V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 3

—Slide 1—

Formal Syntax

- Syntax and semantics of a language is described by a **meta-language**
- **Abstract Syntax** lists all possible forms for each of the syntactic classes. It lists the syntactic classes along with the symbols that stand for arbitrary elements of the classes.
- **Concrete Syntax** tells us the *phrase structure* of a well-formed sentence in the grammar.
- **Example** Abstract syntax tells us that following statements are well-formed

```
if a then p
if a then p else q
if a then if b then p else q
```

Concrete syntax tells us which `then` the `else`-part matches to.

—Slide 2—

Meta-Languages for Concrete Syntax

- **Variants of CFG**
  (Context-Free Grammar)
  - BNF (Backus-Naur Form)
    (Also called PBF—Panini-Backus Form)
  - EBNF (Extended BNF)
  - Syntax Diagrams (or Syntax Charts)

—Slide 3—

Backus-Naur Formalisms

- **Terminal Symbols (Tokens):**
  Atomic Symbols in a language: `a`, `b`, ..., `1`, `2`, ..., `+`, `*`, ..., `or`, `div`, ...
- **Non-terminal Symbols (Syntactic Constructs)**:
  `<expression>`, `<term>`, `<literal>`, ...
- **Starting Nonterminal:**
  A distinguished non-terminal representing the main-construct.
- **Production Rules:**
  Rules specifying components of a construct.
  1. LHS = Non-terminal
  2. RHS = A string of terminals & non terminals

```
LHS ::= RHS
<expression> ::= <term>
               | <expression> <addop> <term>
<term> ::= <factor> | <term> <multop> <factor>
```

—Slide 4—

Example: Grammar for Expression

```
<expression> ::= <term>
          | <expression> <addop> <term>

<term> ::= <factor>
          | <term> <multop> <factor>

<factor> ::= <identifier>
          | <literal>
          | <expression>

<identifier> ::= a | b | c | ... | z

<literal> ::= 1 | 2 | 3 | ... | 9

<addop> ::= + | - | or

<multop> ::= * | / | div | mod | and
```

—Slide 5—

Phrase Structure

- The grammar specifies the phrase structure (i.e., expression, terms, factors, etc.) of the valid text...Not merely what text is recognizable.
- The phrase structure disambiguates in the following example, by assigning *higher precedence* to <multop> than <addop>.

```
 a + b*c                            a*b + c
      <expression><addop><term>
```

- The phrase structure also disambiguates by making both <multop> and <addop> *associate* to the left.

```
 a / b / c                     a - b - c
<term><multop><factor>        <expression><addop><term>
```

- In general, there is no way of testing whether a syntax is ambiguous.

—Slide 6—

Extended BNF

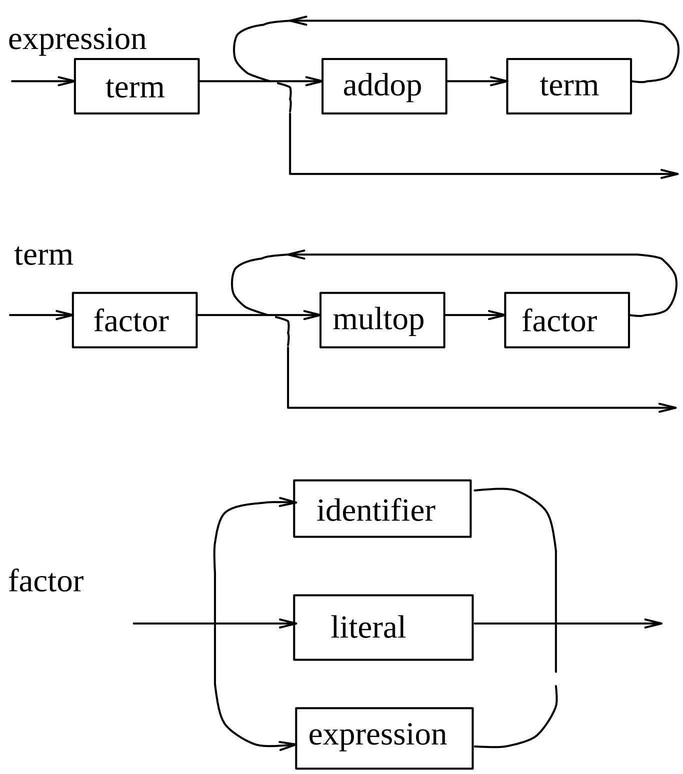
- **Nonterminals** begin with capital letters
- **Terminals** are quoted
- **Metalanguage**
  `...|...` = choice, `(...)` = grouping
  `{...}` = repetition (zero or more)
  `[...]` = optional construct
- *Example*

```
<expression> ::= <term> {<addop> <term>}
<term> ::= <factor> {<multop> <factor>}
<factor> ::= <identifier> | <literal> | <expression>
<identifier> ::= a | b | c | ... | z
<literal> ::= 1 | 2 | 3 | ... | 9
<addop> ::= + | - | or
<multop> ::= * | / | div | mod | and
```

—Slide 7—

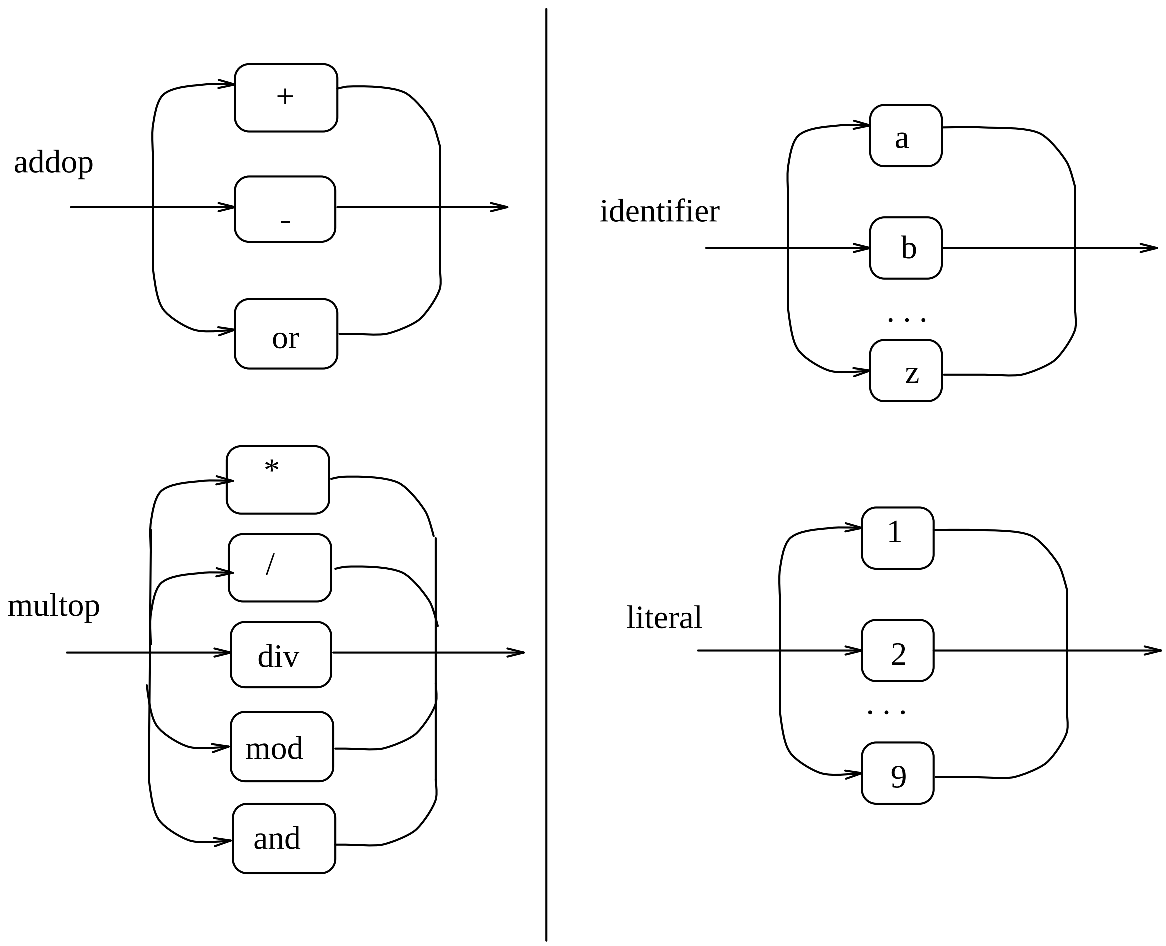
Syntax Charts

- A graphical way of writing the productions (grammar)
- **Nonterminal** ⇒ Sub-Charts
- **Productions** ⇒ Paths through the charts



—Slide 8—

Syntax Charts (contd)



—Slide 9—

Formal Semantics

- A formal description of the *semantics* of a programming language is a precise specification of the meaning of programs.
- To be used by
  Programmers, **Language Designers & Implementers**, **Theoreticians**—investigating language properties.
- **Denotational Semantics**
  A Semantic Function: mapping Syntactic Structures into Mathematical Objects.
  Denotational: Meaning of any composite phrase is expressed in terms of the meanings of its immediate constituents.

—Slide 10—

Example: Binary Numerals

- **Abstract Syntax**
  $N \in \mathbf{Nml}$ Binary Numerals

```
N ::= 0 | 1 | N0 | N1
```

- **Semantic Domain**

$$\begin{aligned} \mathbf{N} &= \{\text{zero}\} + \mathbf{N} \qquad \text{natural numbers} \\ &= \{0, 1, 2, \ldots\} \end{aligned}$$

- **Semantic Function**

$$\mathcal{N} : \mathbf{Nml} \rightarrow \mathbf{N}$$

$$\begin{aligned} \mathcal{N}[[\mathtt{0}]] &= 0 \\ \mathcal{N}[[\mathtt{1}]] &= 1 \\ \mathcal{N}[[\mathtt{N0}]] &= 2 \times \mathcal{N}[[\mathtt{N}]] \\ \mathcal{N}[[\mathtt{N1}]] &= 2 \times \mathcal{N}[[\mathtt{N}]] + 1 \end{aligned}$$

—Slide 11—

Semantic Domains

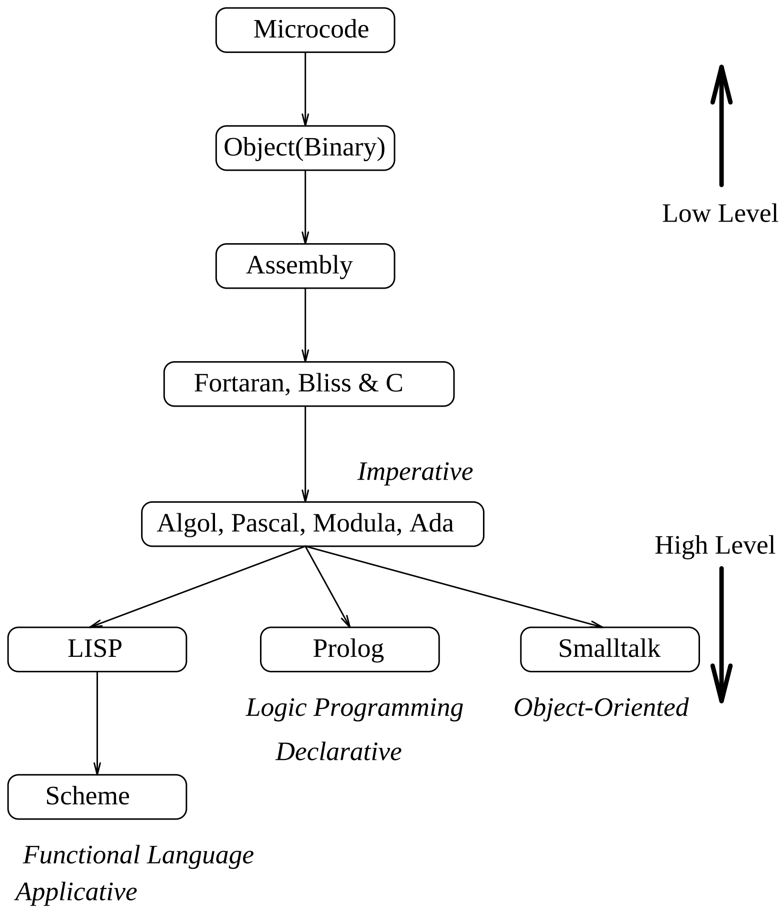
- **Basic Values**, **B**: E.g., Truth Values, Integers, ...
- **Stores**, $s \in \mathbf{S} = \mathbf{Ide} \to (\mathbf{B} + \{\text{unused}\})$
  Maps an identifier $I \in \mathbf{Ide}$ to its value $s[[I]]$.
- **Expressions**, $E \in \mathcal{E} : \mathbf{Exp} \to \mathbf{S} \to (\mathbf{B} + \{\text{error}\})$
  $\mathcal{E}[[E]]s$ is the value of E relative to store s.
- **Commands**, $C \in \mathcal{C} : \mathbf{Com} \to \mathbf{S} \to (\mathbf{S} + \{\text{error}\})$
  $\mathcal{C}[[C]]s$ is the result of executing C relative to store s.
- Thus, we have *semantics of* **;**

$$\mathcal{C}[[C_1; C_2]]s = \begin{cases} \mathcal{C}[[C_2]]g, & \text{if } g \in \mathbf{S} \\ & \text{where } g = \mathcal{C}[[C_1]]s \\ \text{error}, & \text{otherwise.} \end{cases}$$

- Other domains: **Environments**, **Continuations**, ...

—Slide 12—

Hierarchy of Languages



- "Low Level" means close to machine language
- "High Level" means away from machine language
  —Closer to natural description of an algorithm.

—Slide 13—

Classes of Language

- **Imperative** (procedural), **Applicative** (functional) & **Declarative**
- **Imperative**:
  - A program is a sequence of commands changing state/store.
  - Objects are constructed by these statements.
- **Applicative**:
  - A program is a sequence of function definitions and applications.
- **Declarative**:
  - Objects are described rather than constructed.

[End of Lecture #3]