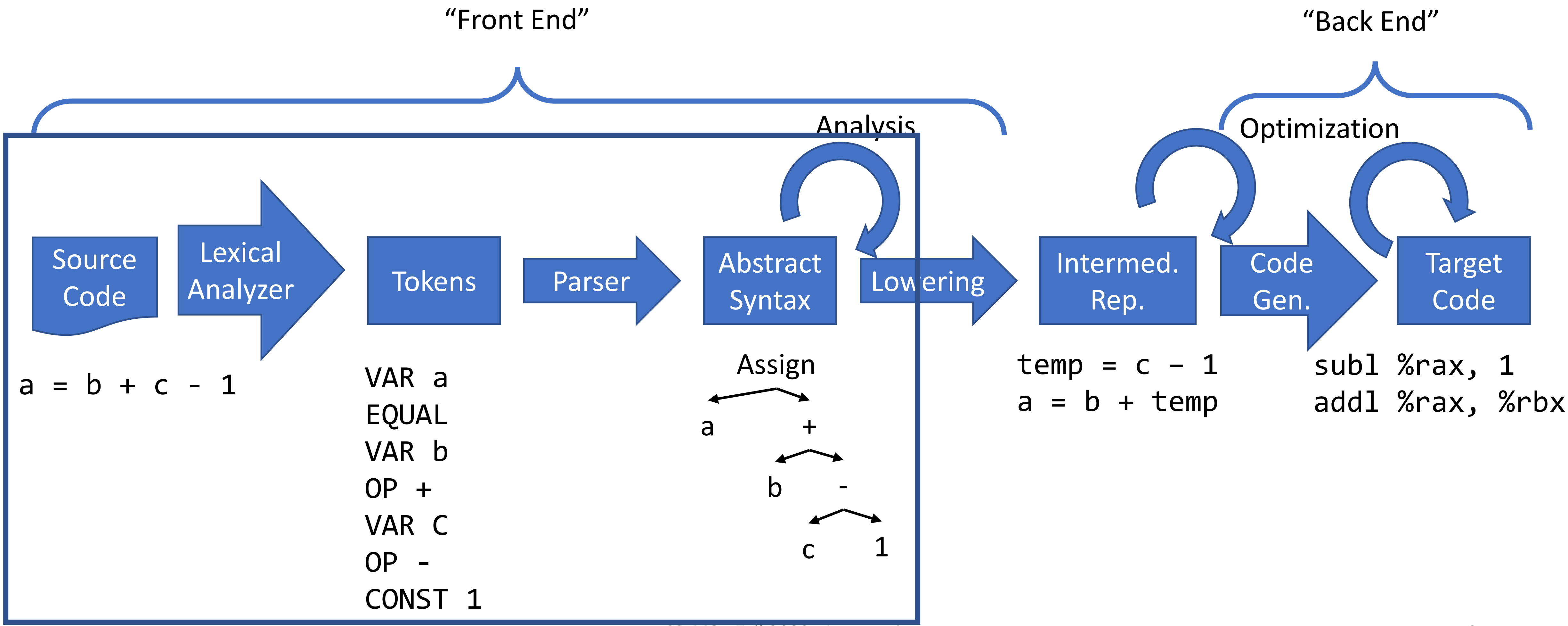
# CS443: Compiler Construction

Lecture 1: Lexing and Parsing

# Compilers translate code in phases



# Terminology

- *Lexical analysis* “lexing”
- Performed by *lexical analyzer* “lexer”
- Produces stream of *tokens*

# Tokens are specified using a *regular* grammar

- Regular expressions R:

| | | |
|---|---|---|
| • ε | Empty string | |
| • abc | Exactly the string abc | Literal |
| • $R_1 R_2$ | $R_1$ followed by $R_2$ | Concatenation |
| • $R_1 \mid R_2$ | $R_1$ or $R_2$ | Alternation |
| • $R^*$ | Zero or more $R$ | Kleene Star |
| | | |
| • $R^+$ | One or more $R$ | |
| • $R?$ | Optional $R$ | |
| • [a-z] | a, b, c, d, …, z | |

# Tokens are specified using a *regular* grammar

digit ::= [0-9]

alpha ::= [a-z]

ident ::= alpha (alpha | digit)$^*$

num ::= digit$^+$

ident → IDENT s

num → NUM s

“while” -> WHILE

“+” -> PLUS…

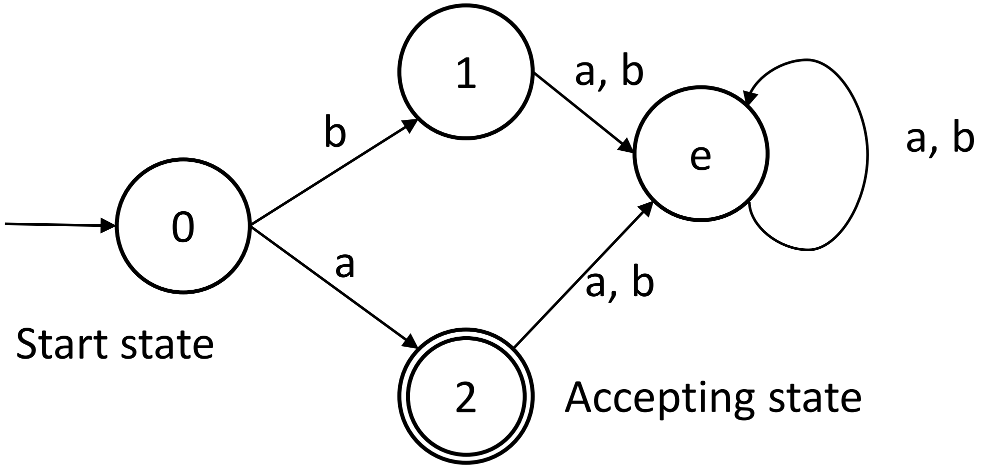
# Lexing examples

- while (i < 5) WHILE; LPAREN; IDENT “i”; LT; NUM 5; RPAREN
- while i < 5) WHILE; IDENT “i”; LT; NUM 5; RPAREN
- whole (i < 5) IDENT “whole”; LPAREN; IDENT i; LT; NUM 5; RPAREN

Might be syntax errors during *parsing. Not* errors during lexing.

# Regex matching can be done by finite state machines (FSMs)

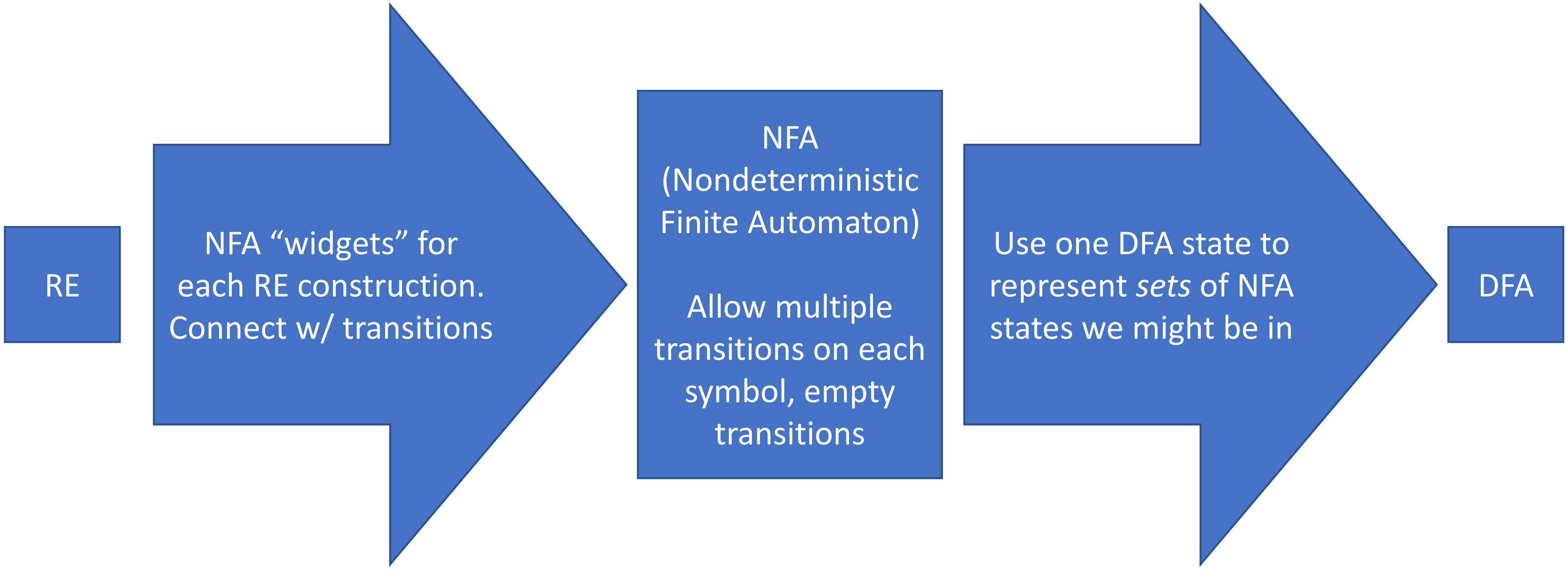
- Deterministic Finite Automaton (DFA)
- *States* + *Transition function* + *Start state* + Set of *Accepting states*




Accepts exactly “a”

# Can convert regexes to DFAs

- Full algorithm in Appel, PDB. General idea:



- Draw DFA example

# BNF grammars are “context-free”

$$e ::= \mathrm{x} \mid \mathit{num} \mid e\ \mathit{bop}\ e \mid \mathit{uop}\ e$$

nonterminal

terminals

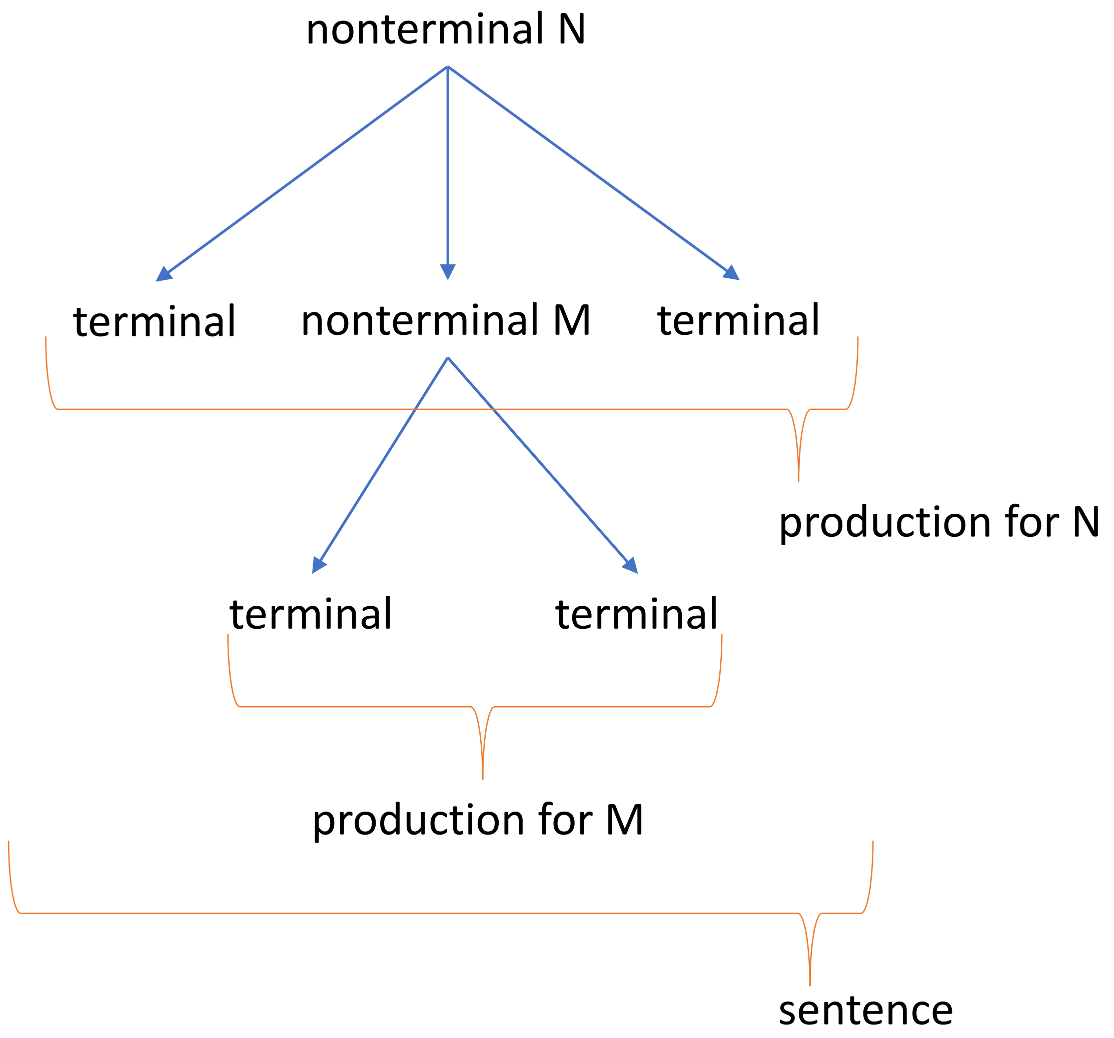
productions

# Derivation: Expand one nonterminal at a time using a production

e ::= n | e + e | (e)          Input: 1 + (2 + 3)

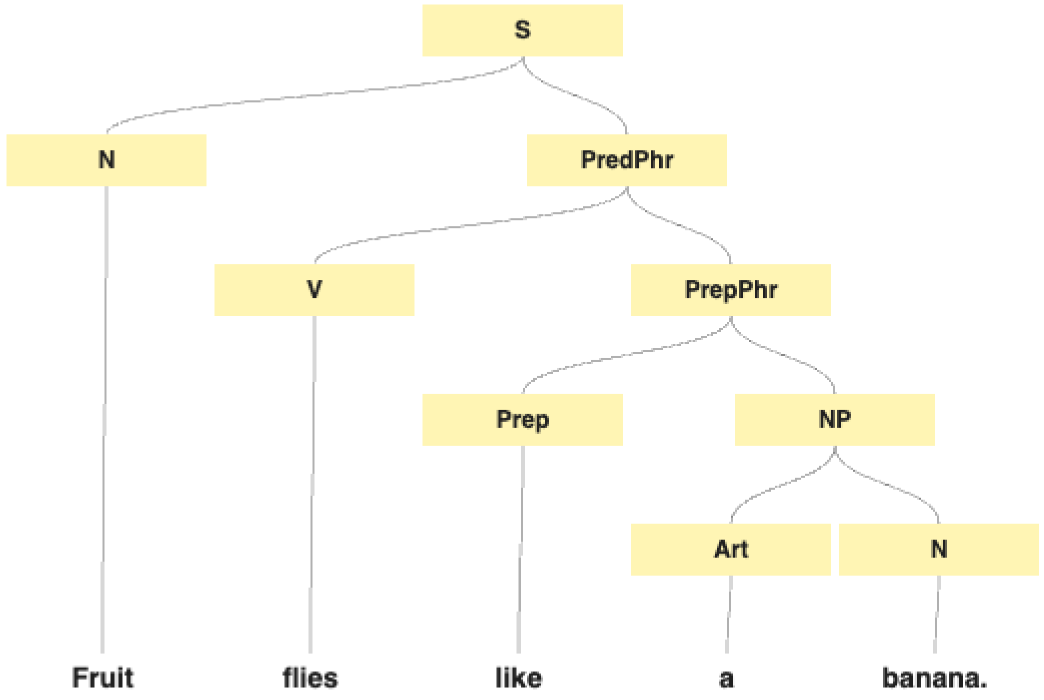
| Leftmost Derivation | Rightmost Derivation |
|---|---|
| e | e |
| e + e | e + e |
| 1 + e | e + (e) |
| 1 + (e) | e + (e + e) |
| 1 + (e + e) | e + (e + 3) |
| 1 + (2 + e) | e + (2 + 3) |
| 1 + (2 + 3) | 1 + (2 + 3) |

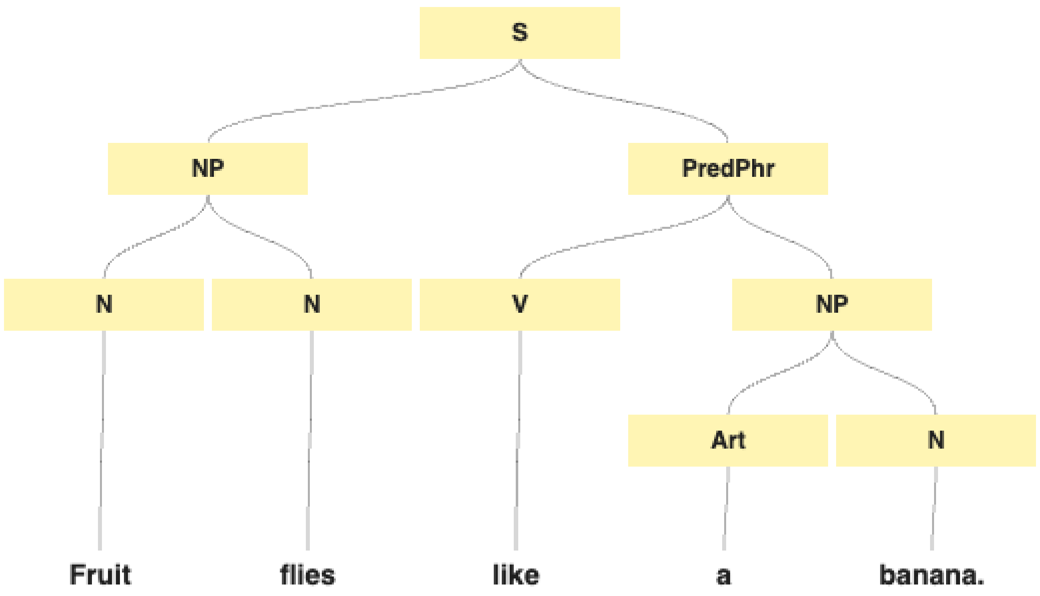
# Parsing: Produce a *parse tree* from a stream of tokens



# Ambiguous grammars allow multiple correct parse trees

- “Fruit flies like a banana”



# *Associativity* is one source of ambiguity

- $e$ ::= num | $e + e$ | $e - e$
- 1 + 2 + 3 + 4 → ((1 + 2) + 3) + 4, (1 + 2) + (3 + 4), …
- Solution:
  $e$ ::= num | e + num | e – num | $e$ + ($e$) | $e$ – ($e$)

# *Precedence* is one possible source of ambiguity

- Abstract syntax: $e ::= e + e \mid e - e \mid e * e \mid e / e$
- 1 + 2 * 3 – 4 ????
- Solution: Factoring out productions
- $f ::=$ num $\mid$ $(e)$
  $t ::= f \mid t * f \mid t / f$
  $e ::= t \mid e + t \mid e - t$

# Classic example: “dangling else”

- *s ::=* if *e s* | if *e s* else *s*

- if $e_1$ if $e_2$ $s_1$ else $s_2$
  - By convention: if $e_1$ (if $e_2$ $s_1$ else $s_2$)

- Solution:
  *closedstmt* ::= if *e closedstmt* else *closedstmt*
  | … (non-if stmts not ending with an *openstmt*)
  *openstmt* ::= if *e closedstmt* else *openstmt* | if *e stmt*
  | … (non-if stmts ending with an *openstmt*)
  *stmt* ::= *openstmt* | *closedstmt*