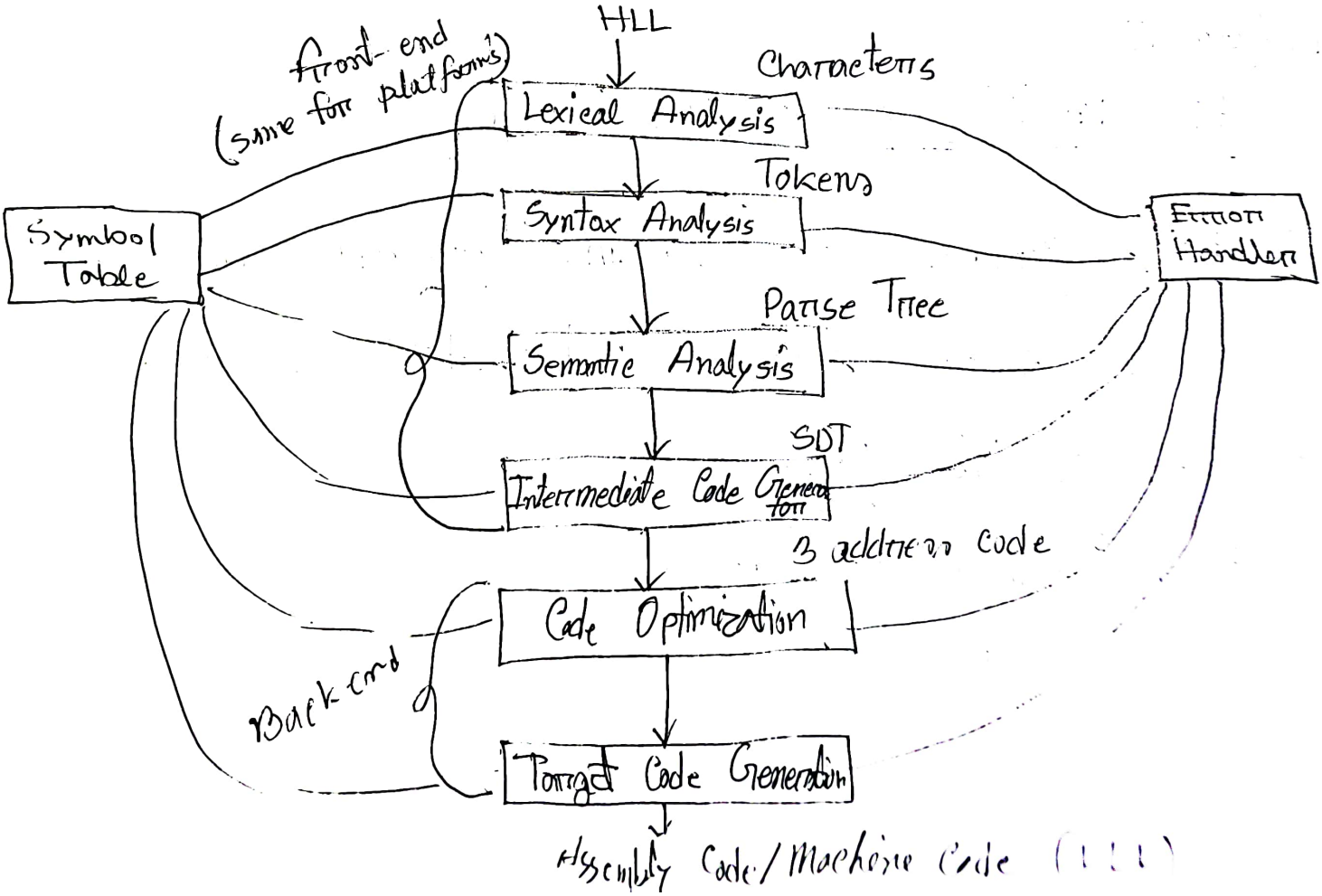
Compilers

12 Nov-23
6:25 PM

Compiler Design :

1. Lexical Analysis
2. Parsers (Syntax Analyzers) **
3. Semantic Analysis
4. Intermidiate Code generation
5. Code Optimization

Phases of Compilers

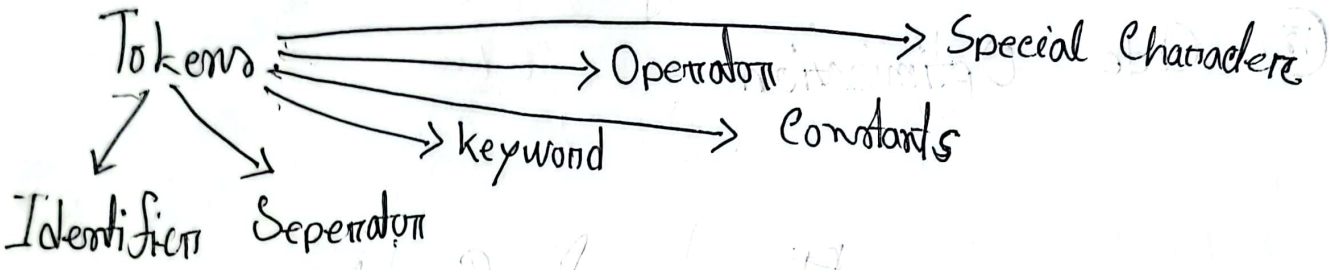


Lexical Analysis [Lexer, Tokenizer, Scanner]

① Tokenization

② Give Error messages → Exceeding length
→ Unmatched string
→ illegal character

③ Eliminate Comments, White Space
(Tab, new space, new line)



Lecture-5

Find first() in Compiler Design

First(A) Contains all terminals present in first place of every string derived by A.

1) S → abc | def | ghi
2) First (Terminal) = terminal
3) First (ε) = ε

Example

* $S \rightarrow ABC \mid ghi \mid jkl$ (a, b, c, g, j)
  $A \rightarrow a \mid b \mid c$ (a, b, c)
  $B \rightarrow b$ (b)
  $D \rightarrow d$ (d)

* $S \rightarrow ABC$ $(a, b, c, d, e, f, \varepsilon)$
  $A \rightarrow a \mid b \mid \varepsilon$ (a, b, ε)
  $B \rightarrow c \mid d \mid \varepsilon$ (c, d, ε)
  $C \rightarrow e \mid f \mid \varepsilon$ (e, f, ε)

* $E \rightarrow TE'$ $(id, ()$
  $E' \rightarrow * TE' \mid \varepsilon$ $(*, \varepsilon)$
  $T \rightarrow FT'$ $(id, ()$
  $T' \rightarrow \varepsilon \mid +FT'$ $(\varepsilon, +)$
  $F \rightarrow id \mid (E$ $(id, ()$

Follow(A) contains set of all terminals present immediate in right of 'A'.

Rules:

① Follow of start symbol in $

$Fo(A) = \{\$\}$

② $S \to ACD$

$C \to a|b$

$Fo(A) = First(C) = \{a, b\}$

$Fo(D) = follow(S) = \{\$\}$

③ $S \to aSbS \mid bSaS \mid \epsilon$

*follow never contain ϵ

$Fo(S) = b, a, \$$

$S \to ABC$

$A \to DEF$

$B \to \epsilon$

$C \to \epsilon$

$D \to \epsilon$

$E \to \epsilon$

$F \to \epsilon$

$Fo(A) = first(B)$

$= first(C)$

$= \$$

Examples

$S \to AaAb \mid BbBa$

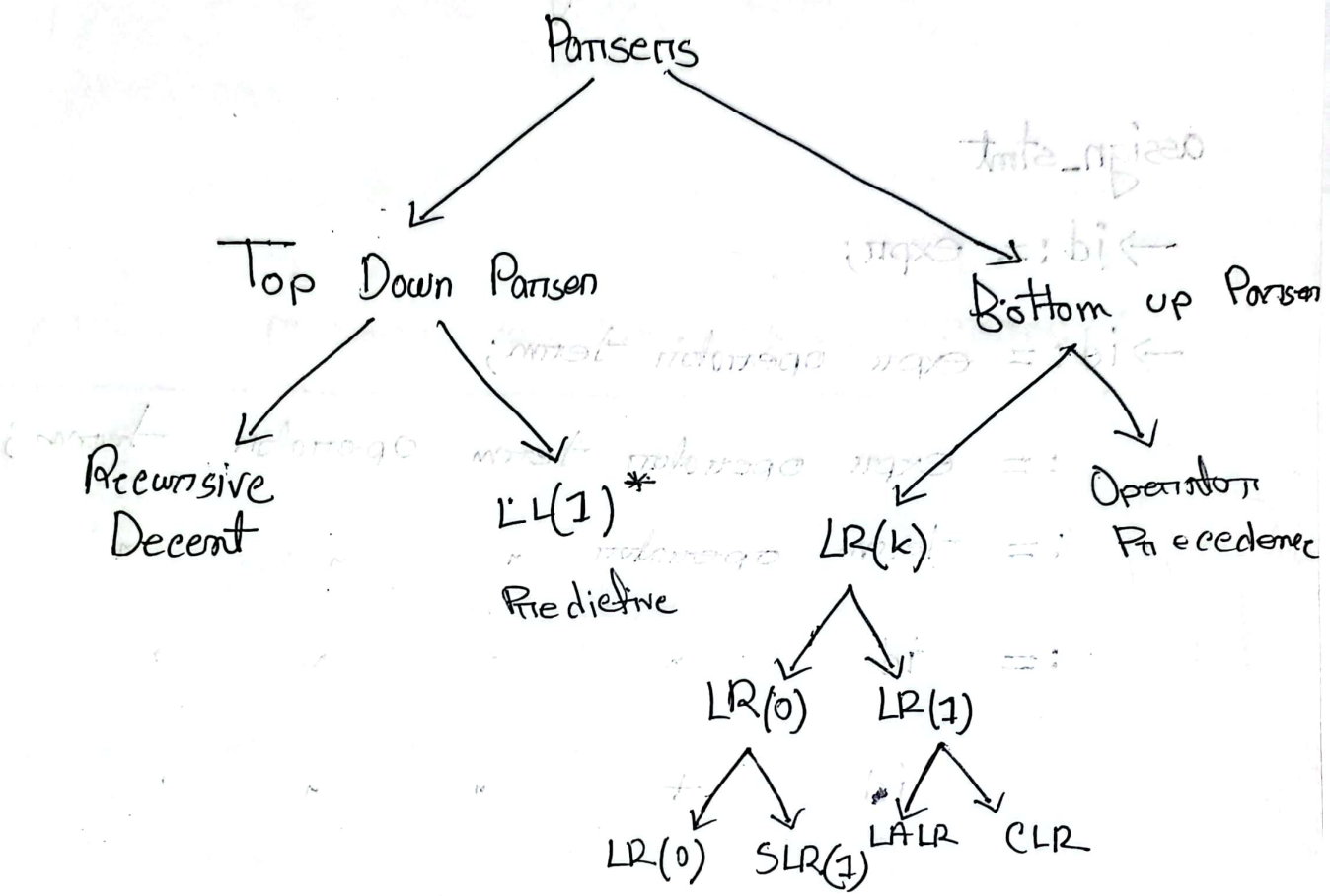
$A \to \epsilon$

$B \to \epsilon$

$Fo(A) = a, b$

$Fo(B) = b, a$

What is Parsing & Types of Parsers:



Parsing : It is a process of deriving string from a given grammar.

Lecture-05

derive

id := id + real - integer;

assign_stmt

→ id := expr;

→ id := expr operator term;

:= expr operator term operator term;

:= term operator " " "

:= id " " " "

id + " " "

id + real " "

id + real - integer

A Grammar is ambigious or not ?

* There is no algorithm to check it. You have to try hit and trial method.

Ambiguous Grammar to Unambiguous Grammar :

Elimination of left recursion :

~~$A \to A\alpha$~~

$A \to A\alpha \mid \beta$

$A \to \beta A'$

$A' \to \varepsilon \mid \alpha A'$

} removed left recursion

$A \to B$

$A \to A\alpha$

$B\alpha$

$A \to A\alpha$

$B\alpha\alpha$

$B\alpha^*$

~~Exam~~

$A \to A\alpha \mid B \quad \longrightarrow \quad \boxed{\begin{array}{l} A \to BA' \\ A' \to \varepsilon \mid \alpha A' \end{array}}$

Example

* $E \to E + T \mid T$

$A \quad A \quad \alpha \quad B$

$E \to TE'$

$E' \to \varepsilon \mid +TE'$

* $S \to SO \mid SOS \mid 01$

$A \quad A \quad \alpha \quad B$

$A \to 01S'$

$S' \to \varepsilon \mid 0 \mid SOSS'$

Example from Slide

E-6 | S-11

* $E \to E + T \mid T$

$A \quad A \quad \alpha \quad B$

$E \to TE'$

$E' \to \varepsilon \mid +TE'$ ✓

* $T \to T * F \mid F$

$A \quad A \quad \alpha \quad B$

$T \to FT'$

$T' \to \varepsilon \mid *FT'$ ✓

Top Down Parsing

(1) Recursive – Descent Parsing -

Input: aabbde

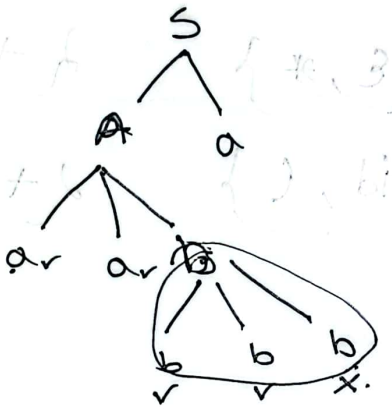
S → Aa | Ce

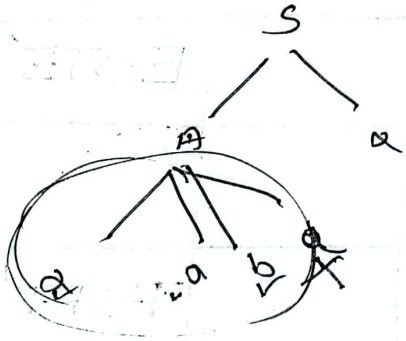
A → aaB | aaba

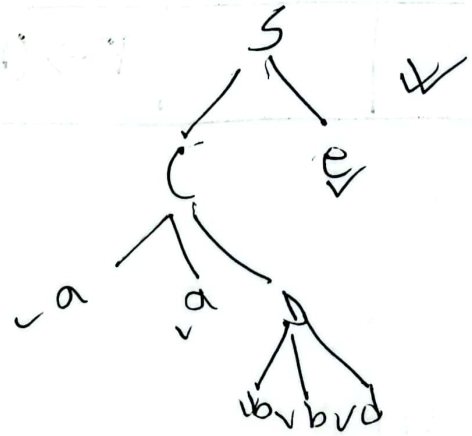
B → bbb

C → aaD

D → bbd







Predictive Parser {LL(1)}

	First	Follow
$E \rightarrow TE'$	{id, (}	{$,)}
$E' \rightarrow \varepsilon \mid +TE'$	{ε, +}	{$,)}
$T \rightarrow FT'$	{id, (}	~~{$,)}~~ {+, $,)}
$T' \rightarrow \varepsilon \mid *FT'$	{ε, *}	{+, $,)}
$F \rightarrow id \mid (E)$	{id, (}	{+, $,), *}

M	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Stack	input	production
$E	(id*id)+id$	E → TE′
$E′T	(id*id)+id$	T → FT′
$E′T′F	(id*id)+id$	F → (E)
$E′T′)E(	(id*id)+id$	POP
$E′T′)E	id*id)+id$	E → TE′
$E′T′)E′T	id*id)+id$	T → FT′
$E′T′)E′T′F	id*id)+id$	F → id
$E′T′)E′T′id	id*id)+id$	POP
$E′T′)E′T′	*id)+id$	T′ → *FT′
$E′T′)E′T′F*	*id)+id$	POP
$E′T′)E′T′F	id)+id$	F → id
$E′T′)E′T′id	id)+id$	POP
$E′T′)E′T′	)+id$	T′ → ε
$E′T′)E′	)+id$	E′ → ε
$E′T′)	)+id$	POP

Stack	Input	Production
$ E'T'	+id$	$T' \to \varepsilon$
$E'	+id$	$E' \to +TE'$
$E'T+	+id$	POP
$E'T	id$	$T \to FT'$
$E'T'F	id$	$F \to id$
$E'T'id	id$	POP
$E'T'	$	POP
$E'	$	POP
$	$	Accepted