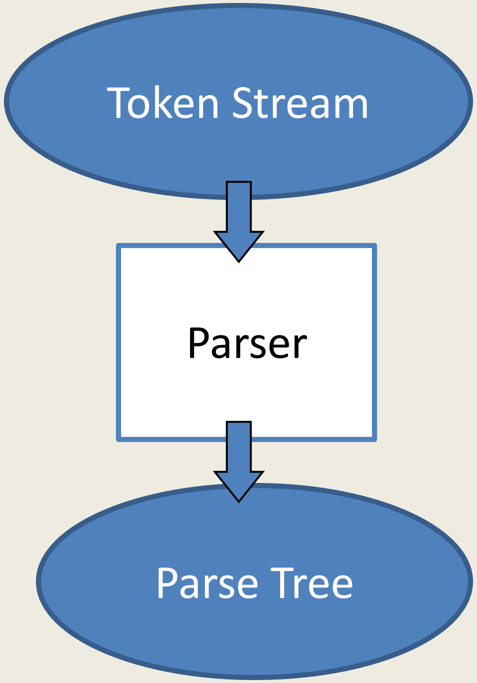
Syntax Analyzer (Parser)

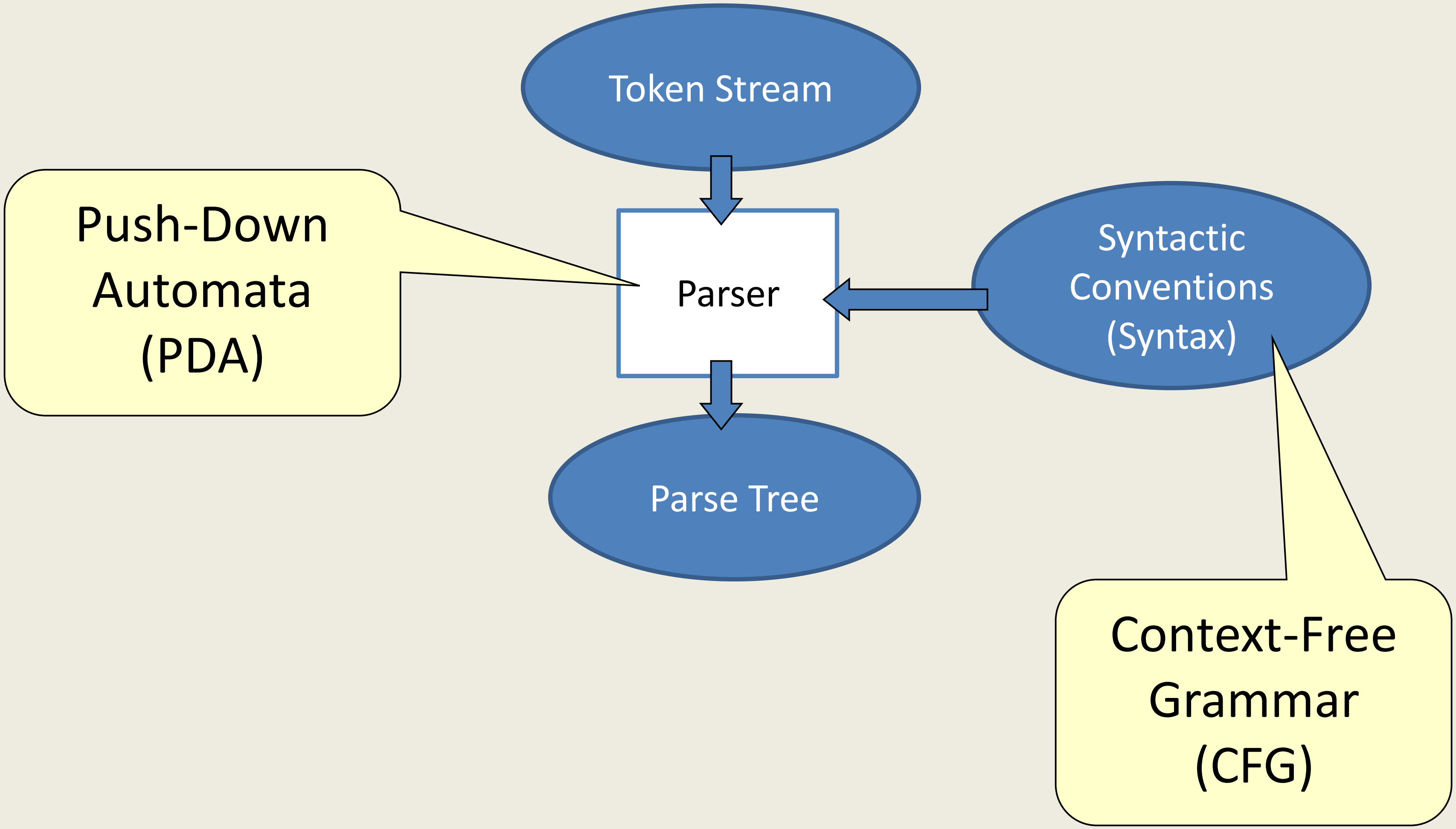
Dinesh Gopalani
dgopalani.cse@mnit.ac.in

Syntax Analyzer (Parser)

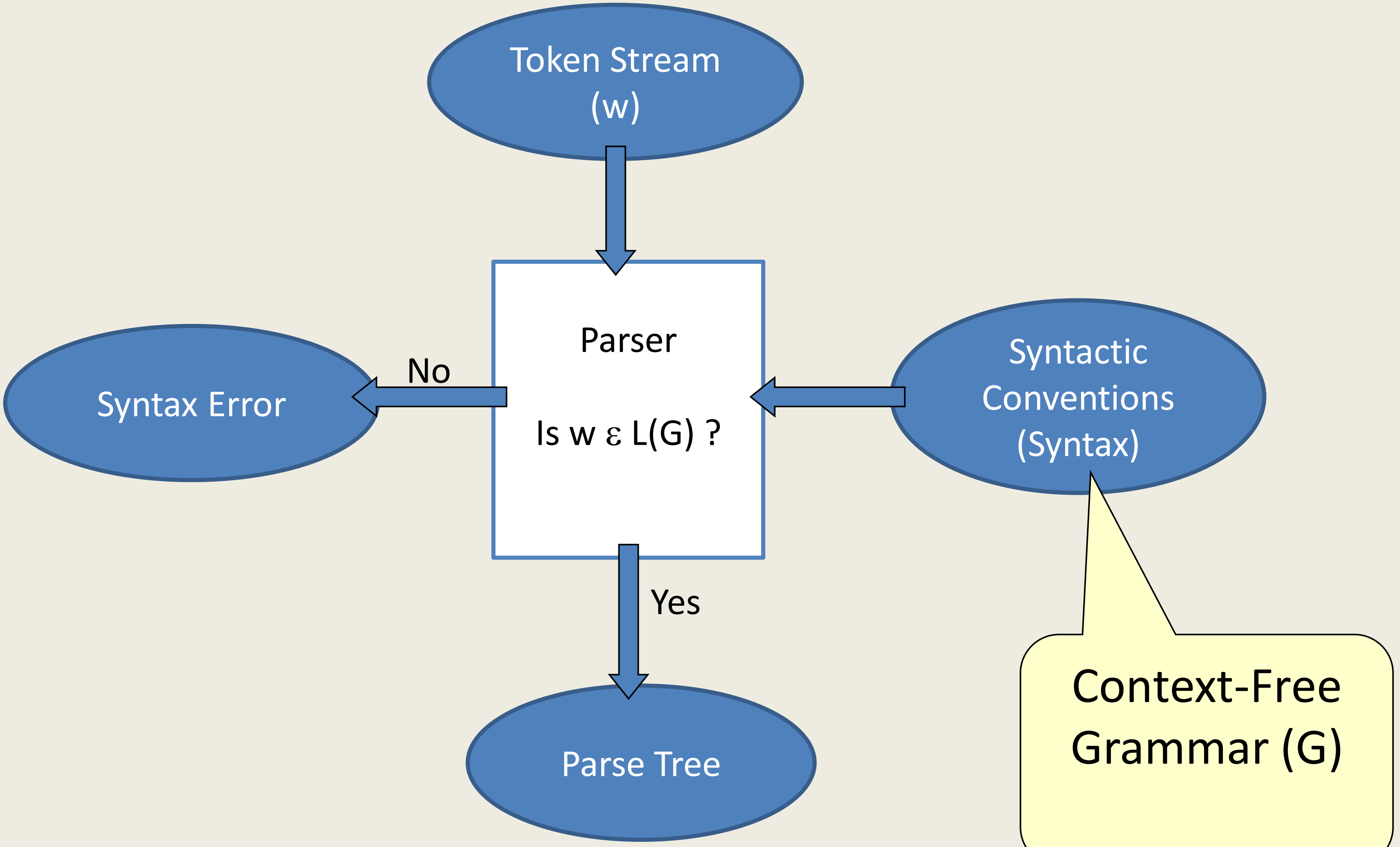
Forms the group of tokens that logically belong together – Syntactic Structures



How a Parser works?



How a Parser works?



Context Free Grammar (CFG)

A CFG is represented by (V_N, Σ, P, S)

- V_N is a finite non-empty set of Non-terminals (Variables).
- Σ is a finite non-empty set of Terminals (Tokens).
- P is a finite set of Productions of the form

$$A \rightarrow \alpha$$

where $A \in V_N$

and $\alpha \in (V_N \cup \Sigma)^*$

- S is a special variable called the Start symbol.

Context Free Grammar (CFG)

Notations:

- a, b, c, … denote terminals (Σ)
- A, B, C, …, A_1, A_2, … denote variables (V_N)
- w, x, y, z, … denote strings of terminals (Σ^*)
- W, X, Y, Z, … denote grammar symbols ($V_N \cup \Sigma$)
- α, β, γ, … denote strings of grammar symbols ($V_N \cup \Sigma$)*

Context Free Grammar (CFG)

- Derivation:

  If $A \rightarrow \alpha$ is a production in a grammar G and β, γ are any two strings of grammar symbols then

  $$\beta A \gamma \Rightarrow \beta \alpha \gamma$$

  This process is called One-step Derivation.

- Derivation in multiple steps is represented as:

  $$\overset{*}{\Rightarrow}$$

  If $\alpha \Rightarrow \beta_1 \Rightarrow \beta_2 \; \ldots \; \Rightarrow \beta$ then

  $$\alpha \overset{*}{\Rightarrow} \beta$$

Context Free Grammar (CFG)

- Leftmost Derivation:

  A derivation $S \overset{*}{\Rightarrow} w$ is called a Leftmost derivation if we apply a production only to the leftmost variable at every step.

- Rightmost Derivation:

  A derivation $S \overset{*}{\Rightarrow} w$ is called a Rightmost derivation if we apply a production only to the rightmost variable at every step.

Context Free Grammar (CFG)

- Example:

  Grammar G:

  $E \rightarrow E + E$

  $E \rightarrow E - E$

  $E \rightarrow id$

- Leftmost Derivation:

  $E \Rightarrow E - E \Rightarrow E + E - E \Rightarrow id + E - E \Rightarrow id + id - E \Rightarrow id + id - id$

- Rightmost Derivation:

  $E \Rightarrow E - E \Rightarrow E - id \Rightarrow E + E - id \Rightarrow E + id - id \Rightarrow id + id - id$

Context Free Grammar (CFG)

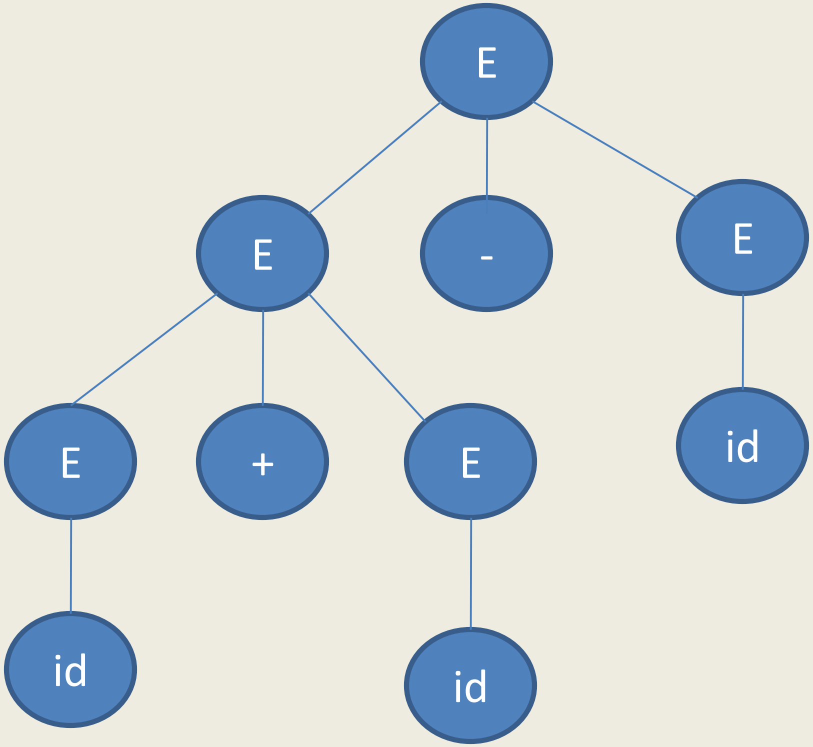
Derivation Tree:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

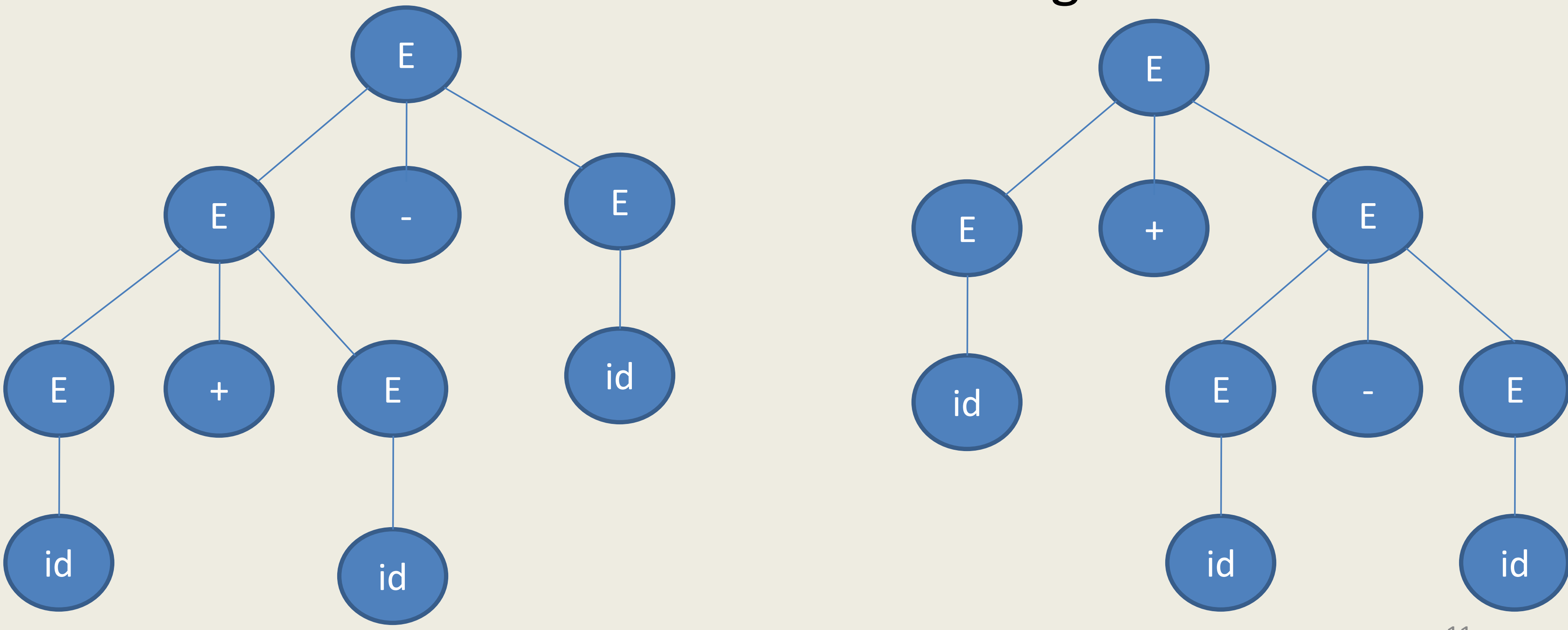
$E \rightarrow id$



Context Free Grammar (CFG)

Ambiguity:

A grammar that produces more than one derivation trees for the same sentence is ambiguous.



Context Free Grammar (CFG)

- Language L(G):

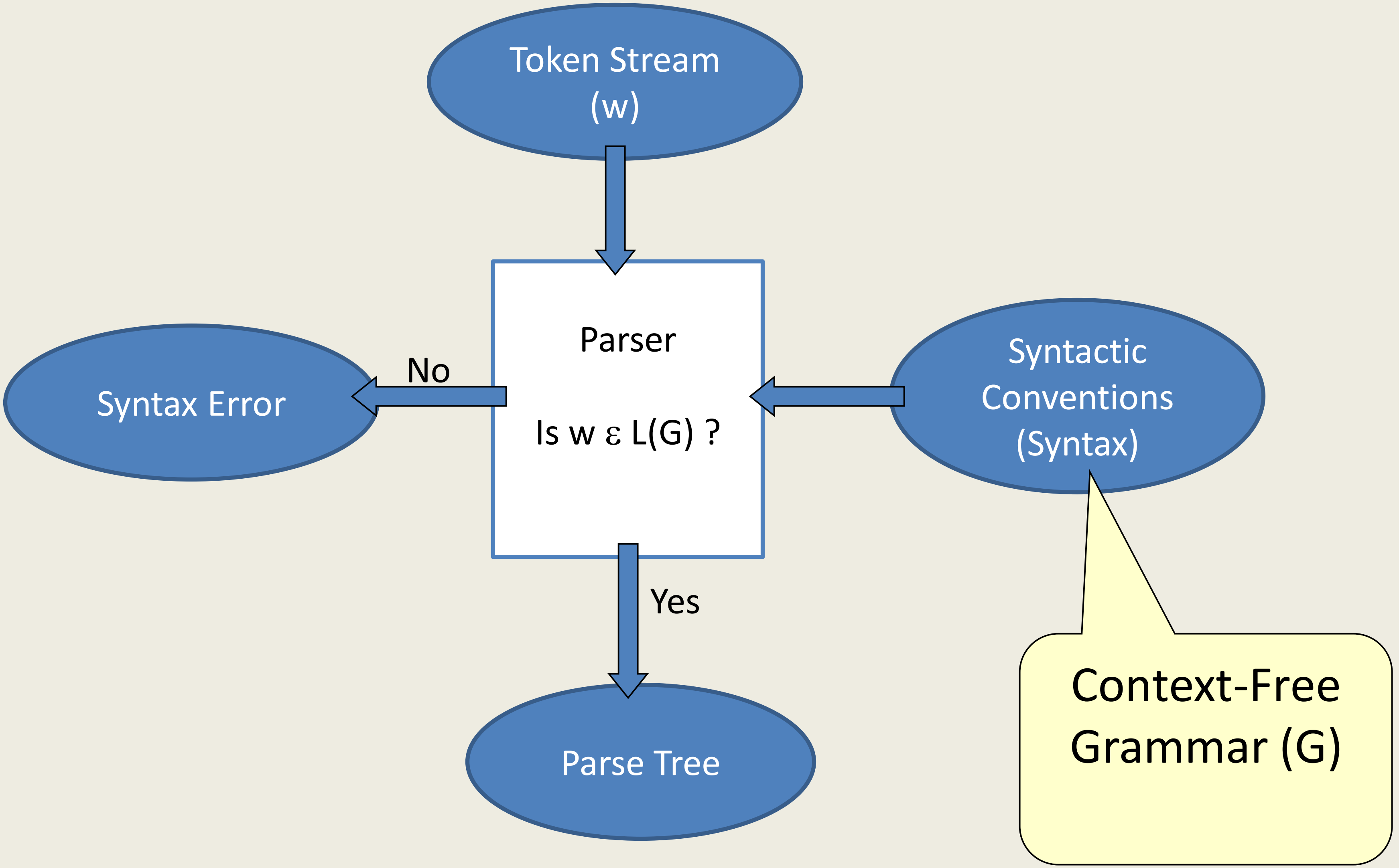
  The language generated by a grammar G is defined as

  $\{ w \mid w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w \}$

- The elements or strings of L(G) are called Sentences.
- L(G) is the set of all terminal strings derived from the start symbol S of grammar G.
- Example: $G : S \rightarrow a\,S\,b \mid ab$

  $L(G) = \{a^n\, b^n \mid n >= 1\}$
- If $S \overset{*}{\Rightarrow} \alpha$ then α is called a Sentential form.

How a Parser works?



Parser

A Parser for grammar G is a program that takes as input a string w and produces as output either a parse tree for w, if w is a sentence of L(G), or an error message indicating that w is not a sentence of L(G).

Parser

Two Approaches:

1. Bottom-Up

   Bottom-up parsers build parse trees from the bottom (leaves) to the top (root).

2. Top-Down

   Top-down parsers start with the top (root) and work down to bottom (leaves).

In both cases input is scanned from left to right.

Bottom-up Parsing

- It is a process of reducing a string w to the start symbol S of a grammar.
- At each step a substring matching the right-side of a production is replaced by the symbol on the left of that production – Handle.
- The process here may be viewed as one of finding and reducing handles.
- The parsing sequence will be exactly the reverse of rightmost derivation of w.

Bottom-up Parsing

Example:

$S \rightarrow a\ A\ c\ B\ e$

$A \rightarrow A\ b\ |\ b$

$B \rightarrow d$

Let $w = a\ b\ b\ c\ d\ e$

Bottom-up Parsing

Example:

$S \rightarrow a\ A\ c\ B\ e$

$A \rightarrow A\ b \mid b$

$B \rightarrow d$

Let $w = a\ b\ b\ c\ d\ e$

$\Downarrow A \rightarrow b$

$a\ A\ b\ c\ d\ e$

Bottom-up Parsing

Example:

$S \rightarrow a\ A\ c\ B\ e$

$A \rightarrow A\ b\ |\ b$

$B \rightarrow d$

Let $w = a\ b\ b\ c\ d\ e$

$\Downarrow A \rightarrow b$

$a\ A\ b\ c\ d\ e$

$\Downarrow A \rightarrow A\ b$

$a\ A\ c\ d\ e$

Bottom-up Parsing

Example:

$S \to a\ A\ c\ B\ e$

$A \to A\ b \mid b$

$B \to d$

Let $w = a\ b\ b\ c\ d\ e$

$\Downarrow A \to b$

$a\ A\ b\ c\ d\ e$

$\Downarrow A \to A\ b$

$a\ A\ c\ d\ e$

$\Downarrow B \to d$

$a\ A\ c\ B\ e$

Bottom-up Parsing

Example:

$S \rightarrow a\ A\ c\ B\ e$

$A \rightarrow A\ b \mid b$

$B \rightarrow d$

Let $w = a\ b\ b\ c\ d\ e$

$\Downarrow\ A \rightarrow b$

$a\ A\ b\ c\ d\ e$

$\Downarrow\ A \rightarrow A\ b$

$a\ A\ c\ d\ e$

$\Downarrow\ B \rightarrow d$

$a\ A\ c\ B\ e$

$\Downarrow\ S \rightarrow a\ A\ c\ B\ e$

S

Bottom-up Parsing

Example:

$S \rightarrow a\ A\ c\ B\ e$

$A \rightarrow A\ b \mid b$

$B \rightarrow d$

Let $w = a\ b\ b\ c\ d\ e$

$\Downarrow$ $A \rightarrow b$

$a\ A\ b\ c\ d\ e$

$\Downarrow$ $A \rightarrow A\ b$

$a\ A\ c\ d\ e$

$\Downarrow$ $B \rightarrow d$

$a\ A\ c\ B\ e$

$\Downarrow$ $S \rightarrow a\ A\ c\ B\ e$

S

Rightmost Derivation : $S \Rightarrow a\ A\ c\ B\ e \Rightarrow a\ A\ c\ d\ e \Rightarrow a\ A\ b\ c\ d\ e \Rightarrow a\ b\ b\ c\ d\ e$

Bottom-up Parsing

It is a process of reducing a string w to the start symbol S of a grammar by traversing in the reverse of rightmost derivation of w.

$$S\ (\gamma_0) \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n\ (w)$$

Here the handle β_n is located in γ_n and replace β_n by left side of appropriate production to get right-sentential form γ_{n-1}. Next handle β_{n-1} is located in γ_{n-1} and replace β_{n-1} by left side of appropriate production to get right-sentential form γ_{n-2}. The process is repeated until we get S.

Handle in Bottom-up Parsing

A Handle of a right-sentential form γ_i is a production $A \rightarrow \beta_i$ and a position of γ_i where the string β_i may be found and replaced by A to produce the previous right-sentential form γ_{i-1} in a rightmost derivation of γ_i

If $S \overset{*}{\Rightarrow} \overbrace{\alpha\, A\, x}^{\gamma_{i-1}} \Rightarrow \overbrace{\alpha\, \beta_i\, x}^{\gamma_i} \overset{*}{\Rightarrow} w$

then $A \rightarrow \beta_i$ in the position following α is a handle of $\alpha\, \beta_i\, x$

Shift-Reduce Parser

- This is simplest to implement and based on bottom-up approach.
- The parser uses a stack and input buffer.
- It operates by shifting input symbols onto the stack until a handle β is on top of the stack.
- The parser then reduces β to the left-side of the appropriate production.
- The parser then repeats this cycle until the stack has the start symbol and the input is empty (Accepted) or detected an error (rejected).

Shift-Reduce Parser

Example: G: $E \rightarrow E + E \mid E - E \mid id$

<u>Stack</u>	<u>Input</u>	<u>Action</u>
$	id+id-id$	Shift

Shift-Reduce Parser

Example: G: $E \rightarrow E + E \mid E - E \mid id$

Stack	Input	Action
\$	id+id-id\$	Shift
\$id	+id-id\$	Reduce($E \rightarrow id$)

Shift-Reduce Parser

Example: G: $E \rightarrow E + E \mid E - E \mid id$

Stack	Input	Action
$	id+id-id$	Shift
$id	+id-id$	Reduce($E \rightarrow id$)
$E	+id-id$	Shift

Shift-Reduce Parser

Example: G: $E \rightarrow E + E \mid E - E \mid id$

Stack	Input	Action
$	id+id-id$	Shift
$id	+id-id$	Reduce($E \rightarrow id$)
$E	+id-id$	Shift
$E+	id-id$	Shift
$E+id	-id$	Reduce($E \rightarrow id$)
$E+E	-id$	Reduce($E \rightarrow E + E$)
$E	-id$	Shift
$E-	id$	Shift
$E-id	$	Reduce($E \rightarrow id$)
$E-E	$	Reduce($E \rightarrow E - E$)
$E	$	Accepted

Shift-Reduce Parser

Parse Tree Construction:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$

Shift-Reduce Parser

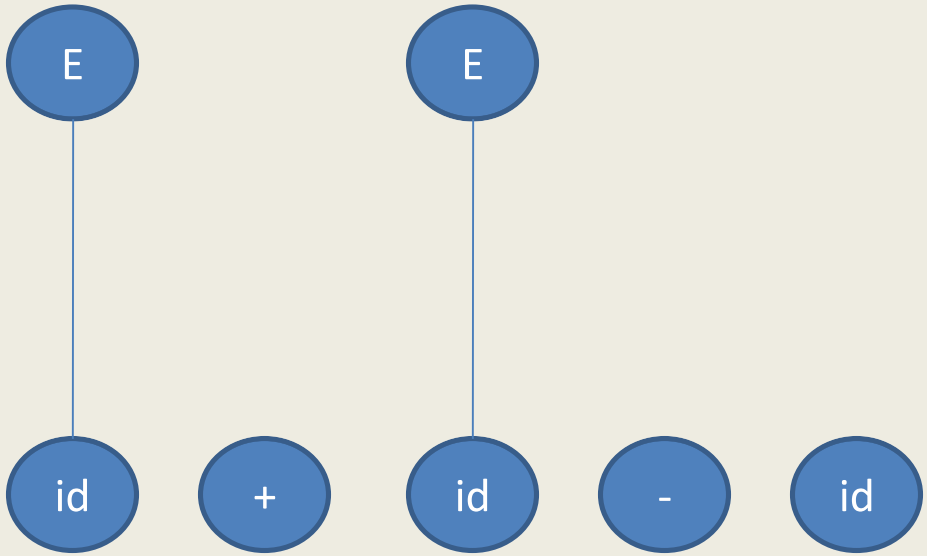
Parse Tree Construction:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$



Shift-Reduce Parser

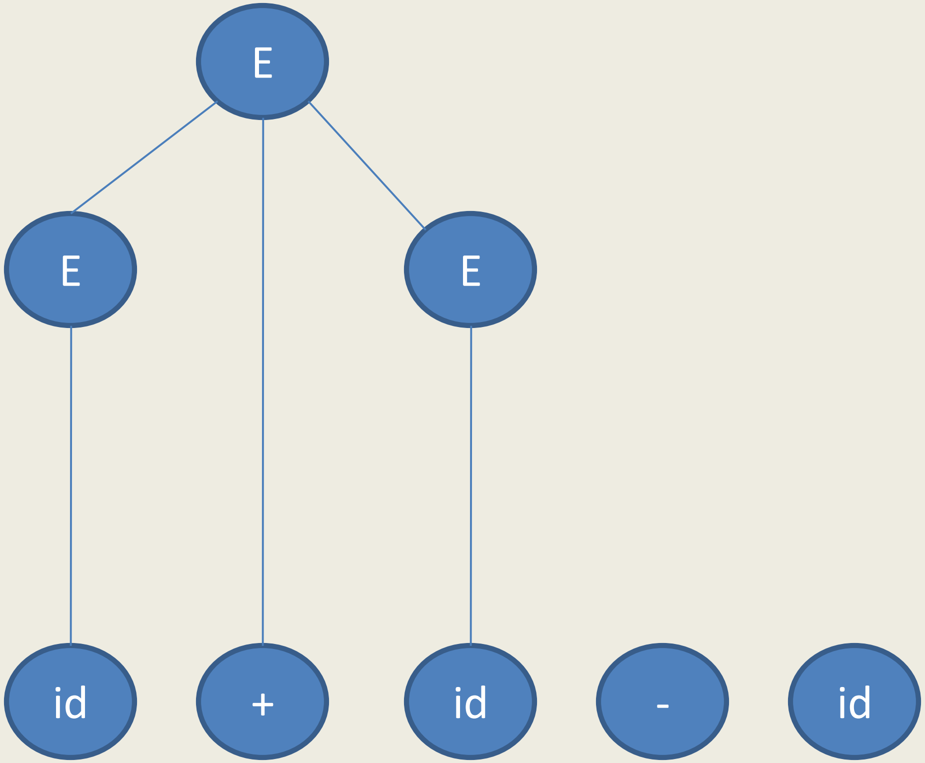
Parse Tree Construction:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$



Shift-Reduce Parser

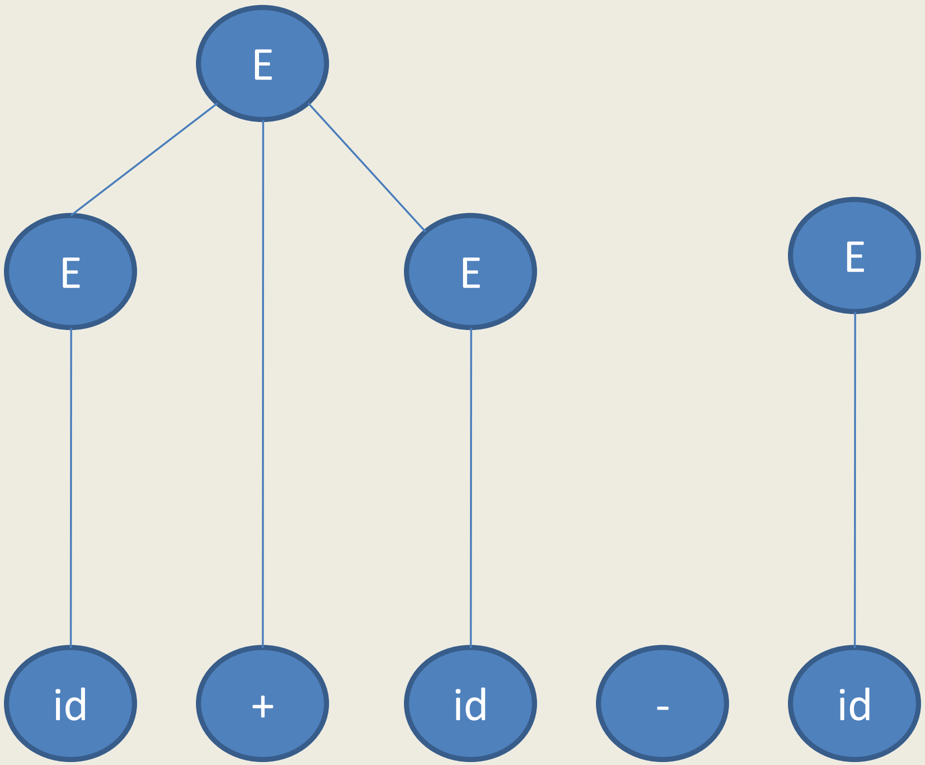
Parse Tree Construction:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$



Shift-Reduce Parser

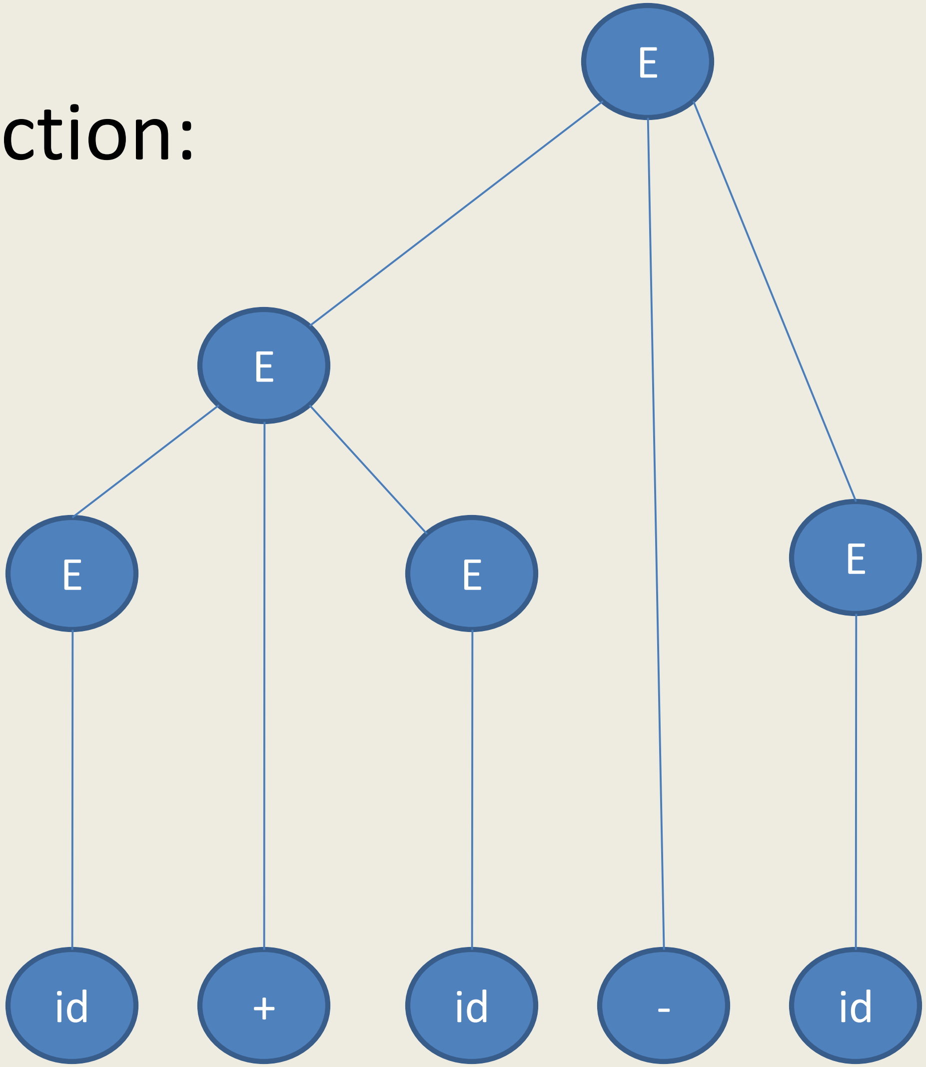
Parse Tree Construction:

Grammar G:

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow id$



Limitations of Shift-Reduce Parser

- May not work for many grammars.
- A substring on top of the stack may match with the right-side of a production but may not be the handle at this point of time.
- When the handle β is on top of the stack, it reduces β to the left-side of the appropriate production but what if

$$A \to \beta$$

$$B \to \beta$$

$$\ldots$$

Limitations of Shift-Reduce Parser

Example: G: $S \rightarrow a\ A\ c$

$A \rightarrow b \mid bA$

Stack	Input	Action
\$	abbc\$	Shift
\$a	bbc\$	Shift
\$ab	bc\$	Reduce($A \rightarrow b$)
\$aA	bc\$	Shift
\$aAb	c\$	Reduce($A \rightarrow b$)
\$aAA	c\$	Shift
\$aAAc	\$	Error

Operator-Precedence Parser

- This is also based on bottom-up approach.
- Used primarily for Expressions.
- Applicable for a small but important class of grammars – Operator Grammar.
- Operator Grammar:

1. ^ is not allowed on the RHS of any production.
2. Two or more adjacent non-terminals are not allowed on the RHS of any production.

Example: $E \rightarrow E\ A\ E \mid id$ $\quad A \rightarrow + \mid -$

$E \rightarrow E + E \mid E - E \mid id$

Operator-Precedence Parser

- The parser uses precedence relation between pair of terminals to decide whether Shift or Reduce action is to be performed.
- Precedence Relations:

1. $a < b$ a gives precedence to b
2. $a = b$ a has the same precedence as b
3. $a > b$ a takes precedence over b

Operator-Precedence Parser

Precedence Relations are defined using precedence and associativity rules:

- If operator θ_1 has higher precedence than operator θ_2

  $\theta_1 > \theta_2$ and $\theta_2 < \theta_1$

  Example: $* > +$

  $+ < *$

- If operators θ_1 and θ_2 are of equal precedence, then

  $\theta_1 > \theta_2$ and $\theta_2 > \theta_1$ if operators are left-associative

  or $\theta_1 < \theta_2$ and $\theta_2 < \theta_1$ if operators are right-associative

  Example: $+ > -$

  $- > +$

  $+ > +$

  $- > -$

Operator-Precedence Parser

Precedence Relations are defined using precedence and associativity rules:

- For an identifier id and an operator θ

  id > θ and θ < id

- For an operator θ and special symbol \$

  θ > \$ and \$ < θ

Operator-Precedence Parser

Precedence Relations are defined using precedence and associativity rules:

$$G: \; E \rightarrow E + E \mid E * E \mid id$$

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

Precedence Relations are defined using precedence and associativity rules:

$$G : E \rightarrow E + E \mid E * E \mid id$$

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>	Error	>
$	<	<	<	Accept

Operator-Precedence Parser

Working:

Precedence relation between topmost terminal symbol on Stack and current input symbol is checked

If < or = then Shift Action

otherwise (>) Reduce Action

Operator-Precedence Parser

$G: E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

$G: \; E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift
$id	+id*id$	> , Reduce($E \rightarrow id$)

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

$G: \; E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	$<$, Shift
$id	+id*id$	$>$, Reduce($E \rightarrow id$)
$E	+id*id$	$<$, Shift

	+	*	id	$
+	$>$	$<$	$<$	$>$
*	$>$	$>$	$<$	$>$
id	$>$	$>$		$>$
$	$<$	$<$	$<$	

Operator-Precedence Parser

$G: E \to E + E \mid E * E \mid id$

Stack	Input	Relation and Action
\$	id+id*id\$	< , Shift
\$id	+id*id\$	> , Reduce($E \to id$)
\$E	+id*id\$	< , Shift
\$E+	id*id\$	< , Shift

	+	*	id	\$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
\$	<	<	<	

Operator-Precedence Parser

$G: E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift
$id	+id*id$	> , Reduce($E \rightarrow id$)
$E	+id*id$	< , Shift
$E+	id*id$	< , Shift
$E+id	*id$	> , Reduce($E \rightarrow id$)

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

$G: \; E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift
$id	+id*id$	> , Reduce($E \rightarrow id$)
$E	+id*id$	< , Shift
$E+	id*id$	< , Shift
$E+id	*id$	> , Reduce($E \rightarrow id$)
$E+E	*id$	< , Shift

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

$$G: E \rightarrow E + E \mid E * E \mid id$$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift
$id	+id*id$	> , Reduce(E → id)
$E	+id*id$	< , Shift
$E+	id*id$	< , Shift
$E+id	*id$	> , Reduce(E → id)
$E+E	*id$	< , Shift
$E+E*	id$	< , Shift

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

$G: E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id*id$	< , Shift
$id	+id*id$	> , Reduce(E → id)
$E	+id*id$	< , Shift
$E+	id*id$	< , Shift
$E+id	*id$	> , Reduce(E → id)
$E+E	*id$	< , Shift
$E+E*	id$	< , Shift
$E+E*id	$	> , Reduce(E → id)
$E+E*E	$	> , Reduce(E → E * E)
$E+E	$	> , Reduce(E → E + E)
$E	$	Accepted

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	

Operator-Precedence Parser

Another Example

$G: E \rightarrow E + E \mid E * E \mid id$

Stack	Input	Relation and Action
$	id+id+id$	< , Shift
$id	+id+id$	> , Reduce($E \rightarrow id$)
$E	+id+id$	< , Shift
$E+	id+id$	< , Shift
$E+id	+id$	> , Reduce($E \rightarrow id$)
$E+E	+id$	> , Reduce($E \rightarrow E + E$)
$E	+id$	< , Shift
$E+	id$	< , Shift
$E+id	$	> , Reduce($E \rightarrow id$)
$E+E	$	> , Reduce($E \rightarrow E + E$)
$E	$	Accepted

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>		>
$	<	<	<	