CHAPTER 2: Programming Language Syntax

CMPSC 460 – Principles of Programming Languages

Syntax And Semantics

- Programming language syntax:
  - How programs look, their form and structure
  - Syntax is defined using a kind of formal grammar
- Programming language semantics:
  - What programs do, their behavior and meaning
  - Semantics is harder to define

Lexical Structure & Phrase Structure


```
result = 2 * count + 17.65;
```

Lexical Structure & Phrase Structure

- Usually there are two separate components
  - lexical structure
  - syntactical structure

BNF and Context-Free Grammars

- Context-Free Grammars
- Backus-Naur Form (1958)

ALGOL 60 Specifications

```
<adding operator> ::= + | -
<multiplying operator> ::= × | / | ÷
<primary> ::= <unsigned number>|<variable>|<function designator>|
              (<arithmetic expression>)
<factor> ::= <primary>|<factor>↑<primary>
<term> ::= <factor>|<term><multiplying operator><factor>
<simple arithmetic expression> ::= <term>|<adding operator><term>|
                 <simple arithmetic expression><adding operator><term>
<if clause> ::= if <Boolean expression> then
<arithmetic expression> ::= <simple arithmetic expression>|
     <if clause><simple arithmetic expression>else<arithmetic expression>
```

BNF Grammars

- A BNF consists of
  - A set of *terminals* *T*
  - A set of *non-terminals* *N*
  - A *start symbol* *S* (a non-terminal)
  - A set of *production rules*

Grammar Rules

Productions:

- has a left-hand side, the separator : :=, and a right-hand side
  - The left-hand side is a single non-terminal
  - The right-hand side can be either a token or a non-terminal

BNF Grammars

- BNF can be used in two different ways:
  - Language Recognizers
  - Language Generators

Alternatives

<exp> ::= <exp> **+** <exp> | <exp> ***** <exp> | **(** <exp> **)**
| **a** | **b** | **c**

<exp> ::= <exp> **+** <exp>
<exp> ::= <exp> ***** <exp>
<exp> ::= **(** <exp> **)**
<exp> ::= **a**
<exp> ::= **b**
<exp> ::= **c**

Example 1

<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> - <var>
| <var>

- Examples:
  1. begin A = A + B; B = C end
  2. begin B = B + C + A end

BNF Grammars

- How do we define the syntax of :
  a) An assignment statement
  b) A list of one or more function arguments
  c) An if statement without an else part
  d) A list of zero or more function arguments

Empty

- The special nonterminal <*empty*> is for places where you want the grammar to generate nothing

<*if-stmt*> ::= **if** <*expr*> **then** <*stmt*> <*else-part*>
<*else-part*> ::= **else** <*stmt*> | <*empty*>

An Ambiguous Grammar 1


```
<s>  ::= <v> = <e> | <s>;<s> | if <b> then <s> else <s>
<v>  ::= x | y | z
<e>  ::= <v> | 0 | 1 | 2 | 3 | 4
<b>  ::= <e> == <e>
```

```
x = 1; y = 2; if x==y then y=3
```

John C. Mitchell, Concepts in Programming Languages, Cambridge University Press, 2002

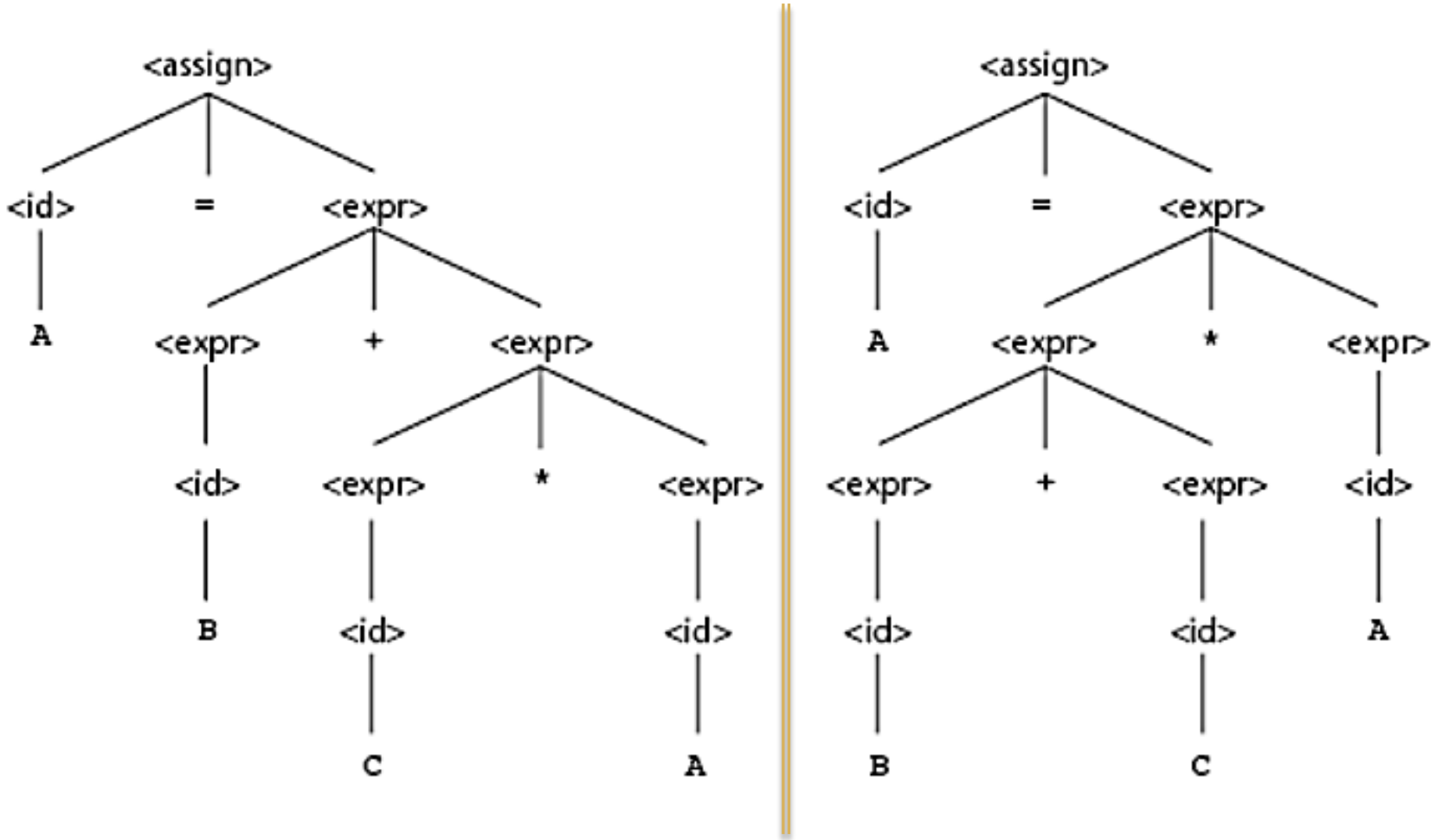
An Ambiguous Grammar 2


```
<asgn>  ::= <id> = <expr>
<id>    ::= A | B | C
<expr>  ::= <expr> + <expr>
          | <expr> * <expr>
          |( <expr> )
          | <id>
```

- Example:
  - A = B + C * A

From: Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012

An Ambiguous Grammar 2



From: Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012

An Example Grammar


```
<asgn>  ::= <id> = <expr>
<id>    ::= A | B | C
<expr>  ::= <id> + <expr>
          | <id> * <expr>
          |( <expr> )
          | <id>
```

From: Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012

An unambiguous Grammar

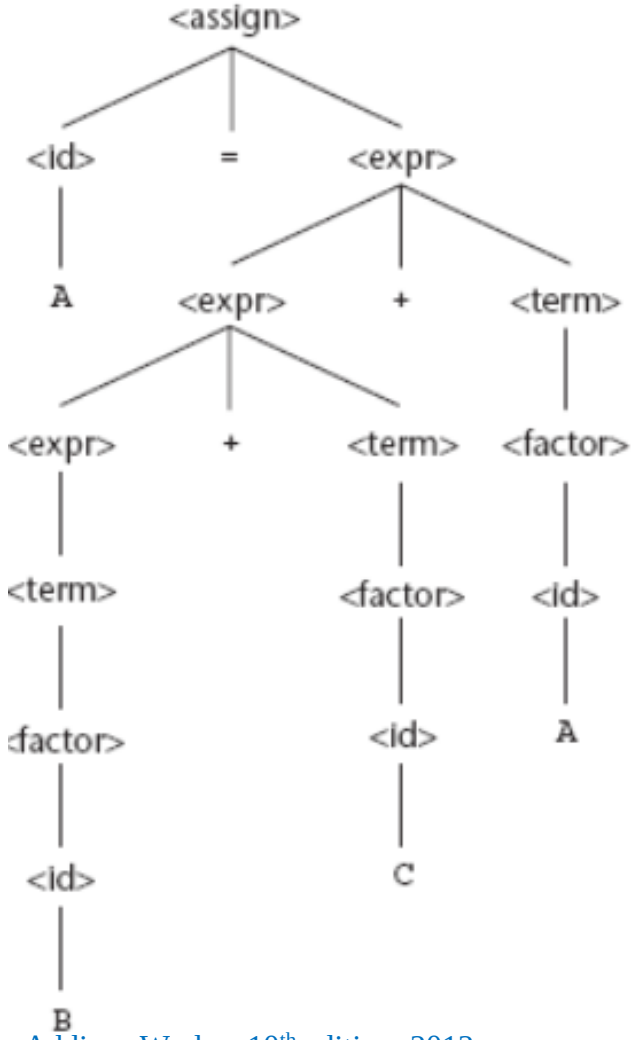

```
<asgn>   ::= <id> = <expr>
<id>     ::= A | B | C
<expr>   ::= <expr> + <term>
           |  <term>
<term>   ::= <term> * <factor>
           |  <factor>
<factor> ::= ( <expr> )
           |  <id>
```

From: Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012

Associativity

- A = B +C + A

```
<asgn>   ::= <id> = <expr>
<id>     ::= A | B | C
<expr>   ::= <expr> + <term>
           |  <term>
<term>   ::= <term> * <factor>
           |  <factor>
<factor> ::= ( <expr> )
           |  <id>
```



From: Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012

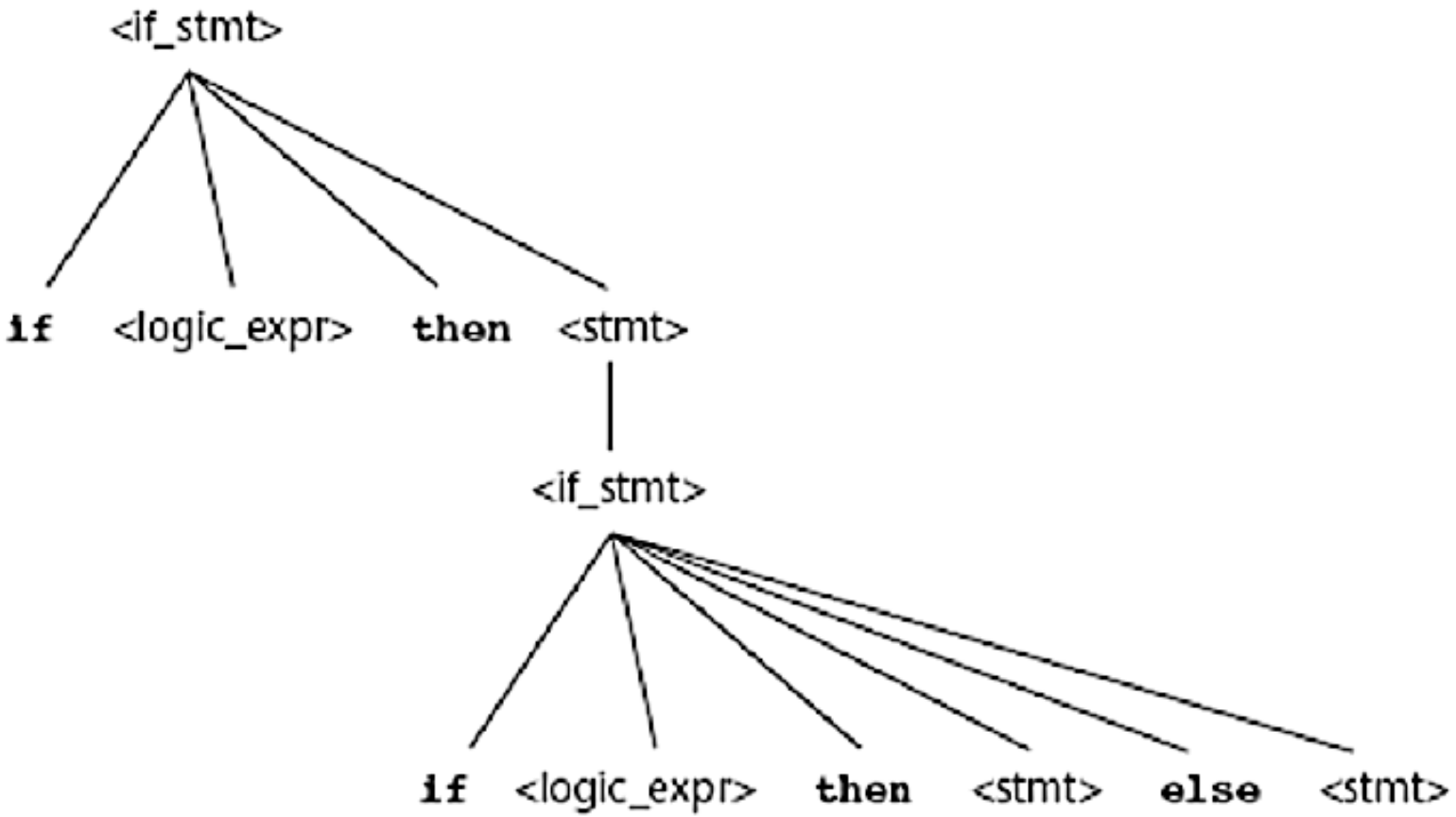
An Ambiguous Grammar 3


```
int x = 3, y = 4;

if ( x > 5 ) then if ( y > 5 ) then write( "x & y  > 5" ); else  write( "x is <= 5" );

if ( x > 5 ) then
    if ( y > 5 ) then write( "x & y  > 5" );
else  write( "x is <= 5" );

if ( x > 5 ) then
    if ( y > 5 ) then write( "x & y  > 5" );
    else  write( "x is <= 5" );
```

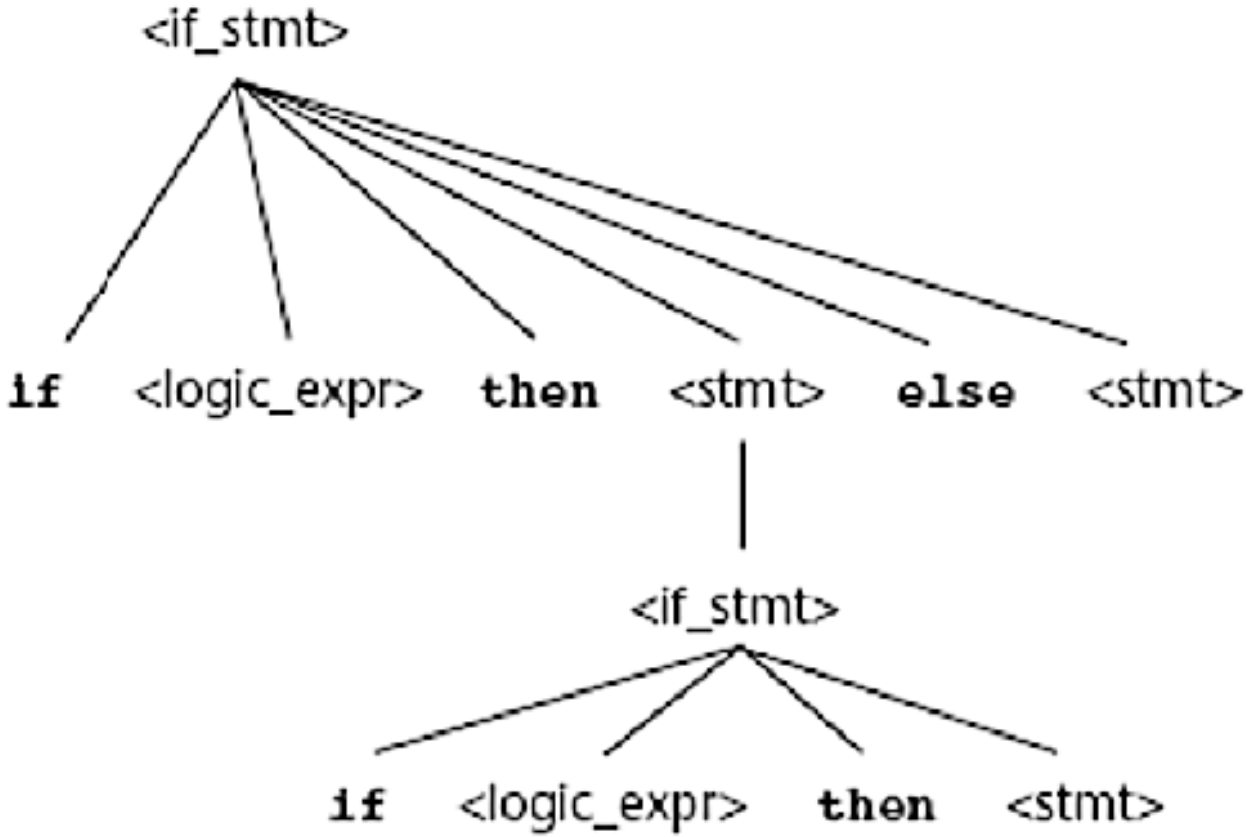
<*if-stmt*> **::=** **if** <logic_*expr*> **then** <*stmt*>

<*if-stmt*> **::=** **if** <logic_*expr*> **then** <*stmt*> **else** <*stmt*>

Context-Free Grammars



Context-Free Grammars



References

- Michael L. Scott, Programming Language Pragmatics, Morgan Kaufmann, 3rd edition, 2009.
- Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012
- Adam Brooks Webber, Modern Programming Languages, Franklin, Beedle & Associates Inc., 2nd edition, 2010.
- John C. Mitchell, Concepts in Programming Languages, Cambridge University Press, 2002.