SYNTAX ANALYSIS

BOTTOM-UP PARSING

Bottom-Up Parsing

- Construction of Parse Tree for an input string beginning at leaves and working towards the root
- Can be visualized as reducing a string "w" to the start symbol of Grammar
- At each reduction step, a specific substring matching the body of production is replaced by Non-Terminal at the head of the production

Bottom-Up Parsing

E → E + T | T

T → T * F | F

F → (E) | **id**

To reduce the string and move towards root symbol

id * id ⇒ F * id

⇒ **T * id**

⇒ **T * F**

⇒ **T**

⇒ **E**

Now if we write above derivation from bottom to top, we get rightmost derivation.

Conclusion: Bottom-Up parsing during left to right scan of the input constructs a rightmost derivation in reverse

Handle Pruning

A Handle is a substring that matches the body of the production and whose reduction represents one step along the reverse of Rightmost Derivation

Right Sentential Form	Handle	Reducing Production
id1 * id2	id1	F → id
F * id2	F	T → F
T * id2	id2	F → id
T * F	T * F	T → T * F
T	T	E → T

E → E + T | T

T → T * F | F

F → (E) | **id**

Handle Pruning

Handle Pruning:

- **We start with a string of terminals w to be parsed**
- **If w is a sentence of the grammar, then let w =** γn

Where γn is the nth right sentential form of some unknown right derivation as

$$S \Rightarrow \gamma 0 \Rightarrow \gamma 1 \Rightarrow \gamma 2 \Rightarrow \dots \Rightarrow \gamma n\text{-}1 \Rightarrow \gamma n$$

- To reconstruct this derivation in reverse order, we locate the handle βn in γn by relevant head of the production A → βn
- The βn will be replaced by A to get previous right sentential form γn-1
- This process we called as Handle Pruning

Shift Reduce Parser

The Process:

Stack	Input
$	w $
…	
…	
$ S	$

- Initially stack is empty and string w is on the input
- The parser operates by shifting zero or more input symbols onto stack until a handle is on top of stack
- The parser then reduces the handle to the left side of the appropriate production
- The parser repeats this cycle until it has detected an error or until the stack contains the start symbol and the input is empty

Shift Reduce Parser

Actions of a Shift Reduce Parser:-

1. **Shift**: Shift the next input symbol onto the top of the stack.
2. **Reduce:**
   a. The right end of the string to be reduced must be at the top of the stack.
   b. Locate the left end of the string within the stack and decide with what non-terminal to replace the string.
3. **Accept**: Announce successful completion of parsing.
4. **Error:** Discover a syntax error and call an error recovery routine.

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id

Shift Reduce Parser

Input: id * id

E → E + T | T

T → T * F | F

F → (E) | id

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id
$ F	* id $	Reduce by T → F

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by $F \rightarrow id$
$ F	* id $	Reduce by $T \rightarrow F$
$ T	* id $	Shift

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id
$ F	* id $	Reduce by T → F
$ T	* id $	Shift
$ T *	id $	Shift

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by $F \rightarrow id$
$ F	* id $	Reduce by $T \rightarrow F$
$ T	* id $	Shift
$ T *	id $	Shift
$ T * id	$	Reduce by $F \rightarrow id$

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id
$ F	* id $	Reduce by T → F
$ T	* id $	Shift
$ T *	id $	Shift
$ T * id	$	Reduce by F → id
$ T * F	$	Reduce by T → T * F

Shift Reduce Parser

Input: id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id
$ F	* id $	Reduce by T → F
$ T	* id $	Shift
$ T *	id $	Shift
$ T * id	$	Reduce by F → id
$ T * F	$	Reduce by T → T * F
$ T	$	Reduce by E → T

Shift Reduce Parser

Input: id * id

E → E + T | T

T → T * F | F

F → (E) | id

Stack	Input	Action
$	id * id $	Shift
$ id	* id $	Reduce by F → id
$ F	* id $	Reduce by T → F
$ T	* id $	Shift
$ T *	id $	Shift
$ T * id	$	Reduce by F → id
$ T * F	$	Reduce by T → T * F
$ T	$	Reduce by E → T
$ E	$	Accept

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id

Shift Reduce Parser

Input: id + id * id

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid (E) \mid id$

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)
$ E + E *	id $	Shift

Shift Reduce Parser

Input: id + id * id

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow$ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)
$ E + E *	id $	Shift
$ E + E * id	$	Reduce by E → id

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)
$ E + E *	id $	Shift
$ E + E * id	$	Reduce by E → id
$ E + E * E	$	Reduce by E → E * E

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)
$ E + E *	id $	Shift
$ E + E * id	$	Reduce by E → id
$ E + $\underline{\text{E * E}}$	$	Reduce by E → E * E
$ E + E	$	Reduce by E → E + E

Shift Reduce Parser

Input: id + id * id

E → E + E | E – E | E * E | E / E | E ↑ **E | (E) | id**

Stack	Input	Action
$	id + id * id $	Shift
$ id	+ id * id $	Reduce by E → id
$ E	+ id * id $	Shift
$ E +	id * id $	Shift
$ E + id	* id $	Reduce by E → id
$ E + E	* id $	Shift (Here Shift-Reduce Conflict)
$ E + E *	id $	Shift
$ E + E * id	$	Reduce by E → id
$ E + $\underline{\text{E * E}}$	$	Reduce by E → E * E
$ E + E	$	Reduce by E → E + E
$ E	$	Accept