Left-recursive vs. right-recursive production
- Derivation: left-most vs. right-most derivation
- Parse tree
- Sentential form
- Sentence
- Language
- Ambiguous grammar
- Concrete syntax and lexical syntax
- Abstract syntax tree