Lecture 6

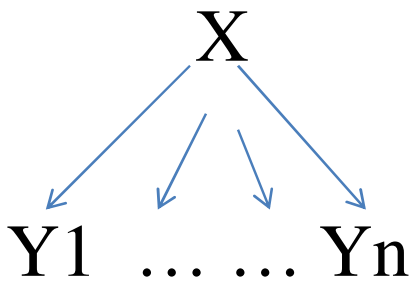
Syntax Analysis II

Derivation

- A derivation is the sequence of productions

$$S \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots \rightarrow \ldots$$

- A derivation can be drawn as a tree
  - Start symbol is the tree's root
  - For a production $X \rightarrow Y_1 \ldots \ldots Y_n$ add children $Y_1 \ldots \ldots Y_n$ to node X



Example: Left-most derivation

- Grammar: E→ E + E | E * E | (E) | id
- String : id*id+id

E

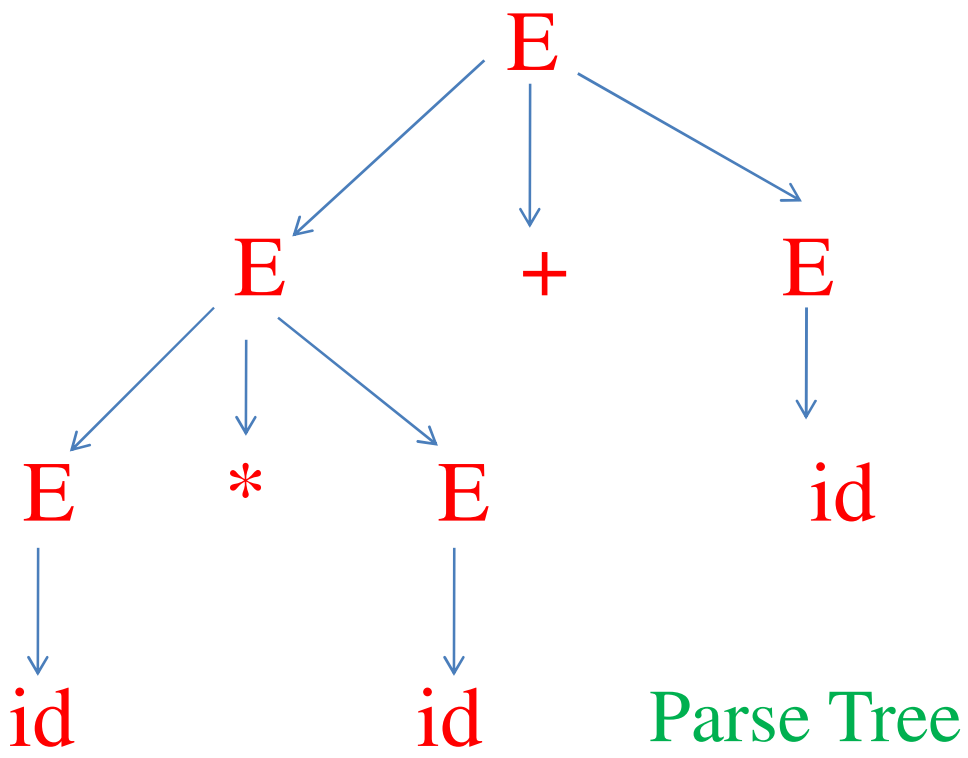
→ E + E

→ E * E + E

→ id * E + E

→ id * id + E

→ id * id + id



Parse Tree

Parse Tree

- A parse tree has
  - Terminals at the leaves
  - Non-terminal at the interior nodes
- An in-order traversal of the leaves is the original input
- The parse tree shows the association of operations, the input string does not

Right-most derivation

- The example is a left-most derivation
  - At each step, replace the left most non terminals

E

→ E + E

→ E * E + E

→ id * E + E

→ id * id + E

→ id * id + id

- There is a equivalent notion of right-most derivation

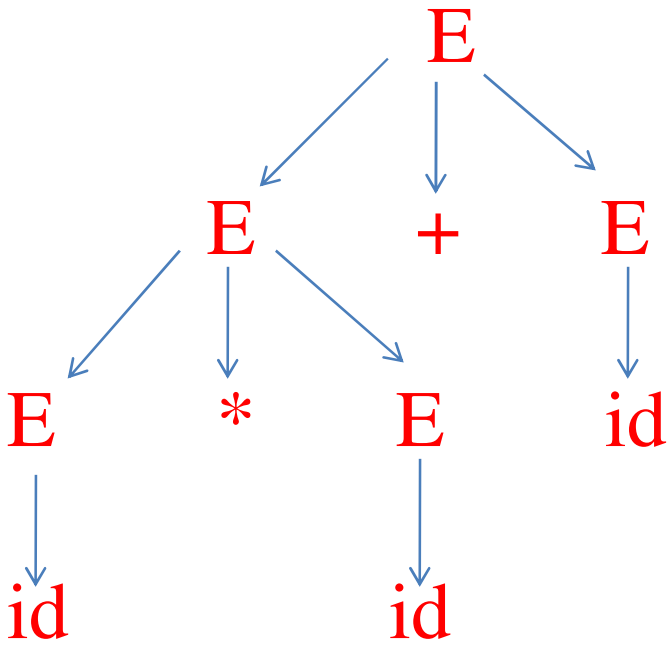
Parse Tree Derivations

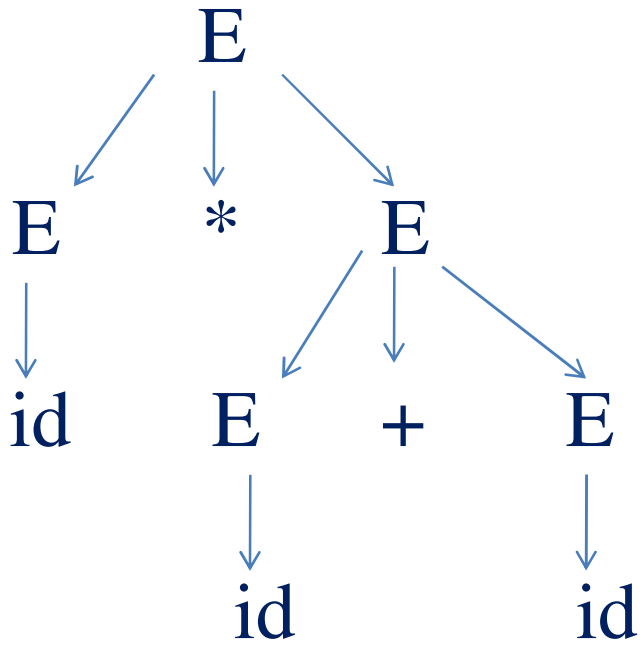
- Note that the left-most and right-most derivation have the same parse tree

- Every parse tree has the left-most and right-most derivations

Ambiguity

- Grammar: E→ E + E | E * E | (E) | id
- String : id*id+id
- This string has two parse trees





Ambiguous Grammar

- A grammar is ambiguous if it has more than one parse tree for some string
  - Equivalently, there is more than one right-most or left-most derivation for some string

- Ambiguity means that some programs are ill-defined

How to handle Ambiguous Grammar

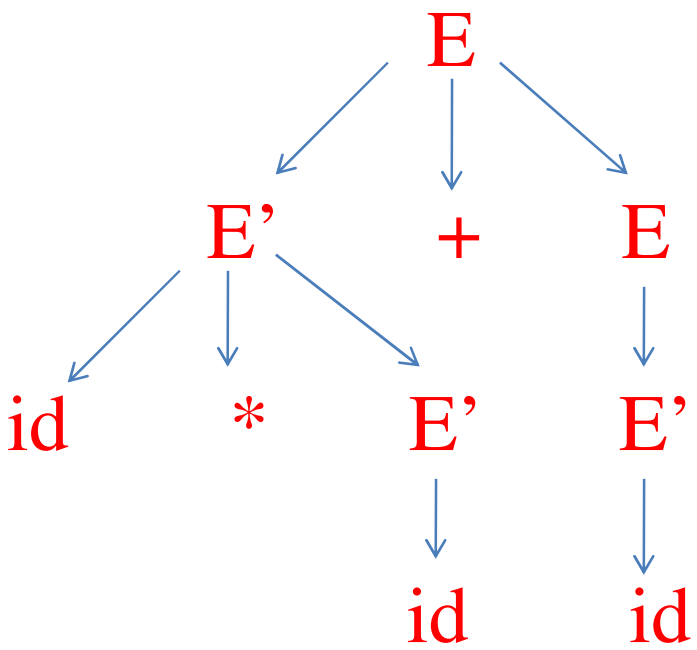
- Several ways to handle
- Rewrite the grammar unambiguously

$$E \rightarrow E' + E \mid E'$$
$$E' \rightarrow id * E' \mid id \mid (E) * E \mid (E)$$

- Enforce precedence of * over +

How to handle Ambiguous Grammar

- Grammar: E→ E'+ E | E'
  E'→ id * E' | id | (E) * E' | (E)
- String : id*id+id



Enforce precedence of * over + by divide the productions in two classes; one handle + and one handles *; so one non terminals for each operators

E → E'+ E → E' + E' + E →
E'+E'+E'+E → E'+… +E'
Handles +

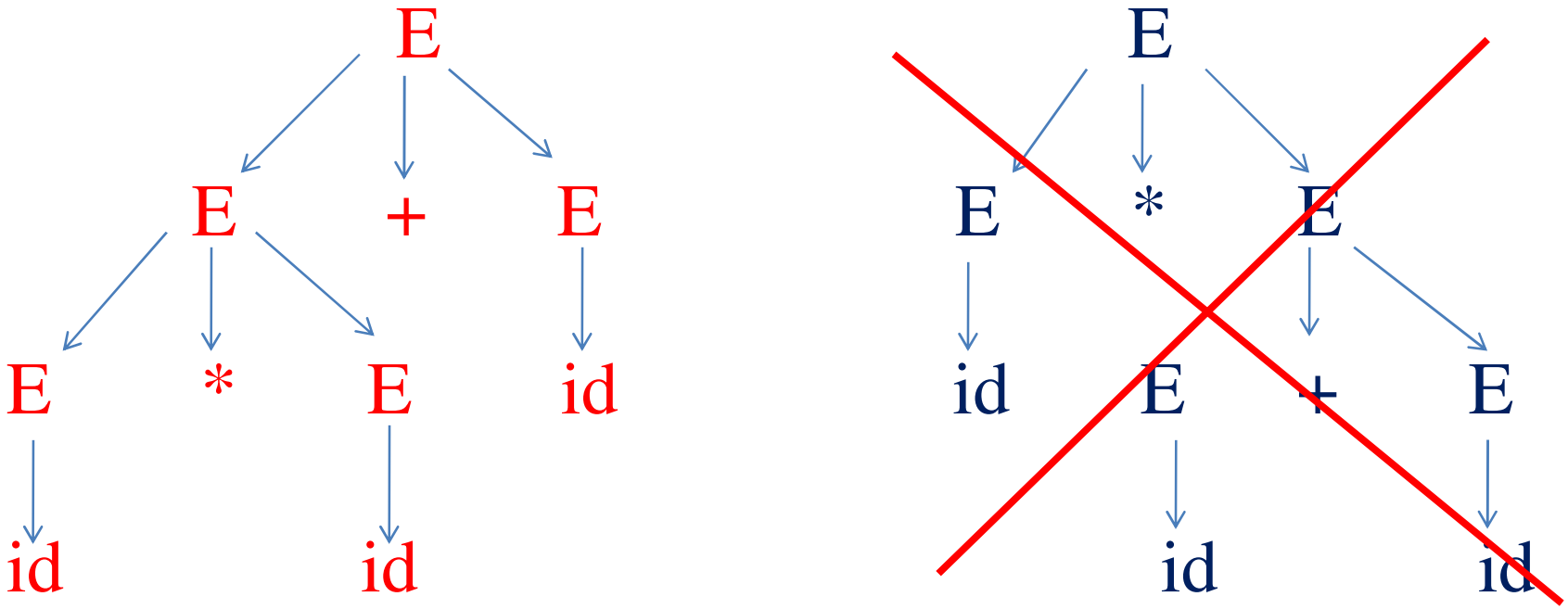
E' → id * E' → id*id*E'→
id*id*id*E' → id*…*id
Handles *

How it works

- The grammar has two separate groups of productions
- All the pluses (+) must be generated before any of the times (*); so time(s) (*) are nested more deeply inside the parse tree; pluses (+) are generated in the outermost levels and times (*) are generated inside the pluses (+).
- So the grammar enforces that the time (*) has higher precedence than the plus (+)

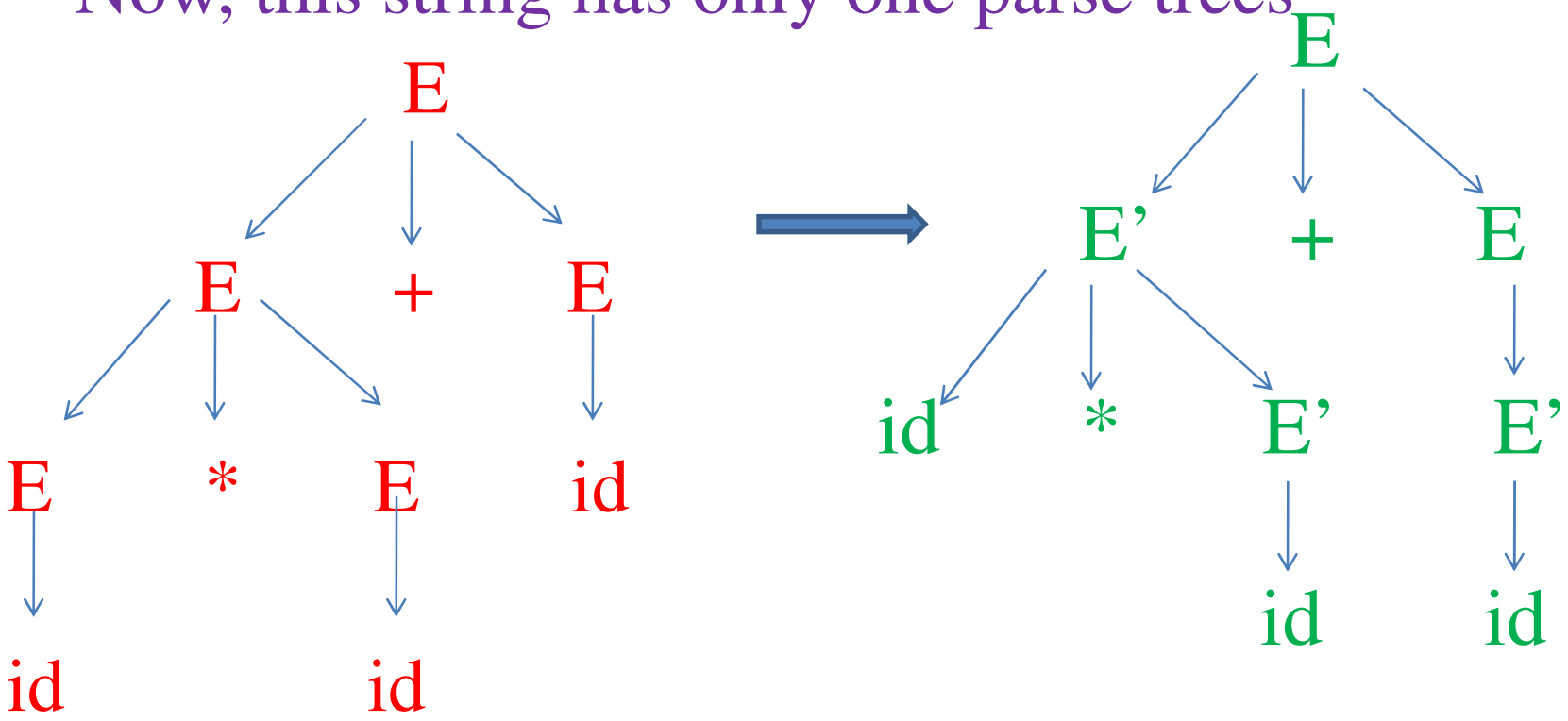
Ambiguity

- Grammar: E→ E + E | E * E | (E) | id
- String : id*id+id
- This string has two parse trees



Ambiguity

- Grammar: E→ E + E | E * E | (E) | id
- String : id*id+id
- Now, this string has only one parse trees



Example

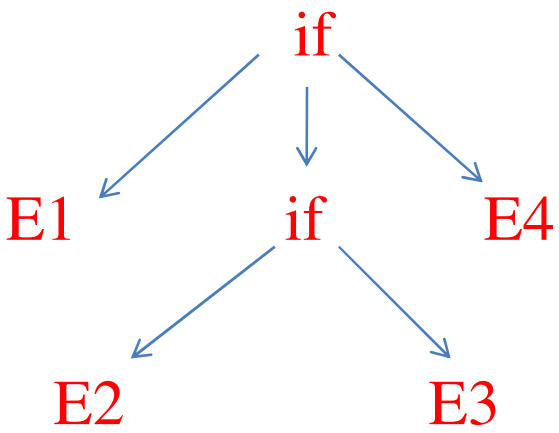
- Another Expression

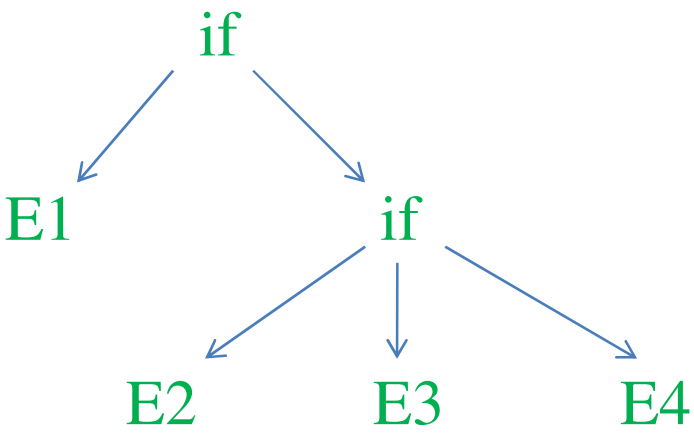
  E → if E then E

  | *if E then E else E*

  | *OTHER*

- The expression: *if E1 then if E2 then E3 else E4* has two separate parse trees





How to handle the Ambiguity

- The property that we want is: *else matches the closest unmatched then*
- Can be resolved as:

E → MIF| UIF

MIF → if E then MIF else MIF

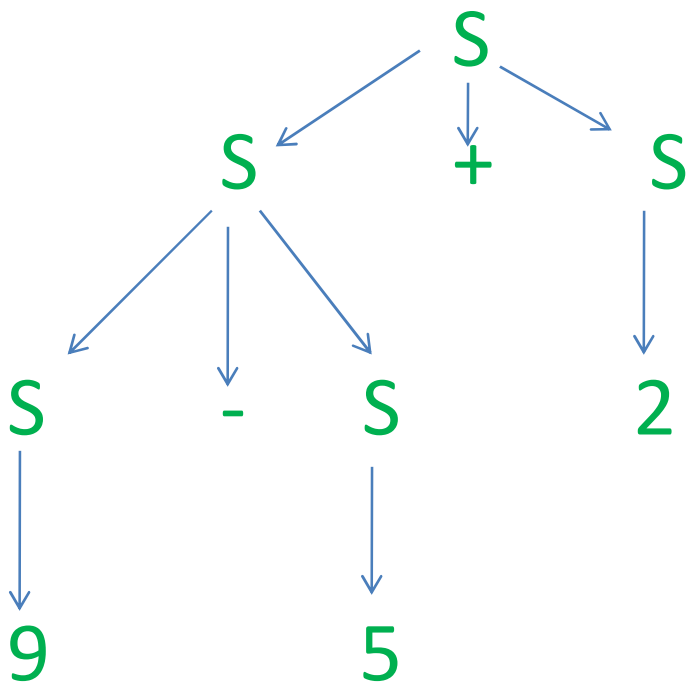
| OTHER

UIF → if E then E

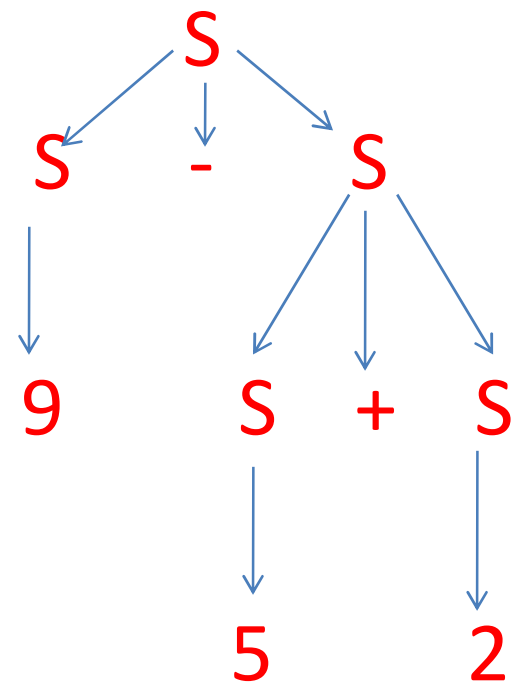
| if E MIF else UIF

Associativity of Operators

- Let a Grammar : S → S + S | S – S | 0 – 9
- Input: 9 – 5 + 2



(9-5)+2 = 6



9-(5+2) = 2

How to handle

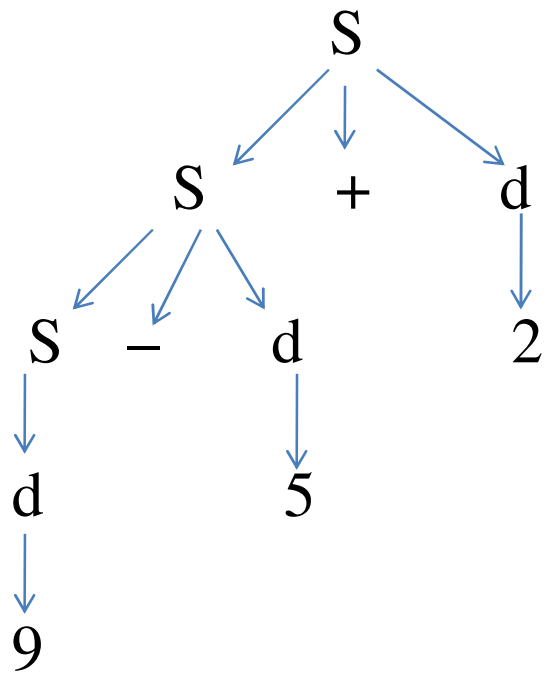
- When an operand like 5 has operators to its left and right, conventions are needed for deciding which operator applies to that operand.

- Rule: operator + is associates to the left. Arithmetic operators (+, -, *, /) are left associative.

- Assignment operator ("=") in C is right associative.

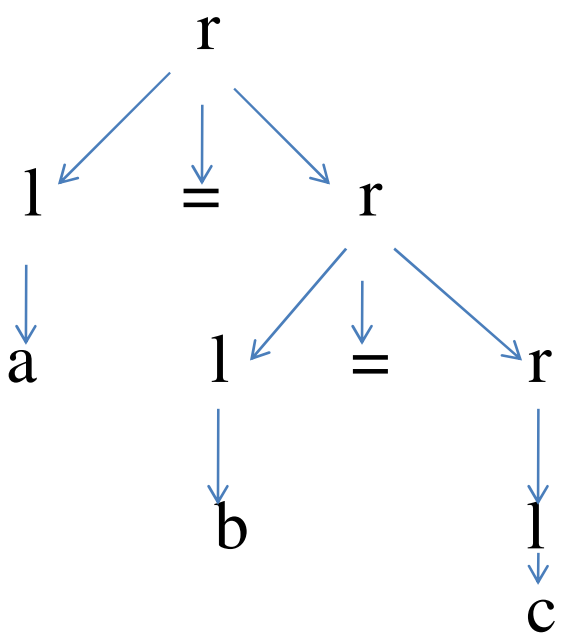
How to handle

- Left Associative
  $S \rightarrow S + d \mid S - d \mid d$
  $d \rightarrow [0\text{-}9]$
- Input: 9 – 5 + 2



Left-most derivation

- Right Associative
  $r \rightarrow l = r \mid l$
  $l \rightarrow [a\text{-}z]$
- Input: a = b = c



Right-most derivation

Push Down Automata

- The language for balanced parenthesis, i.e. $\{(^i)^i \mid i \geq 0\}$, CFG productions are

  $S \rightarrow (S) \mid S$

  $S \rightarrow \varepsilon$

- The situation can be handled if the DFA is augmented with *memory*
  - Memory implemented as a *stack*
  - Such an automata is called *Push Down Automata (PDA)*

Thanks