Introduction

• Syntax of a PL:
  – the form (or structure) of the expressions, statements, and program units
  – Example: \( \text{while } (<\text{boolean_expr}> ) <\text{statement}> \)

• Semantics of a PL:
  – the meaning of the expressions, statements, and program units

• Syntax and semantics are closely related; describing syntax is easier though.

• Syntax and semantics provide a language’s definition
  – Diverse users of a language definition
    • Other language designers
    • Implementers
    • Programmers (the users of the language)
Formal Definition of Languages (two ways)

- **Recognizers**
  - A recognition device (R) reads input strings over the alphabet (Σ) of the language (L) and decides whether or not the input strings belong to L
  - Example: syntax analysis part of a compiler

- **Generators**
  - A device (G) that generates sentences of a language
  - One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator (i.e. Grammar)

Generation Mechanisms: CFG & BNF

- **Context-Free Grammars**
  - Developed by Chomsky in the mid-1950s
  - He developed several classes of language generators (grammars) to describe the syntax of natural languages
  - CFGs define a class of languages called context-free languages, which became useful to define PLs

- **Backus-Naur Form (1959)**
  - A notation invented by Backus and revised by Naur to describe Algol 58 and Algol 60

- **BNF is equivalent to context-free grammars**

BNF Fundamentals

- In BNF, abstractions are used to represent classes of syntactic structures—they act like syntactic variables (also called *nonterminal symbols*, or just *nonterminals*)
- **Terminals** are lexemes or tokens
- A BNF *rule* has a left-hand side (LHS), which is a nonterminal, and a right-hand side (RHS), which is a string of terminals and/or nonterminals
- Note: in this course, we enclose Nonterminals in angle brackets
  - Examples of BNF rules:
    - `<ident_list>` → identifier | identifier, `<ident_list>`
    - `<if_stmt>` → if `<logic_expr>` then `<stmt>`

- **Grammar**: a finite non-empty set of BNF rules
- **A start symbol** is a special element of the nonterminals of a grammar

BNF Rules (another example)

- An abstraction (or nonterminal symbol) can have more than one RHS

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
<single_stmt> → <assign>
<assign> → <var>=<expr>
<var> → a | b | c
<stmt_list> → <single_stmt>;<stmt_list>
             | <single_stmt>
<expr> → ...
```
Describing Lists

- Syntactic lists are described using recursion
  \[<\text{ident\_list}> \rightarrow \text{ident} \mid \text{ident, } <\text{ident\_list}>\]

- A **derivation** is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)

An Example Grammar

- **<program>** \[\rightarrow <\text{stmts}>\]
- **<stmts>** \[\rightarrow <\text{stmt}> \mid <\text{stmt}> ; <\text{stmts}>\]
- **<stmt>** \[\rightarrow <\text{var}> = <\text{expr}>\]
- **<var>** \[\rightarrow a \mid b \mid c \mid d\]
- **<expr>** \[\rightarrow <\text{term}> + <\text{term}> \mid <\text{term}> - <\text{term}>\]
- **<term>** \[\rightarrow <\text{var}> \mid \text{const}\]

An Example Derivation

- **<program>** \[\Rightarrow <\text{stmts}>\]
  \[\Rightarrow <\text{stmt}>\]
  \[\Rightarrow <\text{var}> = <\text{expr}>\]
  \[\Rightarrow a = <\text{expr}>\]
  \[\Rightarrow a = <\text{term}> + <\text{term}>\]
  \[\Rightarrow a = <\text{var}> + <\text{term}>\]
  \[\Rightarrow a = b + <\text{term}>\]
  \[\Rightarrow a = b + \text{const}\]

Derivations

- Every string of symbols in a derivation is a **sentential form**
- A **sentence** is a sentential form that has only terminal symbols
- A **leftmost derivation** is one in which the leftmost nonterminal in each sentential form is the one that is expanded
- A derivation may be neither leftmost nor rightmost
Parse Tree

- A hierarchical representation of a derivation

\[<\text{program}> \Rightarrow <\text{stmts}> \Rightarrow <\text{stmt}> \Rightarrow <\text{var}> = <\text{expr}> \Rightarrow a = <\text{expr}> \Rightarrow a = <\text{term}> + <\text{term}> \Rightarrow a = <\text{var}> + <\text{term}> \Rightarrow a = b + \text{const}\]

ICOM 4036. Ch. 3. Syntax & Semantics 49

Ambiguity in Grammars

- A grammar is **ambiguous** if and only if it generates a sentential form that has two or more distinct parse trees

An Ambiguous Expression Grammar

\[<\text{expr}> \rightarrow <\text{expr}> <\text{op}> <\text{expr}> \mid \text{const}\]

\[<\text{op}> \rightarrow / \mid -\]

An Unambiguous Expression Grammar

- If we use the parse tree to indicate precedence levels of the operators, we can avoid this kind of ambiguity.

\[<\text{expr}> \rightarrow <\text{expr}> - <\text{term}> \mid <\text{term}>\]

\[<\text{term}> \rightarrow <\text{term}> / \text{const} \mid \text{const}\]
**Associativity of Operators**

- Operator associativity can also be indicated by a grammar

```
<expr>  →  <expr> + <expr>
    |   const  (ambiguous)
<expr>  + const
    |   <expr> → <expr> + const
          |   const  (unambiguous)
```

**Extended BNF**

- Optional parts are placed in brackets [ ]

```
<proc_call>  →  ident [(<expr_list>)]
```

- Alternative parts of RHSs are placed inside parentheses and separated via vertical bars (OR)

```
<term>  →  <term> (+|−) const
```

- Repetitions (0 or more) are placed inside braces {}

```
<ident>  →  letter {letter|digit}
```

**BNF and EBNF**

- BNF

```
<expr>  →  <expr> + <term>
    |   <expr> - <term>
    |   <term>
<term>  →  <term> * <factor>
    |   <term> / <factor>
    |   <factor>
```

- EBNF

```
<expr>  →  <term> {(+ | −) <term>}
<term>  →  <factor> {(* | /) <factor>}
```

**Recent Variations in EBNF**

- Alternative RHSs are put on separate lines
- Use of a colon instead of →
- Use of opt for optional parts
- Use of one of for choices

**Examples:**

```
EXPR : TERM {one of (+ −) TERM}
CONSTRUCTOR : NAME (FORMAL_PARAMETER opt)
```

- Given a CFG for L, a parser for L can be algorithmically constructed.