# Parsing/Syntactic Analysis

CIS*4650 (Winter 2020)

# Review

- Compiler: translates a source program into target machine code
  - Front-End Analysis: language-specific, focused on analysis, systematic solutions with tools
  - Back-End Synthesis: machine-specific, focused on synthesis, and ad hoc solutions through coding

- Front-End Analysis:
  - Scanning/Lexical Analysis: break an input stream into a token sequence
  - <u>Parsing/Syntactic Analysis:</u> parse the phrase structure of a program
  - Semantic Analysis: calculate the meaning and generate the intermediate code of a program

# What is a Parser?

- Validate the ordering of tokens and determine the phrase structure (syntax tree):

# Limits of Regular Expressions

- Languages with the pattern “$a^n b^n$” are not regular languages, since they can’t be expressed by a finite number of states.

```
digits = [0-9]+
sum = (digits "+")* digits          e.g., 28 + 301 + 9

digits = [0-9]+
sum = expr "+" expr
expr = "(" sum ")" | digits

e.g., (109 + 23), 61, (1 + (250 + 3))

expr = "(" expr "+" expr ")" | digits
expr = "(" ( "(" expr "+" expr ")" | digits) "+" expr ")" | digits
…
```

# Towards a Context-Free Grammar

- Allow recursive rules:

  sum = expr "+" expr
  expr = "(" sum ")" | digits

- Remove alternations:

  expr = a b (c | d) e
  aux = c | d
  expr = a b aux e

  aux = c
  aux = d
  expr = a b aux e

- Remove Kleene closures:

  expr = (a b c)*

  expr = a b c expr
  expr = ε

# Context-Free Grammars

- Terminals: atomic symbols/tokens for a language (shown in lower cases)

- Non-terminals: symbols/variables that specify the phrases of the language (shown in UPPER CASES)
  - Non-terminals can be described recursively.

- Productions: rules relating variables to their definitions
  - Syntax: symbol -> symbol symbol … symbol
  - Terminals can't appear on the left-hand-side (LHS)

- Start symbol (S): a special symbol that starts all derivations

# Example CFG

- Arithmetic expressions with + and - operators, id tokens, and balanced parentheses:

  S -> EXPR

  EXPR -> EXPR + EXPR

  EXPR -> EXPR – EXPR

  EXPR -> (EXPR)

  EXPR -> id

  - Why +, -, (, and ) need not be quoted?

# Derivations

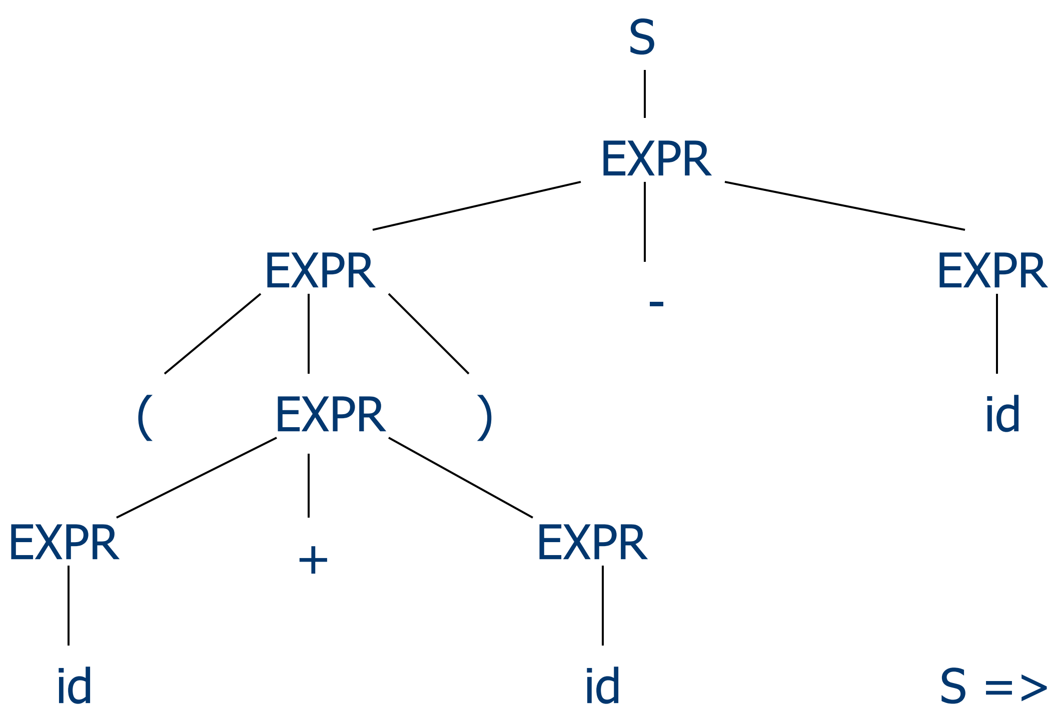
- Derivation: begin with the start symbol and repeatedly replace a non-terminal with one of its RHS's.
- Sentence: a derivation becomes a sentence if every symbol in it is a terminal.

```
S => EXPR
  => EXPR + EXPR
  => EXPR + id
  => id + id
```

```
S => EXPR
  => EXPR - EXPR
  => (EXPR) - EXPR
  => (EXPR) – id
  => (EXPR + EXPR) – id
  => (EXPR + id) – id
  => (id + id) - id
```

# Parse Trees

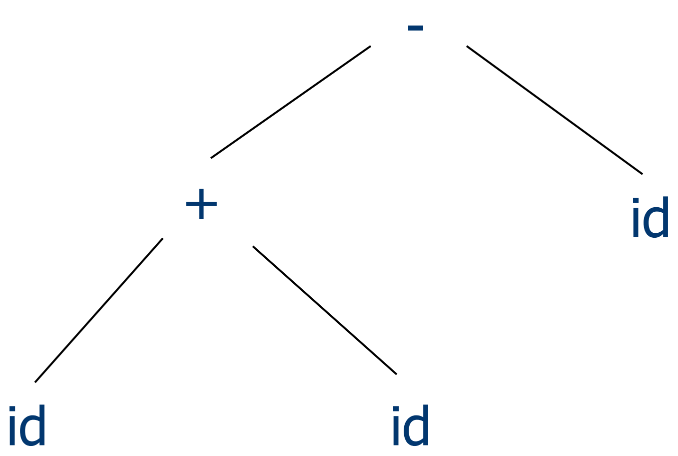
- Graphically show a derivation with LHS connected to its RHS components:
  - Several derivations can have the same parse tree.




S =>* (id + id) - id

# Abstract Syntax Trees

- Only capture the information needed for semantic analysis and code generation:
  - Also called syntax trees for short



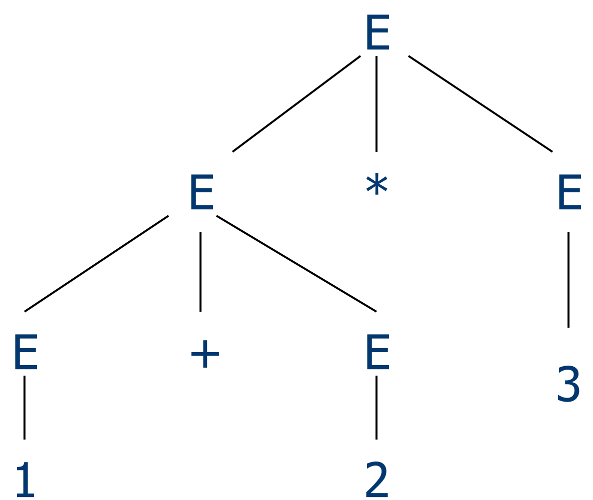
# Derivation Sequence

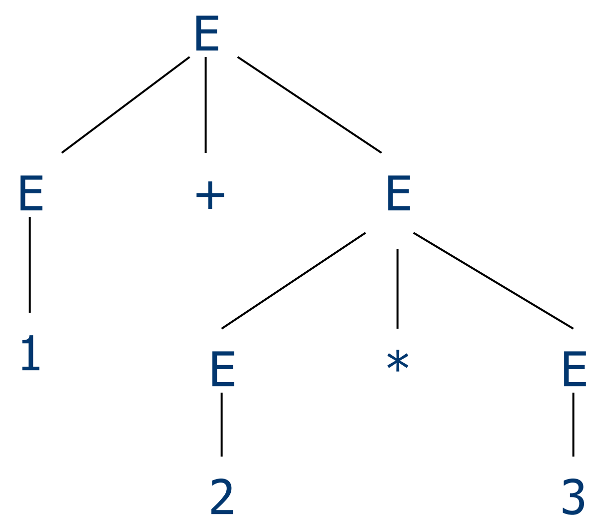
- Many different possible derivations of the same sentence
  - If more than one non-terminal appears on the LHS of a production, we can choose which to expand next.
- Two obvious conventions:
  - Leftmost derivation:
    - Choose leftmost non-terminal to expand
    - Top-down parsing process
  - Rightmost derivation:
    - Choose rightmost non-terminal to expand
    - Bottom-up parsing process

# Ambiguous Grammars

- A grammar is ambiguous if we can derive a sentence with two or more different parse trees.

E -> id
E -> num
E -> E * E
E -> E / E
E -> E + E
E -> E - E
E -> (E)

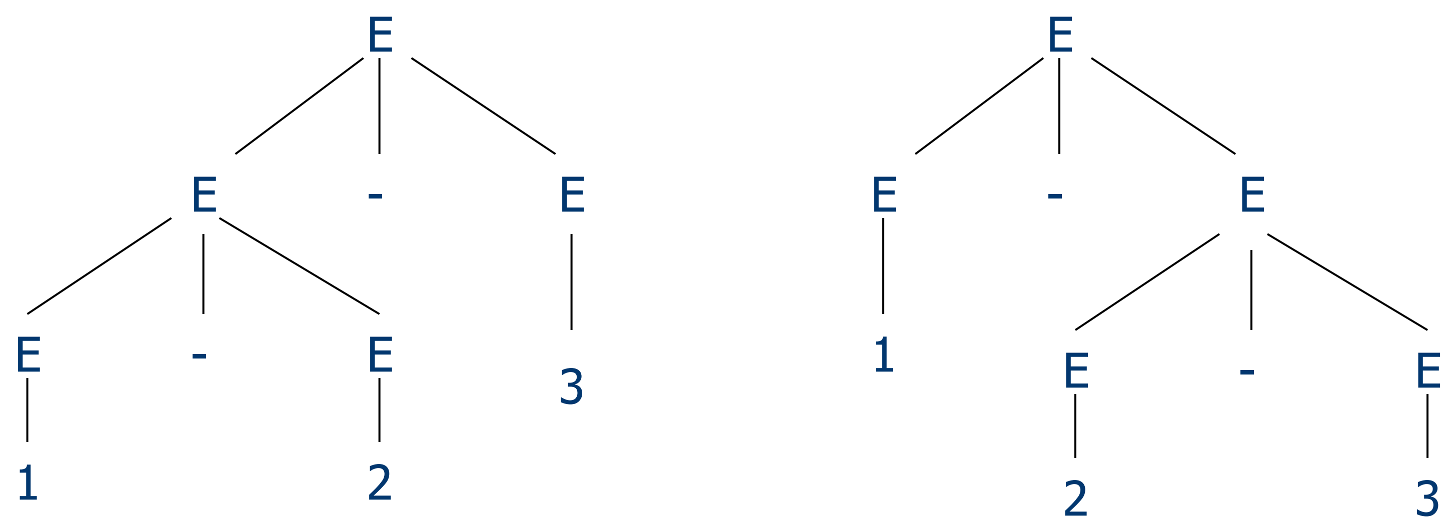


# Ambiguous Grammars

- Ambiguity can also occur for the same operations:

| | |
|---|---|
| E -> id | E -> E + E |
| E -> num | E -> E - E |
| E -> E * E | E -> (E) |
| E -> E / E | |



# Resolving Ambiguities

- Explicitly state which parse tree is correct
  - No change required to the grammar

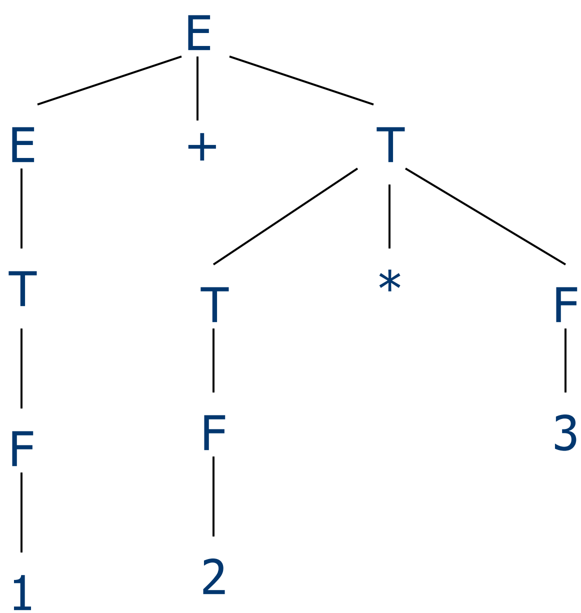
- Precedence:
  - Stated order of derivation based on operator
  - Sub-trees are evaluated before the root expression
    - The order of derivations is opposite to the order of evaluations.

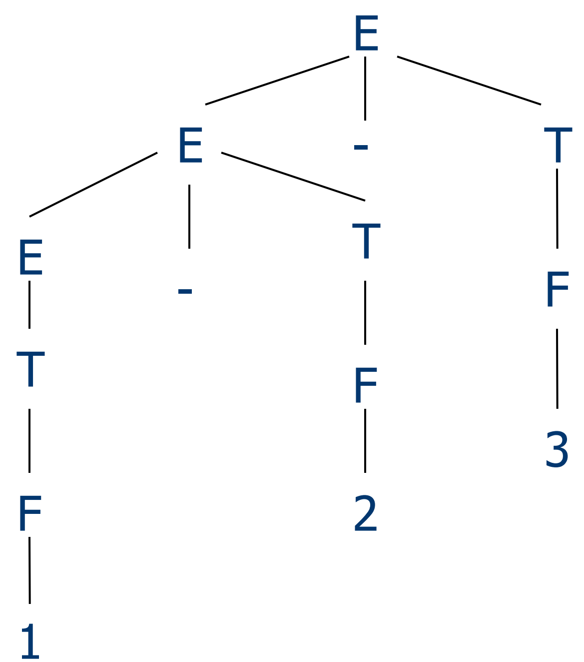
- Associativity:
  - Stated order of derivation based on location
  - Left associative: derivation from the first choice
  - Right associative: derivation from the last choice

# Resolving Ambiguities

- An unambiguous grammar for expressions:

| | | |
|---|---|---|
| E -> E + T | T -> T * F | F -> id |
| E -> E - T | T -> T / F | F -> num |
| E -> T | T -> F | F -> (E) |



# Resolving Ambiguities

- Associativity of operators:
  - Left recursion specifies left associativity:

    E -> E – T    T -> T / F

  - Right recursion specifies right associativity:

    F -> B ** F    B -> id
    F -> B         B -> num
                   B -> (E)

# Backus Normal Form

- BNF is equivalent to context-free grammars.
- In BNF, non-terminals are represented by pointed brackets:

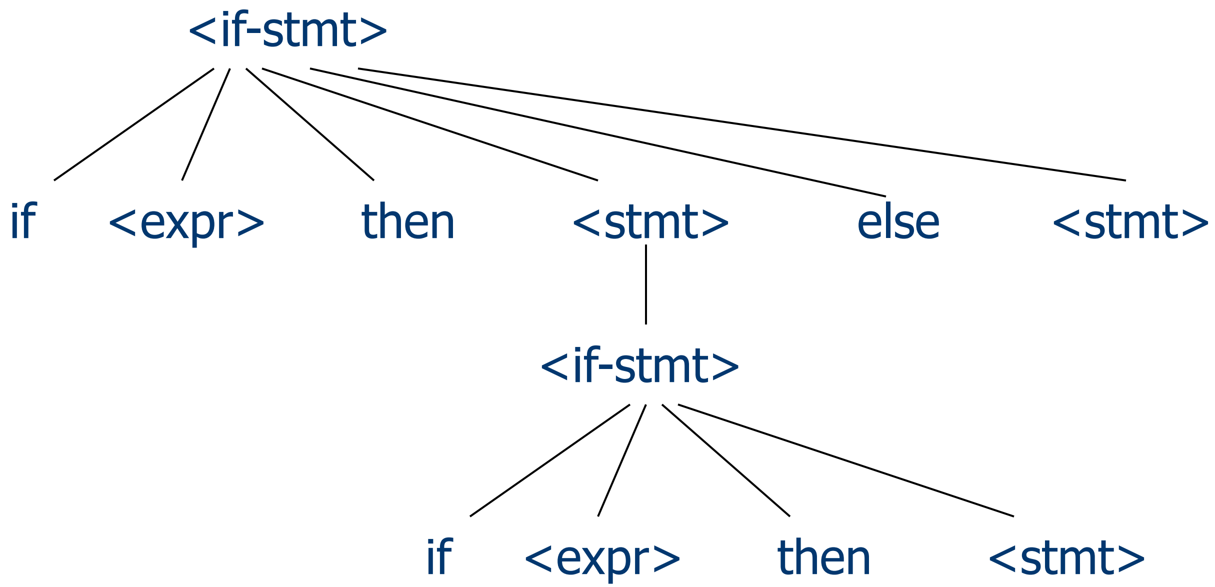
```
<program> -> begin <stmt-list> end
<stmt-list> -> <stmt> | <stmt> ; <stmt-list>
<stmt> -> id := <expr>
<expr> -> <expr> + <term> | <expr> - <term> | <term>
<term> -> <term> * <factor> | <term> / <factor> | <factor>
<factor> -> (<expr>) | id | num
```

- Extended BNF includes shorthand notations for repetitive and optional constructs.

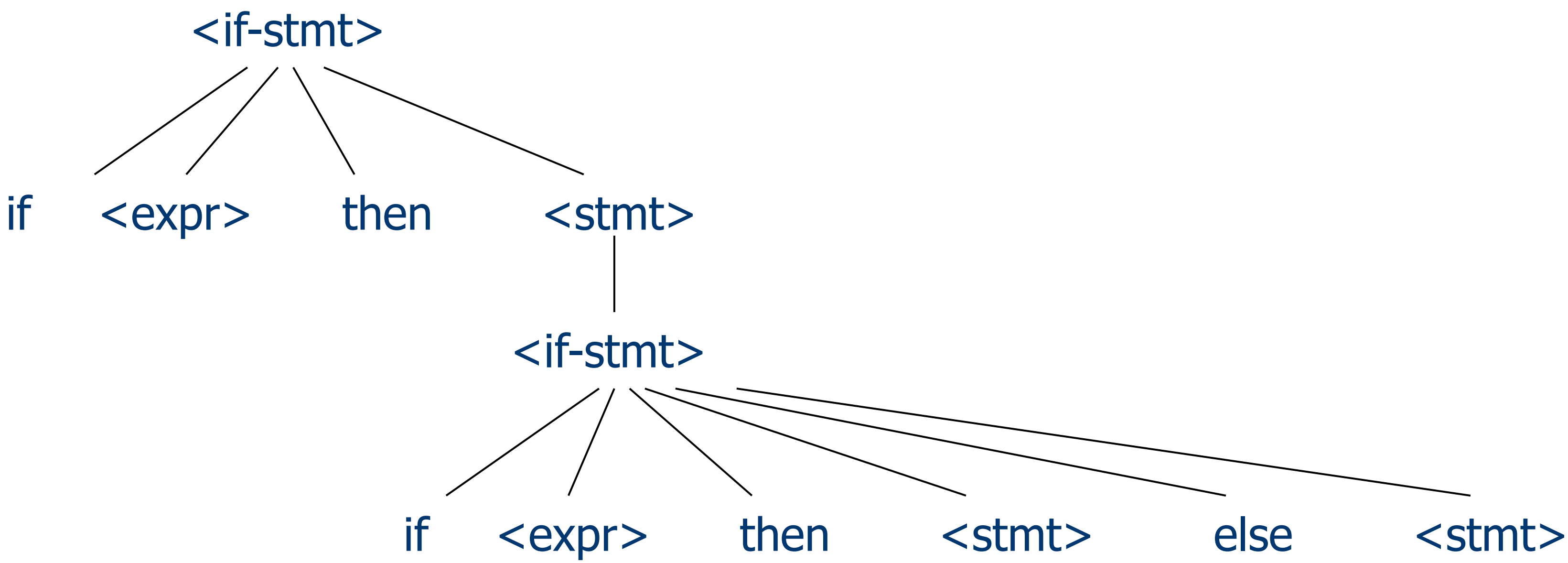
# Dangling Else Problem

- Ambiguous grammar for if-statements:

<stmt> -> <if-stmt> | <other-stmt>
<if-stmt> -> if <expr> then <stmt> |
if <expr> then <stmt> else <stmt>



# Dangling Else Problem




- Unambiguous grammar for if-statements:

<stmt> -> <matched> | <unmatched><br>
<matched> -> if <expr> then <matched> else <matched> |<br>
                <other-stmt><br>
<unmatched> -> if <expr> then <stmt> |<br>
                    if <expr> then <matched> else <unmatched>

# Chomsky's Language Hierarchy

- Classes of grammars:
  - Regular grammar

    A -> zB | z   or   A -> Bz | z   but not both
  - Context-free grammar

    A -> B where A is a non-terminal and B is any string
  - Context-sensitive grammar

    xAz -> xBz where A is a non-terminal and B is any string
  - Unrestricted grammar

    Also called recursively enumerable

- Context issues in programming languages:
  - Variables are declared before being used
  - Disambiguating rules
  - Deferred to a parser generator or semantic analyzer